PROCEEDINGS

3rd. symposium on small computers in the arts

OCTOBER 14-16 1983
PHILADELPHIA, PENNSYLVANIA

SPONSORED BY:
IEEE COMPUTER SOCIETY and IEEE PHILADELPHIA SECTION

ISBN 0-8186-0499-9
IEEE CATALOG NO. 83CH1930-7
LIBRARY OF CONGESS NO.83-81625
IEEE COMPUTER SOCIETY ORDER NO. 499

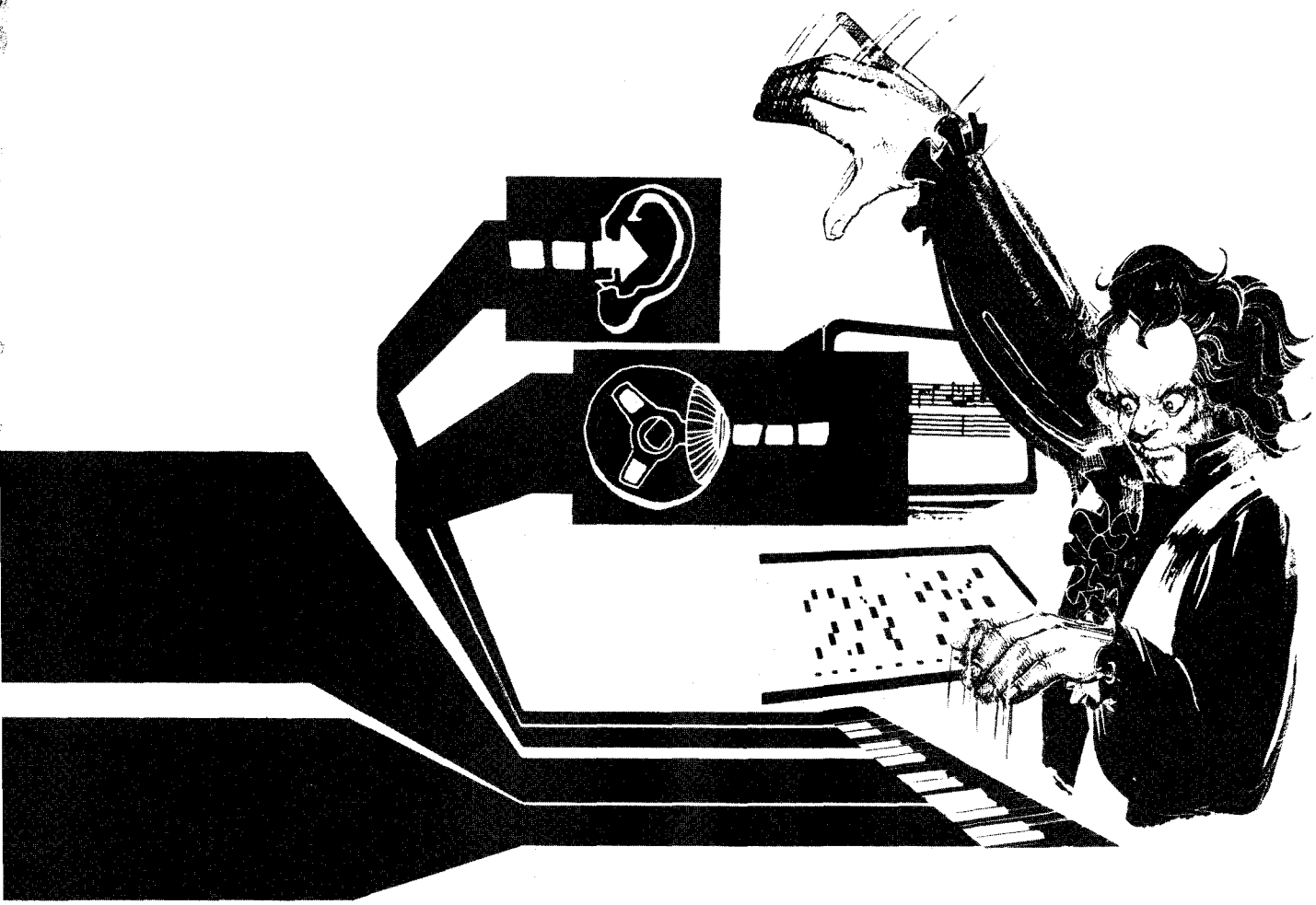
1884 1984 A CENTURY OF ELECTRICAL PROGRESS
THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

IEEE COMPUTER SOCIETY PRESS

PROCEEDINGS
3rd. symposium on small computers in the arts

OCTOBER 14-16 1983
PHILADELPHIA, PENNSYLVANIA

SPONSORED BY:
IEEE COMPUTER SOCIETY and IEEE PHILADELPHIA SECTION



1884 1984 A CENTURY OF ELECTRICAL PROGRESS
THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

IEEE COMPUTER SOCIETY PRESS

ISBN 0-8186-0499-9
IEEE CATALOG NO. 83CH1930-7
LIBRARY OF CONGESS NO.83-81625
IEEE COMPUTER SOCIETY ORDER NO. 499

Order from: IEEE Computer Society
Post Office Box 80452
Worldway Postal Center
Los Angeles, CA 90080

IEEE Service Center
445 Hoes Lane
Piscataway, NJ 08854

Published by IEEE Computer Society Press
1109 Spring Street
Suite 300
Silver Spring, MD 20910

ISBN 0-8186-0499-9 (paper)
ISBN 0-8186-4499-0 (microfiche)
ISBN 0-8186-8499-2 (casebound)
Library of Congress No. 83-81625
IEEE Catalog No. 83CH1930-7
IEEE Computer Society Order No. 499

Order from: IEEE Computer Society
Post Office Box 80452
Worldway Postal Center
Los Angeles, CA 90080

IEEE Service Center
445 Hoes Lane
Piscataway, NJ 08854

1884 1984 A CENTURY OF ELECTRICAL PROGRESS
The Institute of Electrical and Electronics Engineers, Inc.

1983

Proceedings of the Symposium on Small Computers in the Arts

October 14-16, 1983
Philadelphia

Sponsored by:

IEEE Computer Society
IEEE Philadelphia Section
Delaware Valley Chapter/SIGGRAPH

Organized and produced by:

Small Computers in the Arts Network

Proceedings Committee

Editors	Eric Podietz
	Dick Moberg
	Kerry Sherin
	Donna Mansfield
Layout	Dick Moberg
	Eric Podietz
Symposium Chairman	Dick Moberg

Historical Notes

The Symposium on Small Computers in the Arts grew out of a computer music concert held in downtown Philadelphia in 1978. It was planned as part of the Personal Computing '78 show held at the Civic Center. John Dilks, the founder of the show, graciously backed the idea and provided a hotel ballroom for the event. As word of the upcoming concert spread, we received calls from people as far away as the West Coast asking if they could participate. One musician from New York actually arranged a piece for computer and clarinet especially for the concert. The evening of the concert, over 500 persons showed up and tried to squeeze into a room that only held 300. The concert was recorded and an album made.

The success of that concert led the organizers to form an informal group to produce similar events and to act as a clearinghouse for those interested in computer applications in the arts. The 1979 Personal Computer Music Festival, sponsored by the group, included talks and demonstrations during the day in addition to the evening concert. In 1980, a separate day of computer graphics talks and demonstrations was added to make the Personal Computer Arts Festival. All these events were held at the Personal Computing shows in Philadelphia.

It had always been our desire to some day organize a major meeting solely dedicated to the use of small computers in the arts. This dream became a reality with the 1981 Symposium, thanks to the support of the IEEE Computer Society and the IEEE Philadelphia section.

At the 1981 and 1982 symposiums, 150 computer arts enthusiasts descended upon the Holiday Inn on the University of Pennsylvania campus. An informal and very interactive atmosphere prevailed making the symposium a great success with respect to its goals of increasing the participants' awareness of the uses of small computers in the arts. In May 1982, response to the symposium and succeeding interest led to the publications of Scan, a monthly newsletter on small computers in the arts.

The Small Computers in the Arts Network (formerly the Personal Computer Arts Group), as a volunteer not for profit group, continues to promote the use of computers in the arts through its newsletter, concerts, and other informal events. To contact the group, write to: Small Computers in the Arts Network, Box 1954, Philadelphia, Pennsylvania, 19105.

Table of Contents

Approaches to Computer Literacy and Training for Artists/Designers/Creatives

Renée LeWinter, Joan Shafran and Brian Wiffin

Massachusetts College of Art, Boston, MA
Jack F. Nolan, President; Marilyn Bever, Advisor

The design and implementation of a Comprehensive Computer Graphics and Computer Literacy Program has been seen as an essential addition to the existing course offerings at the Professional Art College. Within the Continuing Education Department and the Day School such courses have been in existence for several years. Through these experimental classes, the College and the Visual Technology Center have seen the need for expansion and growth in the area of Computer Graphics, and are beginning develop interesting course work and present a new methodology for learning approaches.

Introduction

Massachusetts College of Art remains the only publicly supported, professional arts college in the United States. It was founded to train art educators for the public schools and designers for industry. The undergraduate degrees offered are in Art Education, Critical Studies, Design, Fine Arts and Media and Performing Arts. Graduate Degrees are offered in Art Education and Fine Arts. There is also an extensive Continuing Education Program and a Visual Technology Center which is devoted to research in the area of High Technology and the Arts and offers classes and special programs to students and professionals in the Boston area.

Because of the commitment to a professional school atmosphere, the faculty and administration strive to bring the best and most useful curriculum to its students. With the increasing necessity of training in the high technology field, Mass. College of Art introduced courses that would be relevant to the designer and the artist going out into the market place. At present, the school has a permanent Apple lab: eleven 64K Apple II Plus, printers, digitizing camera, Polaroid output, tablets and lightpens. The school also relies on temporary loan of equipment.

Demographically the College is uniquely located near the high tech companies on Route 128, and has sought support from these companies. One result was a class given through the Visual Technology Center using a Camex page make-up system, and was aimed at Mass. Art faculty, students and professionals in the design field. The main objective was to acquaint the student with using a computer as a design tool and to gain some literacy. The College sees as one of its roles that of providing not only training but also keeping abreast of the changing job markets.

Classes this fall will include Introduction to Computer Programming, Computers and Visual Communication, Programming Computer Graphics, Computer Graphics Laboratory, and Advanced Computer Graphics Applications.

The Resistance/Fascination Factor

With the commitment to providing access to the creative, it soon becomes clear that a Pandora's Box, of sorts, has evolved. Through course work and curriculum development at Mass. Art, it has become apparent that the need for "finding out" about computers is strong within the local art community. However, along with the need to know comes a variety of fears, expectations and prejudices. Machine fear is the most common, and in some cases, actually touching the machine, fear that pressing the wrong button will do something horrible. Another very real fear is that these new technologies will generate loss of jobs.

Expectations and prejudices are prevalent, because of the "seduction" of the imagery (fascination factor) or the speed one can access software particularly in a Paint program. The student at first is awed by the ease of image making, but with further exposure discovers the limitations and complexity of the software. The new user, prejudiced by traditional methodologies and techniques, tries to apply them to the creative process rather than exploring new possibilities. The question of "Why can't I do this?" is a constant in a beginning class and it is simply based on the lack of understanding of what the technology can do. The confusion and deception arises with the immediate gratification of seeing something or the sense of drawing while you're thinking.

Although these problems are mainly geared toward using software not writing it, a general literacy of what the machine can do, or potentially do, is imperative. The instructor can then proceed to alleviating these fears and prejudices and begin to affect change and growth to the creator's 'psyche', by getting them to ask the questions

CH1930-7/83/0000/0001$01.00

"How, what and why."

There is also an inherent problem of different levels of comprehension. Although the creative person is often nontechnical, there are exceptions to this. It is often said the greatest problem with artists learning the computer as a medium is that the programming is linear and they cannot think that way. Along with a comprehensive approach to literacy, careful selection of equipment and analogies to traditional artforms are important. References to mediums such as printmaking become obvious, in terms of 'process' being used to create an end.

The Design Process
Traditional graphic elements such as "line, composition, space, form, shape" with the addition of "time" (because of the dynamic nature of the screen) form a foundation from which the new user can draw from and apply to the new media. These associations to known experiences cuts down much of the fear and learning is faster. It can be said also, that understanding graphic elements is helpful in designing comprehensive graphic software as well.

The Question of Aesthetics:
Establishing a New Criteria
When using microcomputers such as Apples, the question of image quality comes to bear. Because of our equipment, the image is unsophisticated and almost too colorful. To the eye of a designer or artist the image presents itself as crude and almost unworkable. The micro does have an aesthetic life of its own and often, because of the directness of the micro, students produce quality graphics once the comparison with other media or systems subsides. Because of the simplicity of the equipment, the student must confront directly the nature of what the image is composed of (in our case the PIXEL, or picture element) and use this as an integral part of their image.

The responsibility of the instructor and the educational institution is to supply and create an environment that allows for maximum understanding of many ways of creating images. The introduction to other applications such as music and performance and their interconnections is important. It is having the foundation from which to build that will allow a new aesthetic to form, whether it be programming or manipulating graphic software.

Some important questions to consider are 1) what do we mean by image quality (should this be dictated by the capabilities of the machine, should we encourage the thought that because it was generated with a computer it must be good; 2) what is the end product, hard copy, video, digital storage, and should the criteria that apply for analog output have application here; 3) the ephemeral quality of the computer image; 4) how the idea becomes an abstraction in its translation to digital data; 5) seeing the screen as a conceptual art form; 6) and still seeing the differences between design vs. fine arts.

Selection of Hardware and Software
When considering the selection of hardware and software to be used in a computer graphics curriculum, several choices must be made regarding dedicated versus all purpose systems; the use of canned software versus writing one's own and the establishment of a criterion and a purpose with respect to limited access and an attempt to create a broad base of minimum literacy.

Another consideration is addressing the issue of obsolescence. The very nature of the rapidly changing computer graphics field dictates the necessity for intelligent selection of hardware and software. The importance of effective research and development to keep abreast of these changes allows you to maintain a viable computer graphics program.

As one example of the use of available software as a tool for literacy, the use of the Gibson Labs Lightpen and the Graphics Tablet and associated softwares have proven to be a very acceptable link between traditional drawing media and packaged graphics software and serve as a steadying influence in the transition into "technology".

In the case of using the Gibson Lightpen software in a week-long twenty-hour intensive literacy workshop (Computer Literacy for Designers) we found giving the students actual written explanations, or reference sheets as to what the menu was, how the software interconnected, began student understanding of the structure of the software. With the use of only one screen, it is impossible for the new student to remember the menu choices. When two monitors are available, it may be important to rewrite the software so the menu appears on one monitor and the graphic image is previewed on the other. Supplying each student with "PIXEL" paper helps the student relate to the new medium in terms of traditional methodologies. We are not suggesting that ultimately a student should "hang on" to old ways, only that for comprehensive literacy, it supplies a connection that reduces fear and instills some level of confidence.

Problems with Computer Graphics Education
As with most institutions, obtaining money and space to purchase equipment for labs and classrooms is difficult. There are great difficulties in obtaining hardware donations. Many of the high technology companies who are clearly interested in creative usage of their equipment are too small to be able to afford equipment donations to schools. One way around this is to arrange to be a beta site in lieu of donations. In this way, the institution would have the option of testing the equipment and offer suggestions to the manufacturer, in trade for having state of the art equipment on site temporarily. The Federal Government supports donations to science programs but not to art programs and ways have to be found around this bias. We have found that you need to educate high technology companies that are either in your area or

their representatives, of the need for computer equipment in art related programs. It should be stressed upon them that many graduates will be in need of equipment upon graduation. Emphasize the need for more collaborative efforts between engineers and artists to produce better equipment for all users.

The limits of computer systems are a great problem. The larger more powerful equipment is prohibitively expensive and the micro systems are limited in quality. As seen through the courses offered through Mass. Art, we have seen problems with the limitations of single user workstations such as Camex, Scitex and the Artronics. Access time also presents problems, with these types of workstations, and the micros to date, although affordable do not perform the vast functions needed to present a truly comprehensive program.

Any school or institution should begin to establish an identity of what its major goals are and who they wish to accommodate. In the need to provide computer literacy both on a global level and a basic direct level, it is important to remember that at the base is a development of visual literacy and the introduction of the computer into the classroom should assist in that application. Programs should not be developed in a patchwork quilt fashion. The goals of the school should be carefully defined, along with guidelines. Current faculty should be trained and additional faculty selected. At present, many of the current faculty in these institutions are overcome with the same problems as are the students.

Computer graphics does not necessarily fit into any of the programs already in existence. It reaches all disciplines, and with clearly defined goals standards should be set for admission requirements and how prerequisite sequences should be handled.

In our experience, rather than placing people directly in a programming class, we have offered a basic foundations course which has consisted of understanding computers, history, and introducing them to various applications of computer equipment. There is encouragement to attend conferences and trade shows. A development of a reading list and a general awareness of changing career options is initiated.

To combat fear and resistance among faculty and students, providing a team teaching approach proves successful. Drawing instruction from several disciplines offers the student or faculty a way of seeing all aspects of computer understanding. A program coordinator who oversees all the classes and encourages new approaches to creativity with the new technologies is essential to maintain a coherent program.

Conclusion

It is true that computer literacy, training and education are open ended questions for any educator. No one has yet explored the potential of the equipment, its systems and output possibilities. What this means for artists and designers and the places and people training them will be seen within the next few years. It is important to understand the goals of your institution, to rely on the expertise of not only technicians and computer scientists, but creatives who have worked in computers in the arts. As instructors at Massachusetts College of Art it has been important to develop a curriculum that reaches all of the disciplines, and to develop a common philosophy, for it is no longer valid to teach design or art without the enhancement of a tool such as an interactive graphics system.

About the Authors

Joan Shafran
Poet/Computer Artist/Educator
Masters of Science in Visual Studies
Massachusetts Institute of Technology
Works in visual poetry using computers, electrographics and airbrush. Working towards educating designers and artists in the use of computers and teaching visual awareness to engineers. Currently instructing at Massachusetts College of Art, and consults to the computer industry in the areas of computer graphic applications and training and education.

Brian Wiffin
Artist/Graphic Designer/Computer Artist/Educator
Bachelors of Fine Arts
Massachusetts College of Art
Works in printmaking, illustration, electrographics and computers. Currently involved in curriculum development and computer graphics instruction at Massachusetts College of Art. Computer curriculum consultant to Melrose Public Schools, Melrose, Massachusetts. Operates a free-lance graphic design business. Committed to involving artists and designers in the use of computers and writing of creative software.

Renée LeWinter
Designer/Printmaker/Educator
BFA Pratt Institute
MA Goddard College
Lecturer at the Visual Technology Center—Camex Project, and faculty member of the Computer Graphics program, both at Massachusetts College of Art. Uses typographic imagery as symbols and textured marks to create spacial landscapes. Exploring the possibilities of page make-up systems as drawing tools and consults in this area.

Camera In / Camera Out

Thomas Porett

Philadelphia College of Art

Abstract: As microcomputer graphic capabilities become more sophisticated, video digitization is becoming a standard input process. Hardcopy output includes both paper printout and video/photographic recording devices. This paper will outline several distinct approaches that are now available for optical scanning and hardcopy output.

ANALOG/DIGITAL

Video digitization is actually a sophisticated analog/digital conversion process that "looks" at the continuously varying voltage coming from the video camera or recorder, and assigns a numeric quantity to signify that voltage. Typically a system is comprised of a black and white video camera, an interface card that accomplishes the A/D conversion, some software in RAM or ROM that directs the hardware, and finally, one or more image capture devices ranging from a disk drive, and/or printer, or in some cases a film recorder.

In the simplest configuration, data are organized through hardware or software to output directly onto the graphic screen in whatever resolution the particular device is equipped to display. An example of this is the manner in which a still video image is scanned into an Apple II+ computer by the Microworks DG-65 digitizer. With this system, an image is converted into a high resolution (280 by 192) image using one of two conversion routines located in ROM, on the interface card. The resulting image is a rather crude black and white facsimile of the original. Contrast, brightness, and picture width are adjustable through three potentiometers located on the interface.

A more refined image can be obtained through the use of disk based software that offers the user a fair degree of control over conversion parameters such as contrast, brightness, and dithering algorithims. Further controls are available that amount to basic image enhancements that offer substantially greater control of the final image. Typically, these controls allow the user to transpose values, and manipulate false color by assigning a color to a gray scale value.

In spite of these desirable controls, the inherent limitations of the Apple's screen resolution keep the image from attaining a successful illusion of gradation in gray scale. This problem is confronted in some dramatically differing approaches that are discussed below.

PRINTER RESOLUTION

One rather elegant strategy that some manufacturers have settled upon involves bypassing the Apple raster screen display in favor of using the print resolution of a dot matrix printer which is typically three to four times finer in its ultimate resolution. The Photocaster system available from Commsoft Inc., is one prominent example of this approach. This imaging system is more accurately described as an integrated imaging environment. It relies primarily on software oriented data manipulation with extensive image enhancement

capabilities that are designed to output to the dot matrix printer.

The machine display screen is used to roughly indicate what changes are being effected by the system. These manipulations are implemented through an extensive image enhancement section of the software, and an editing feature that permits text to be added to the image. The final image (128 x 128) with 16 level gray scale is realized on the printer. Although this figure seemingly implies limited resolution, the implementation of gray scale yields an image of startling accuracy when compared to conventional dot matrix screen dumps.

TRUE COLOR PRINTOUTS

Another unique feature is the system's potential to create accurate, full color images by inputting red, green and blue filtered images and printing a composite image with a color capable printer. A full color subject is presented the monochrome video camera, first with a red filter in place in front of the lens, and and scanned into memory then saved as a file. The same procedure is followed with the green and blue filters, resulting in the three files saved on a floppy disk. Each file is then printed in the appropriate color using a printer such as the IDS Prism 80 or 132 ribbon printer. The task is simplified substantially by using an intelligent parallel interface such as the PKASO card from Interactive Structures Inc. The colors are printed in three successive passes of the paper through the printer, one for each of the primary colors, leading to a full color image remarkably faithful to the original.

IMAGE TRANSMISSION

An added feature of the Photocaster system is its built in modem that allows the user to convert an image into an audio signal that can be sent over phone lines or transmitted on radio. The data structure of the image is configured to standard slow scan television format, common to image transmission used by amateur radio operators. A futher product enhancement is soon to be offered that will allow a 256 x 256 image resolution.

HIGH RESOLUTION GRAPHICS INTERFACE

A radically different approach to extending digitized image resolution is facilitated through an additional interface that establishes an entirely new video display using the host CPU strictly for processing information. This interface is manufactured by Number Nine, and is available for the Apple or IBM personal computers. The highest resolution available is 1000 x 1000, or less depending upon the number of colors desired. The complexity of the hardware necessitates some rather demanding programming, a task that has been elegantl filled by Visual Data Enterprise, a firm offering software for the Number Nine that allows use of a graphics tablet and video digitazation with an Apple II+ or IIe. This hybrid approach allows the small computer to acquire attributes of a sophisticated imaging workstation, capable of serving the needs of the professional graphic artist.

IMAGE STORAGE DIFFICULTIES

Although the added resolution of such a system is most desirable, a problem emerges regarding final image output and storage. The typical approach used to store an Apple "high resolution" image is the floppy disk. Traditionally up to sixteen images can be stored on

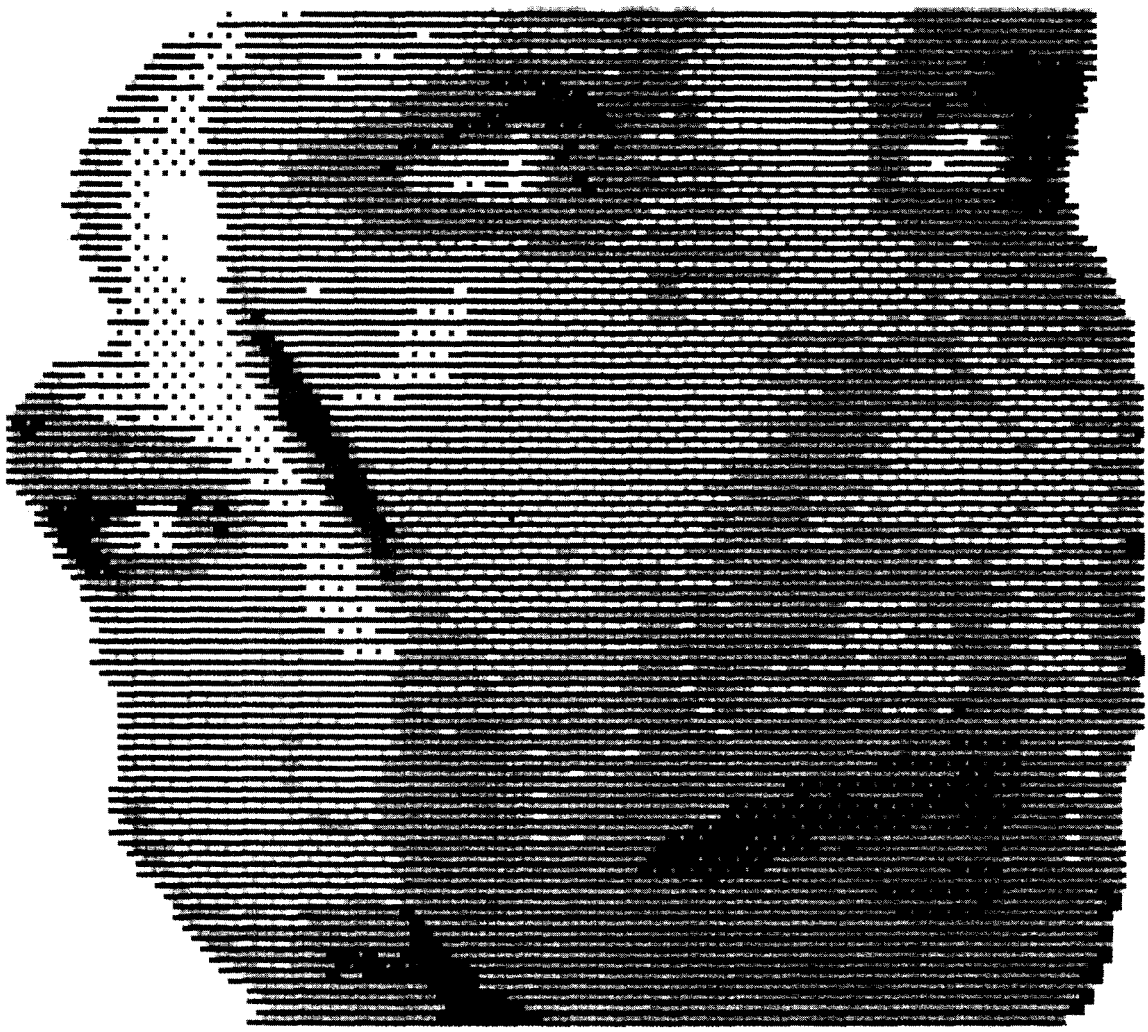
one disk, but this figure is radically reduced if higher resolution images are stored. Assuming a resolution of 500 x 500 with 16 colors, one image would take up to three floppy disks depending upon how the data is organized. Additionally, such an image cannot be printed on commonly available printers, necessitating a turn to some photographic storage process. Although it is possible to photograph a color monitor directly, suffice it to say this approach will not lead to optimum image quality.

FILM RECORDER

In order to gain full benefit of the enhanced imaging system, one must utilize a dedicated film recorder device. Essentially, this device consists of a flat screen black and white monitor, a camera, a filter wheel with red, green, and blue filters, along with associated electronics that utilize an RGB video signal. Although some recent small computers provide RGB output, such is not the case with the Apple II+ or IIe. Instead, an additional interface is required to provide this signal to the monitor. The Amdek DVM 80e, designed for the Apple IIe 80 card auxiliary slot, is one well designed example. The video signal from the card is used to drive an RGB monitor and film recorder. In a typical image recording, the camera is loaded with a color transparency film, and three successive exposures are made as the red, green and blue filters are rotated into place, and the appropriate values of that color are displayed on the flat screen monitor.

The price of film recorders has been in the range of six to ten thousand dollars for a low end unit. This is rapidly changing as, devices such as the Lang Videoslide 35 make a strong showing in a price bracket of less than three thousand, with the performance that equals the more expensive units. Using this approach, the image maker can photographically save a high resolution image as photographic hardcopy on negative or positive, black and white or color films.

The idea of creating an image through camera input, digitally mediating that image through enhancement programs or electronic paint programs, and then finalizing that image in a photographic medium is truly a viable approach made possible by these new graphic tools for the small computer. These tools are admirably suited to the graphic or fine artist as they are reasonably priced when compared to commercially available systems, and offer a great deal of flexibility through the variety of configurations possible.



MOVIEMAKER: A REAL TIME MICROCOMPUTER ANIMATION SYSTEM

Eric S. Podietz

Interactive Picture Systems

ABSTRACT

The average home computer (under $2000 range) provides a balance of features geared toward a high degree of interactivity. An emphasis is made on graphics and sound capability since seeing and hearing are human facilities that transcend computer literacy. The anthropomorphic term 'friendly' is attributed to computer programs that relate to people in a manner that comes naturally to them. MovieMaker is an animation program that takes advantage of these features and is easy to use.

INTRODUCING: MovieMaker

MovieMaker is a microcomputer program that allows one to create two dimensional animation, naturally. The 'natural' component is bred into the system through its means of interacting with the user:

- It is a visual, screen oriented system: "What you see is what you get"
- The keystrokes used to direct the system correspond to English words that relate to animation. (e.g 'A' for Action, 'C' for Color)
- Creation, editing and replay occur immediately, in real time. The result of one's actions is never hidden for any period of time.
- Division between modules is organized in such a way as to avoid a tedious progression through menus.

If one thinks of animation as the motion of shapes over a background in time, three distinct data sets emerge:

- a set of shapes
- a background
- an animation describing the motion of the shapes over the background

In MovieMaker, shapes and backgrounds are developed in the Compose module, and animation is created and edited in the Record module. Shapes, backgrounds and animations are stored in three distinct files: SHP, BKG and ANI files (see figures 1,2,3). The Record module also allows sound effects and music to be added at any point in the animation. The sounds are selected from predefined sets of sound/music effects stored in a fourth file type, a SND (sound) file. These four file types are stored on disk, with names chosen by the user.

COMPOSE: Building Shapes and Backgrounds

In Compose one creates a shape page and background (see figures 4,5). Compose utilizes simple image processing techniques, such as mirroring, color transposition, opaqing, brushing, and window duplication to allow image data to be rapidly built up on the screen. For example, to create a walking sequence, one would paint one position in the walk sequence, copy that position to several other parts of the screen, and then alter each copy to correspond to the remaining positions in the walk sequence. Similarly, a forest background could be built by painting one tree, and then copying it all over the screen.

The shape and background pages in Compose are separate screens which can be flipped between by hitting the Escape key. Image data can be copied from either of these screens to the other using the Window and Duplicate functions. The user frames the area using the Border command, picks it up with the Duplicate command, directs it to the area where it is to be placed using a joystick or graphics tablet, and drops it by hitting the Duplicate key again. This flexibility allows the Background page to be used as a kind of work page or palette where textures can be built up, icons edited, etc.

A shape page is a screen full of shapes; shapes are delineated by the user's placing a window around the image data representing the shape, and pressing the Outline key. When a shape is Outlined, Compose remembers where it is on the screen, thus allowing the user to return the window to that shape, move the shape, and so on. Once several shapes have been Outlined, they may

be put in a sequence. Sequences are created using the Sequence keystroke; the user designates the order of shapes in a sequence by moving the window to the next shape to be in the sequence and pressing the joybutton. The first two shapes of a sequence are shown in figures 5 and 6.

A sequence can consist of up to 16 shapes; there can be 9 sequences. Once a sequence has been defined, it can immediately be tested by pressing the Action key. Action displays the background page and flips through the shapes in the sequence at a rate from one to thirty frames per second, selectable with the Frame rate key. Using the joystick, the user can position the moving sequence at any place on the background page, providing a means for motion testing the sequence.

At any point in Compose the user can save or load the shape or background pages. Thus new shape pages can be built out of data from several previous shape pages. The ability to copy data between shape and background pages would allow the user, for example, to copy a tree from an old background onto the shape page, make it into a shape, enter it into a sequence, and animate it.

RECORD: Creating and Editing Animation

In Record animation is created and edited. Shape, Background, Animation and Sound files are all used in the Record module (see figure 7). Record is designed to emulate a multi-track videotape recorder, with familiar functions such as play, rewind, and freeze frame. There is a 'track' for each of the elements that can be changed during the animation. The length of the animation, types and numbers of tracks varies with the host computer. MovieMaker on the Atari 800 allows 300 frames of the following tracks to be recorded as part of an animation:

- 6 actor tracks
- 4 color tracks
- 4 sound tracks
- 1 zoom/pan track

Recording of all the tracks is done in real time, as if filming real action. All recording is done by designating which track one wishes to record on, selecting a frame in which to begin recording, selecting a frame rate at which to record, and starting the record process. As the recording proceeds, the user makes changes to the track which is being recorded. Recording can be done single frame, or at frame rates ranging from 1 to 30 frames per second. The animation can be replayed at any of these frame rates as well. Figures 8 thru 12 show various frames of an animation being replayed. The frame counter is at the lower left.

To record on actor track 1, A1 (for Actor 1) is pressed, a sequence selected, and the space bar hit to begin recording. The selected sequence then begins execution, and, as in the Action mode of Compose, the sequence can be positioned anywhere on the screen with the joystick. Record remembers the screen location of the actor as each frame is recorded. To aid the user in establishing the interaction between actors, Record plays back whatever was recorded before.

For example, an animation of a man walking his dog would be done as follows: First the sequence of the man walking would be laid down on Actor track 1, by positioning the walking man on the left hand side of the screen and moving him across to the right side of the screen. This animation would then be Rewound (and you would see the man walk backwards). Then the dog walking sequence would be selected, and recorded as Actor 2. As the dog was recorded, the man would be played back, thus allowing the dog to be positioned alongside the man in every frame.

Up to 6 actors can be recorded in this manner. Of course, it is possible to switch sequences at anytime, making it easy to change from a man walking left to a man running right. Any track can be recorded over, making the editing of screen position or shape information a cinch.

The 4 sound tracks correspond to the 4 voices available on the Atari 800. For each sound track, 8 predefined sounds are available, for a total of 32 possible sounds. These sounds can be continuous sounds that proceed until they are shut off, or sounds that shut off of their own accord, such as a musical note or explosion. Sounds are recorded by selecting which of the four sound tracks to record on (e.g Noise 1 for sound track 1), selecting a frame rate and hitting the space bar. As the existing frames are played back, the user hits the key corresponding to the sound which is to be initiated in the frame being displayed. The sound is heard immediately. Since Record is playing back everything that had been recorded besides sound track 1, synchronizing the sound to the animation is easily achieved.

The 4 color tracks allow color animation to be recorded. The 4 color tracks correspond to the 4 color registers of the Atari 800. Each of the color registers can be one of 128 colors, which can be broken down into 16 hues each with 8 luminances. Recording color changes allows the hue and luminance of each color register to be adjusted during the animation. Aside from enhancing effects such as sunrises, lightning, and explosions, traditional color map animation can also be done.

Finally, the zoom/pan track allows the 'camera' filming the animation to be zoomed to 1x, 4x or 16x magnification. In the 4x and 16x levels, the camera can be 'panned' to show the action proceeding on any portion of the background. Zoom/pan recording is invoked with the Zoom key, and pan tracking is achieved with the joystick.

POST PRODUCTION: Smooth and Play Modules

The Smooth and Play modules of MovieMaker are invoked once the user has completed an animation. The Smooth module combines the shape, background, animation and sound files into one file, for playback purposes only. The function of the Play module is to play these files. Since the data in the files is compressed, the playback files take up considerably less disk space than the files from which they were derived.

Since the Play module offers flicker free playback, it is possible to videotape the animations as they are replayed. Animations that were split into several parts due to space limitations can be linked together by pausing the videotape recorder between the replay of each animation.

SECTION 5: Applications of Moviemaker

MovieMaker's ease of use and immediacy make it a valuable tool for previewing animation. This has application in the animation, broadcast, advertising and microcomputer software industries. The issue of prime importance is that live action can be created and reviewed rapidly. Traditional animators using MovieMaker have reported that overall efficiency is improved 2 to 4 times over traditional cel animation techniques. This does not include improvements in the overall quality of the animation gained by being able to repeatedly review and edit.

Since the entire animation is in a data as opposed to program format, the animation is device independent. This means that the animation can be moved from one computer to another. Differences in graphics hardware, such as color and screen resolution, would be accomadated by translation programs employing color and pixel reduction/expansion algorithms. Sound effects might have to be omitted on some systems.

Perhaps the most important current application of MovieMaker is in the design of software for low cost microcomputers. By utilizing the MovieMaker graphic primitives, collectively known as the MovieMaker Development System, animation produced using MovieMaker can be included inside of a user's program. This would allow, for instance, certain animations to be called from disk given certain program conditions. It is also possible using the runtime routines to produce animation on the fly, obviating the need for continuous disk access.

CONCLUSION

While microcomputers have been viewed as 'playtoys', they provide an environment in which highly interactive graphics programs can run. MovieMaker is one such program, which allows animation to be rapidly developed, edited and reviewed. It can be expected that as the graphics resolution and computing power of low cost computers increases that systems of this type will become standard items in the toolkits of those wishing to use computers to produce video and film.

ACKNOWLEDGEMENTS

I would like to thank Guy Nouri, Mark Scott, Jimmy Snyder, and Bob Svihovec, all of Interactive Picture Systems for their help on MovieMaker. Guy contributed immensely in ensuring that MovieMaker was easy to use and effective as an animation tool. Mark, Jimmy, and I slaved for 18 months programming and reprogramming MM in 6502 assembly language. Bob Svihovec, our animator, has created innumerable animations (including the one in the illustrations) and has shown incredible patience with the numerous changes and bugs that MM has undergone. Finally I would like to thank the Reston Computer Group who made the financial commitment that made MovieMaker a reality.

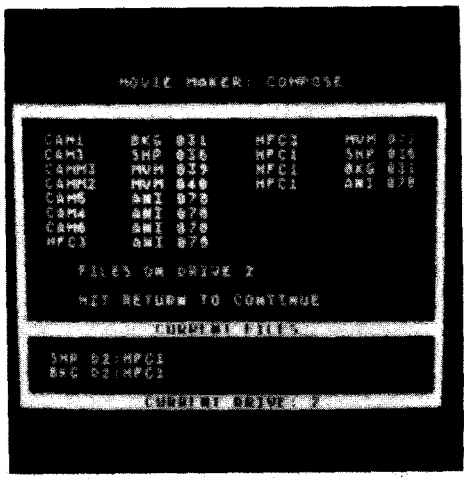


Figure 1

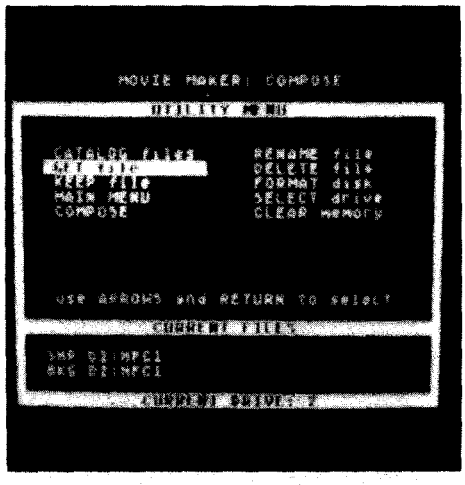


Figure 2

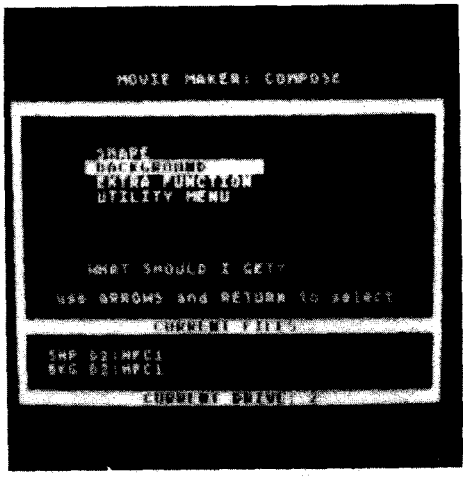


Figure 3

Figure 4

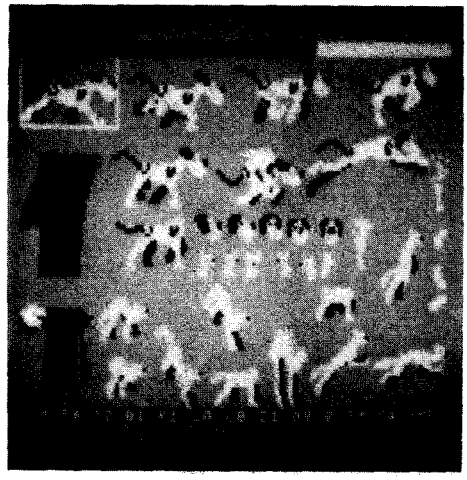


Figure 5

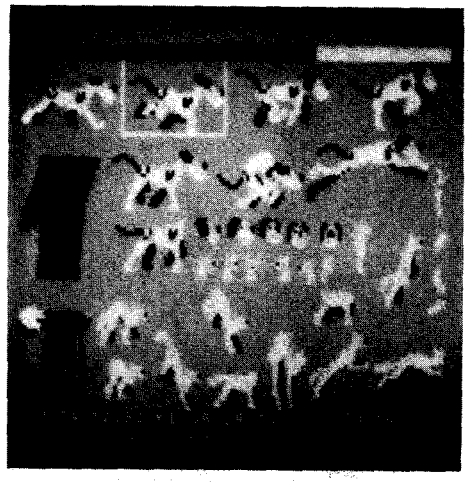


Figure 6

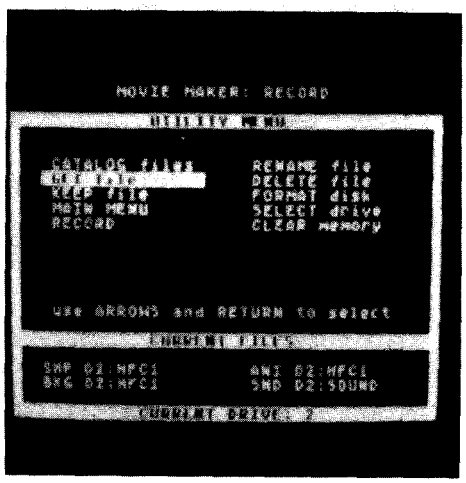


Figure 7

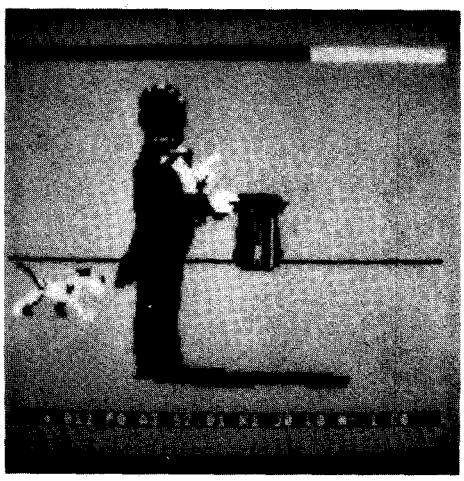


Figure 8

Figure 9

Figure 10

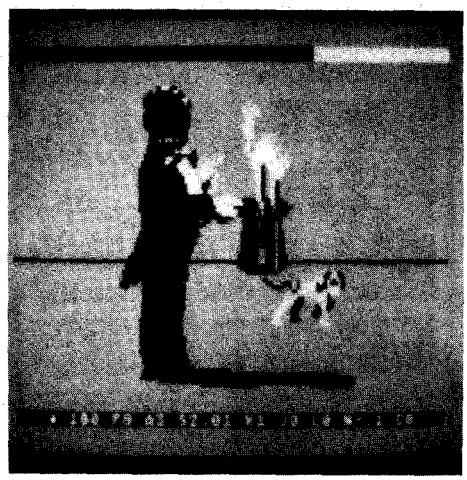


Figure 11

Figure 12

COMPUTER ANIMATION AS AN ART FORM

Donna Mansfield

Drexel University

Introduction

Computer animation has developed from a visual method of representing equations for scientists into an exciting new art form for artists who use the computer to produce art. However, there was a crucial stage before computer animation moved out of the laboratory and into the artist's studio. The computer was used to assist animators in producing conventional animation. Only after the computer was successfully integrated into conventional animation did computer animation emerge as a new art form. Art is a difficult word to define, but for the purposes of this paper we will define art as the act of producing form for its own sake. This definition will help us differentiate between computer graphics used by artists and computer graphics used by scientists in the laboratory.

Photography and film both imitated accepted forms of art before they emerged as artistic disciplines of their own. Computer animation has followed the same path, now emerging as its own art form only after having been successfully integrated into an established art form.

In this paper, I will describe the evolution of the computer in animation from a tool that assisted artists in producing conventional animation to a tool used by artists to create a new art form.

History of Animation

Animation is the name given to that branch of the cinema in which films are created frame by frame and have some appearance of movement. We have defined animation as a branch of the cinema, but it was not always that way.

The earliest animations were simple: brightly colored drawings of dogs running or clowns juggling. Each animation was about 2 seconds long and composed of about 12 different pictures. These animations were shown on mechanical devices like the zoetrope or the phenakistoscope. The drawings were mounted on a drum or disc that revolved. Spectators viewed the drawings through slits.

Animation was further developed by Emile Reynaud who invented the praxinoscope to show his drawings. This machine was similar to the zoetrope but replaced the slits of the zoetrope with mirrors set at an angle.

In 1888, Reynaund opened the Theatre Optique in which he showed colored animations ten to fifteen minutes long. Initially, Reynaud's animation were very popular, but 1888 was also the year of the invention of the first movie camera and the motion picture cinema quickly drew Reynaud's audience away from his animations. No artist of quality would draw pictures for a machine and Reynaud's device remained only a novelty.

It is interesting to speculate what might have happened had Reynaud's device attracted an artist who drew beautiful pictures for it. The history of animation might have developed differently. But during Reynaud's time, great artists did not draw for machines and so mechanical animation died with Reynaud (1).

Subsequent animation did not build on anything Reynaud had begun. Essentially,animation was reinvented in the early 1900's when it reemerged as part of the cinema. Artists produced nothing of the scope of Reynaud's work until the 1930's, and the animations of Walt Disney, perhaps the best known creator of animation. Many of the techniques that Disney used are still being used today.

Techniques of Conventional Animation

Conventional animation is a tedious

CH1930-7/83/0000/0013$01.00 © 1983 IEEE

process; While there is great creativity involved in the making of animated films, much of the work is repetitious and boring. These are some of the steps involved in making an animated film:

1. The characters are drawn in pencil on different sheets of white paper.

2. Backgrounds for the individual frames are painted on cardboard sheets.

3. Director decides how the camera will shoot each frame.

4. Cameraman shoots a pencil test to get a preliminary idea of the final film.

5. The artist traces the outline of the picture using thick black lines.

6. The colors of the characters are filled in.

7. The backgrounds and characters are combined on the camera stand to create the correct shot.

8. The frames are filmed (2).

All of this takes much time and money and many people. Each frame of the film must be created separately. Studios usually employ artists to create the "key" frames, the first and last frame of a particular movement, and use assistant artists to draw the frames in between the key frames. These intermediate frames are called "in-betweens" and the assistants who draw them are called "in-betweeners." The transition of movement must be smooth from one frame to another and each frame in the same sequence must have the same background. A filmed pencil test gives the artist a primitive idea of the final project. But the pencil test uses only black and white outline drawings. Conventional animation techniques offer no really effective way of previewing the action in color before it is finally filmed by the camera. Errors that appear here cannot be easily corrected: drawings must be redone, recolored, etc. It is easy to see why animation is such a difficult and expensive process.

Critics of animation often say that Walt Disney could not afford to produce a "Fantasia" or a "Snow White" today. The salaries required to produce an animated film of this quality would be prohibitive. The popularity of television kept audiences home and motion picture studios were no longer able to afford the high cost of animation. Animation reemerged as a staple of Saturday morning TV.

The quality of animation greatly deteriorated after its move to TV in the late 1950's (3). Disney animations were famous for their expressive movement and meticulous attention to background detail, but now artists were forced to use the techniques of "limited animation" for TV. Limited animation meant that the artists restricted their representations of movement and simplified the background detail as much as possible. This was a result of the smaller budgets and shorter deadlines of TV. Donald Heraldson says "the animation industry today (1975) is in financial trouble and often claims are made that it is dying. Obviously, for a fast dollar, it is being smothered from the inside out" (Heraldson, p.82).

The technology of motion pictures has improved dramatically since 1888. Filmakers now reguarly use technical effects that the early motion picture pioneers would not have dreamed possible. George Lucas recently said that the technical effects used in "The Return of the Jedi" simply were not possible when he made "Star Wars" nine years ago (4). Animation presents a paradoxical example, in that it is unlikely, given the techniques of conventional animation and the present financial realities that we can produce animations of the same quality that motion pictures studios produced fifty years ago.

Conventional animation techniques have changed very little in the last hundred years. There was almost no technology involved in conventional animation. Conventional animation was a labor intensive, machine poor art form trying to survive in an industry dominated by technologically superb motion pictures. For animation to survive as an art form, animators like John Halas agreed that "mechanization was essential to advance the whole industry" (5). But what technology could be used to automate conventional animation without changing its unique style?

History of Computer Graphics

The use of the computer to generate movement evolved accidentally, the result of early experiments by scientists and engineers who used prototypes of today's sophisticated systems to analyze objects in motion and to create or simulate conditions with graphics that could be displayed on screens.

Early computer graphics work exploited the amazing "number crunching" capabilities of computers. Scientific

events governed by complex equations could be visually displayed on a computer screen. Objects could be moved according to the laws of physics. Computer graphics was also used extensively to simulate activities like movements in space that man could not see.

Computer graphics offered a technique for moving objects around. First, objects were defined. The programmer entered the coordinates for the first and last frame of movement into the computer and then, the computer automatically interpolated the rest. Each frame of a sequence did not need to be computed separately. Objects could also be colored easily. The programmer specified an area, selected the color he wanted, and that object appeared, so colored on the screen. The animation of objects using the computer was certainly much easier than animations done with the techniques of conventional animation.

These animations using the computer were done only in laboratories and in other scientific settings. No one attempted to move computer graphics out of the laboratory. However,in the 1970's people began to see a natural alliance between the computer and the beleagured animation industry. The computer had been successfully used in animating images in the laboratory. There was certainly no reason why that "talent" of the computer could not be applied to an art form which tediously executed the same steps by hand. Conventional animators were still using the same techniques that had been developed nearly eighty years ago.

Animators hoped the computer would liberate them from some of their more tedious tasks. If the computer could be successfully integrated into the animation process, animated films of quality might produced again. And for computer scientists, animation represented a good research problem. The computer scientists were not interested in changing the art of animation but only in transferring some of the techniques of computer graphics to conventional animation.

Computer Assisted Animation

Much of the work of conventional animation is repetitious, time consuming and tedious. This is the sort of work the computer does well. In the seventies, some computer graphics researchers began to actively adapt computer technology to the techniques of conventional animation. They reasoned that the computer was already being used to interpolate between frames in scientific work and they saw no reason why this technology could not be adapted to speed up the process of conventional animation. Specifically, they hoped that the computer could be used to assist in the following phases of animation:

1. tracing the outlines of figures;
2. drawing the in-between frames;
3. filling in the colors for the figures;
4. visualizing the final project;
5. filming the final project (Wallace,p.254).

Animators wanted the computer to help them, but they wanted the computer to be unobtrusive. They wanted animation to look the same as it always had. They did not want a new "look"; they wanted a tool to help them produce quality animation like Disney used to do. They wanted to be freed from some of the more time-consuming chores of animation. A tool like the computer could perform some of these steps. Animators could spend more time developing original images and would not have to revert to "limited animation" as had happened in the 1960's.

Perhaps, the computer could free animators to be original again. Paradoxically enough, this new tool might help them produce animations as original and innovative as Disney's fifty years ago.

Computer scientists were willing to cooperate to help the artists. These researchers were committed to using the computer as a tool to help animators. They were not interested in changing the "face" of animation. They began to develop software programs to automate some of the techniques of conventional animation. Their committment to automating established techniques of conventional animation can be seen in their naming of the new software programs.

The names of their programs were drawn from the vocabulary of conventional animation. An animator who had worked with Disney would have no trouble understanding what each of the new programs did. The programs that were used to fill in the character outlines with color were called "paint" programs and the programs used to create the frames between the main action frames were called "in-betweening" programs. Both the basic

steps of animation and the language of animation remained the same, whether that animation was completely hand drawn or computer assisted.

This is important. It is often said that artists are uncomfortable with machines. Certainly, there was a time when no artist of quality would work with a machine. The early history of animation is an example of this. Part of this reaction to the machine must come from a concern about the new structure that the machine imposes on their work. The computer was introduced into animation in a way that would still those fears. Roy Madsen says about computer-assisted animation that the the researchers considered the artists' needs and the pleasing result was that "working as a unit (computer and artist), the computer not only challenges the animator's most imaginative efforts but augments them and responds to the slightest whim in less time than it takes to sharpen his pencil. Using familiar tools, as well as equally familiar industry terms the animator can generate and control movement, shape and color simultaneously and instantaneously" (6).

People fear that the machine will replace them. The computer was introduced into animation to help the artist, not to replace him. Marc Levoy, a computer scientist who has been developing computer-assisted animation software for Hanna-Barbera says, "We're not out to replace the cartoon animator. Our goal is just to relieve him of some of the horribly tedious, boring and repetitive work that normally goes into hand-coloring each of the cels in an animation sequence"(7)

Computer-assisted animation has been accepted by the animation industry, and for good reason. One computer-assisted animation system now on the market can do the following: "make in-between drawings from the key frame drawings what the animator enters into the system, and will play back a full pencil test, take conventionally prepared cel-animated figures and scan/digitize them; paint the cels; paint the background art; and record and composite the images on videotape or 35mm film, including angles such as zooms and pans"(8). But this system does not change the way animation looks, "it preserves the integrity and clarity of the animator's art"(Ferderber, p.59). The Hanna-Barbera studio uses a computer to help them make Fred Flinstone cartoons now, and these animations look the same as they did before the computer became a part of the animation process. The technique of making Fred Flinstone might be "high-tech", but Fred still looks quite Stone Age.

The computer has not altered the "look" of conventional animation, only automated some of the processes involved in making this special art. Neal Weinstock says in his article, "New Technologies for the Realization of Ideas" that with "a computer to remember the details and a graphics terminal to make the changes, most of the drudgery can be taken out of the animation process. The animator who doesn't want to work with a computer should be about as rare as the housekeeper who doesn't want to use labor-saving devices" (9).

The computer is a tool, and it has been successfully introducted into animation without changing the vocabulary or the style of conventional animation. Ed Catmull, a leading pioneer in the field of computer-assisted animation says that "computer-assisted animation far from shrinking the role of artist as has sometimes been thought, greatly enhances their productivity and so improves the economics of animation production. As a tool in the hands of animation artists, a dedicated computer system automatically performs many of the non-creative tasks which have had much to do with the high cost and extended production schedules of animated films"(10).

The computer has been successfully introduced into animation. Artists worried that the computer would dehumanize art, but computer-assisted animation looked the same as conventional animation. Artists were not replaced by the computer because the computer was after all, only a tool. The quality of the animation still depended on the imagination and talent of the artist. The computer simply helped the artists speed up the production process. The success of computer-assisted animation proved that the computer could be used in art without replacing the artist or depersonalizing his art. Now that the computer had been successfully integrated into this very old established art form and was being used enthusiastically by artists it was time to see if the computer could be used to create art that was new and distinctive.

The Computer in Art

I do not believe that computer would have used by artists to create a new art form if it had not been successfully integrated into another art form first. The idea of using the computer in art has been frightening to some artists. Perhaps they are afraid the computer will replace them. David Em, a computer artist believes that "the art world has put up a

lot of resistance to to it. But the biggest problem is a pyschological one on the part of artists...There is a tremendous fear of the computer. People think the computer is going to do it all. You are supposed to turn it on and it produces a picture" (11).

This is, of course not true, but it is not an uncommon fear and one that has been true when other technologies were introduced into art.

When still cameras were first developed, artists were aghast. The camera produced pictures of extraordinary clarity: people and landscapes looked "real". Artists must have been a afraid that the photograph would make their representations of people and landscapes superfluous. This did not happen because the photograph and the painting fulfilled different needs. The photograph slavishly imitated reality (it could do nothing else), but paintings created moods and gave a different sort of pleasure. Artists realized that photographs could coexist with paintings and not replace them. Therefore, artists began to feel comfortable enough with photographs to use them as models. If they could not travel to Greece, they could still paint the Parthenon from a photograph. The photograph came to be accepted by the artist as a tool for his art. It was only after the artist accepted the photograph as an aid and not a replacement that artists began to use the camera as a tool to create new art (12). Photography became an art form of its own.

I believe the computer followed this same path in animation. It was first used as a tool to help the artist create animation that looked exactly like conventionally drawn animation. Having been assured that the computer would neither replace them, nor dehumanize their art, artists were able to imagine a new art form using the computer. Kadinsky had envisoned an art form which would radiate directly from the artist's mind without the aid of pigment or brushes. Perhaps, computer animation was that art form.

Computer Animation

The computer had been used successfully to automate the work of conventional animation. But this animation was indistinguishable in style from animation done totally by hand. Artists seeing this marriage of machine to art began to wonder what animation totally generated from the computer would look like. These artists wanted to experiment and isolate what the computer could do that man could not.

Roy Madsen says, "The computer offers promise in areas of animation requiring the maximum of mathematical precision , such as plotting trajectories and depicting complex physical relationships. Pictures may be programmed to move in accordance with the laws of perspective, at any desired speed and following any set of rules or formulae" (p. 190). This is what computer graphics researchers had already discovered, but artists now wanted to use this special "skill" of the computer, not for science but for art. Artists were not interested in visualizing the laws of physics for research, but were interested in using these visualizations as a part of something else, like creating a mood or telling a story. They were not interested in studying movement itself, but in using movement in their art. Using a computer does not bind the artist to scientific laws. Artists can parody the precision of the computer by creating improbable figures which move according to some laws of nature that exist only in the artist's head. The computer is not bound to "the" reality, but is subject to whatever reality the artist chooses for it. The computer, then becomes an extension of the artist's imagination.

Some artists have complained that it is impossible to certain things with the computer. But this limitation does not prevent computer animation from being a unique art form. David Em says that "People say computers don't do this or that. But all art forms are like that...Yet it does things no other medium can do" (Levell, p.39). Artists simply find ways around the computer's limitations, as they do with any other medium's limitations.

For example, the kinds of figures that the computer can generate are fairly limited. So far, no one has animated a completely computer-generated human figure. The figures that are easily generated by a computer are different from anything seen in conventional animation. For years, the computer had been used to assist designers to build better cars and airplanes and most people have seen some examples of the wire frame figures used in this work. These wire frame figures looked like lined grids twisted into whatever shape they were supposed to represent. In the early years of computer graphics, wire frame figures were the easiest to generate. They were used to model automobile and airplane strucures. It was easy for a programmer to rotate, move and change the scale of the figure. Recently, I saw an animated film which used totally wire frame figures, but the film had

nothing to do with designing cars or studying movement (13). Rather the film told a story and created a mood. This is the province of art, not of research. The film featured a wire frame child who flew from his wire frame planet through a wire frame depiction of New York City. The boy flew by the Statue of Liberty which was rotated and shown from several different sides. But this film was not made to demonstrate movement or to model buildings. This was a film that was chsrming, affecting and humorous, a wire frame "Little Prince." It was without a doubt, art, and it had no purpose other than to tell its story, create a mood and give pleasure to its viewers. This film used a figure traditionally used for research to create a special kind of art. The mood was not unique, but the look was. This was not the sort of figure Disney ever drew, but it is the the kind of story he would have told. This film was an artistic application of a design that is unique to the computer.

Computer artists do not just have to contend with the limitations of their new tool. The artist who uses computers can take advantage of some unique properties. One of the most extraordinary is the interactive capability of the computer. Flight simulators have used interactive computer animations for many years. A pilot who chose one path would see an entirely "world" different from a pilot who chose another path. Computer animation can also be interactive. Conventional animators draw the pictures and choose the camera positions which will be used for the final film. The drawings are then filmed and the animation sequence will stay that way forever unless the drawings are refilmed. Computer animation does not always have to be the same. The animations can be different each time. Interactive computer animation is what is used to make video games. While this is not "high art", it is designed to challenge and give pleasure to its players. This was not possible with conventional animation. Interactive animation is unique to the computer and is now being used for something other than research: it is being used to make art.

Computer animation is a distinctive art form and a new one. David Em has said that 1981 was the year computer animation became an art form. It was the year "artists started to emerge and started to do things with it (computer)" (Levell, p.40). It is no longer just a research tool, an aid to another art form or a novelty. Computer animation is beginning to have a personality and distinctive look of its own. Already critics see a change in the look of computer animation, "from grids and wire frame looks which tend to produce a technical clean look and toward highly stylized 3-D work, toward image-oriented material in which objects appear to be spatial but do not necessarily behave acccording to the laws of physics. They are structures that come apart or metamorphose or unfold" (14).

There is already evolution in the look of computer animation and we notice that is away from grids and wire frames which are the very structures computer scientists manipulated in research to new figures which have resulted from artists stretching and expanding the forms of the computer. Saul Bernstein, a prominent computer artist says, "Sometimes I'm glad I don't know their (computer's) limitations. I keep trying to push their machines; sometimes I get results that surprise even their creators" (15).

Our first fear about the computer was that it would constrain the imagination and limit the personal expression of the artist. This does not seem to be true. In fact, the computer responds immediately to the artist. As Kadinsky envisioned, the computer offers a direct path from the artist's imagination to the final piece of art. To change a finished frame in conventional animation, the artist must redo that frame, check the continuity and then refilm the entire sequence. The computer animator can select a frame, change it at his terminal and replay the entire sequence immediately. His animations need only exist in two places: his mind and in the computer, so he does not need to make changes in a number of different media (paper and film) before the project is completed. The computer allows the artist immediate feedback. Conventional animators must wait before the entire sequence is refilmed before they can see how the correction looks.

Far from constraining the artist's imagination, the computer responds directly to it. So powerful is this response that some critics believe that new defintions of art must include the computer as well as the artist (Madsen, p.80).

There is additional evidence for the emergence of computer animation as its own art form. Carl Machover estimates that only 70% of "computer-look" art is done by a computer (16). The other 30% is drawn by hand. Computer animation is now so popular and so distinctive that people who can not afford to use a computer to produce their animations hire an artist to imitate that look.

We seem to have come full circle. Originally, artists let the computer into animation to try to save it, to allow artists to produce quality animation again. The computer was used only to relieve some of the tedium of the animation process. However, after artists saw that the computer could be useful in art, that it would help them and not replace them, they were free to experiment with the computer to see if they could create something unique. And they have. David Em says, "Whenever a new medium comes along people are going to repeat what has already been done. Just as when film came along people tried to do theatre with it. It took a while for the true artist of film to come out and start editing. The same is true of computer art. It's going to take a while to establish itself as a truly unique form. But there's no question that it is" (Levell, p.39).

Conclusion

Computer animation has followed the same path as photography and film in its development as a new art form. Both photography and film first imitated already established art forms before they emerged as artistic displines of their own. Artists used photographs as models to paint from, filmakers first shot movies like theatre with no camera changes at all, and the computer was used to assist animators in producing conventional animation.

New technology brings artists new fears. They worry that the new technology will replace them or dehumanize their art. But, as the new technology imitates older art forms, those fears are quelled. Curiosity displaces the fear and some artists are driven to experiment with the new technology to create art that is unique and different. Computer animation has emerged as an art form recently, but already artists have created designs that are unlike anything seen in the laboratory or in conventional animation.

Artists using the computer have realized what other artists who have experimented with other new technologies eventually realized, that it is the imagination and the talent of the artist that creates great art. So the technology loses its mystique and becomes what it is: a tool to be used, stretched and expanded by artists. The computer has now been accepted as a tool and we can look forward to the further development of this new art form.

References

1. Ralph Stevenson, The Animated Film (London,Tantivy Press,1981), p.8.

2. Bruce Wallace, "Merging and Transformation of Raster Images for Cartoon Animations," Computer Graphics, No.3 (1981), p.253

   Subsequent citations from this work appear parenthetically in my text.

3. Donald Heraldson, Creators of Life (New York:Drake Publishers,Inc.,1975), p.82.

   Subsequent citations from this work appear parenthetically in my text.

4. Paul Scanlon, "George Lucas: The Rolling Stone Interview," Rolling Stone, 21 July 1983, p.15.

5. John Halas, Computer Animation (New York: Hastings House, 1974), p. ix.

6. Roy Madsen, Animated Film (New York: Interland Publishing, 1969), p. 190.

   Subsequent citations from this work appear parenthetically in my text.

7. Robert Rivlin, "Computer Graphics," Omni Magazine, 1982, p.33.

8. Skip Ferderber, "The Commercial Production Designer," Millimeter, Feb. 1983, p.59.

   Subsequent citations from this work appear parenthetically in my text.

9. Neal Weinstock, "New Technologies for the Realization of Ideas,"Millimeter, Feb. 1983, p. 71.

10. Edwin Catmull, "New Frontiers in Computer Animation," American Cinematographers, Oct.1979, p.1001.

11. John Levell, "The Computer Paintings of David Em," Business Screen, 23 Oct. 1981, p.39.

    Subsequent citations from this work appear parenthetically in my text.

12. Ruth Leavitt, ed., Artist and Computer (New York: Creative Computing Press, 1979), p.3.

13. Philippe Bergeron, Nadia Magnenat-Thalmann, Daniel Thalmann, Vol de Reve (Dreamflight), a film, 1982.

14. Suzan Prince, "LIVE! Pratt's Computer Graphics Seminar," Computer Pictures, Jan/Feb 1983, p. 56.

15. Howard Shore,"The Fine Art of Computer Graphics," Softalk, March 1983, p. 150.

16. Carl Machover,"Patterns in Computer Graphics," Computer Pictures, May/June 1983, p. 20.

EXPERIMENTAL VISUAL EVALUATION FOR COMPUTER GRAPHICS

Marek Holynski and Elaine Lewis

Boston University

Abstract

This paper presents an approach for evaluating viewer preference for computer graphics. In this initial experiment, 44 subjects rated 21 computer-generated patterns which varied in terms of order and color. Order was defined through three levels of visual structure -- reflection, translation, and random placement -- which are based on image generation techniques. Findings suggest that more ordered patterns are preferred.

Introduction

Standards for the technical aspects of computer generated imagery are becoming well established. Graphics producers acknowledge acceptable levels for many variables like resolution and speed. These recognized standards can relate to hardware design and software organization, yet few address the basic quality of images. In other words, existing technical standards do not provide clues for practical rules that produce pleasing images. In order to develop aesthetic evaluation standards, we must define structural variables for computer-generated images and relate these variables to human responses like viewer preference.

Aesthetic standards which are tested for viewer response can be important tools for graphics producers. The authors propose an experimental approach for discovering some formal criteria. This study considers a dimension of visual structure, order, as it is defined through computer algorithms that produce abstract patterns. Three levels of order are tested for their relevance to viewer preference.

Previous Work

In Arts and Computers (Holynski, 1976), Holynski suggests five categories for classifying images:

1. Images made by random choice alone.
2. Images made randomly but according to an assumed stochastic distribution which imposes a certain probability for the appearance of basic elements in different parts of the picture.
3. Images in which random choice was limited to a given part of the picture while the rest of the image was developed from a systematic combination of the random portion.
4. Images with very limited random choice which use probability to reach a state which fulfills some assumed conditions.
5. Images made according to completely defined structure using non-random algorithms.

These classifications relate to techniques of image production. Within each, pictures may vary their manifest structure. This relates to the way the form of an image appears to a human viewer, which may in turm be based on methods that people use to structure cognitive coding of visual messages.

Visual principles which stem from concepts in graphic design and which are related to cognitive coding for visual media are relevant dimensions for definition. Historically, many similar approaches toward formalizing criteria for aesthetic standards have been tried by philosophers, designers, and art historians. In order to include the more precise possibilities offered by computer graphics, the authors adopt an empirical approach where criteria are systematically tested for viewer response.

In previous work, three visual principle measures, the variables COMPLEXITY (amount of information), ORDER (pattern of information), and REGULARITY (proportional consistency of information), have been tested for their relation to viewer preference (Lewis 1983, 1981). When represented through computer-generated stimuli, these variables have been shown to be strong predictors of viewer preference. Order is especially significant. In some cases, sixty percent of the variance in preference has been predicted by a stimulus' level of order.

In this study, the authors consider one relevant variable, order, as a standard that relates to image generation technique.

CH1930-7/83/0000/0021$01.00 © 1983 IEEE

Research Objectives

This experiment is the beginning of a larger study which aims to establish visual standards for computer-generated imagery. As a starting point, its goals are limited: 1) To discover the interrelation between three levels of order and viewer preference; 2) To explore how the introduction of color affects viewer reaction for this type of stimuli.

Findings from this and subsequent studies will provide systematic feedback about viewers' expectations. When refined, these findings will serve as production rules. In this way we can produce more pleasing computer graphics.

Stimuli Development

First, a basic element was designed to be abstract, because the addition of representational meaning would confuse the formal effects of viewer preference; and simple, yet capable of producing interesting patterns when presented in a display matrix. This free-form shape, illustrated by Figure 1, was defined by 60 points which were entered from a graphics tablet and stored as a data file containing x-y coordinates. A scaled version of this basic element was reproduced, transformed and manipulated by several computer programs to produce the stimuli.

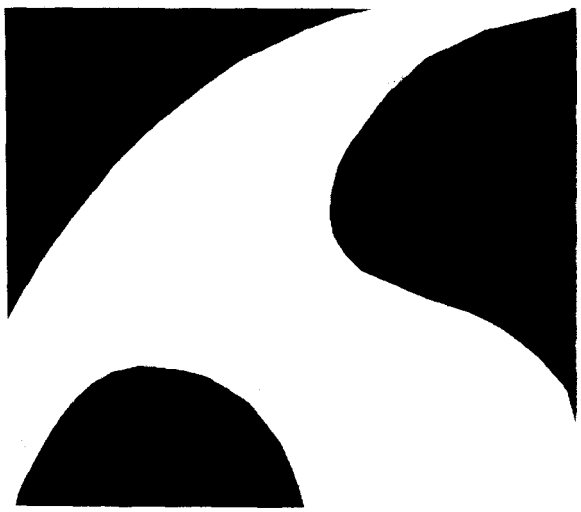


Figure 1: The Basic Element

Each stimulus pattern is a grid of sixteen basic elements. The manner of determining the orientation of these elements defines the order level of each stimulus. Four orientations of the element are possible, as illustrated by Figure 2.

Figure 2: Four Possible Orientations

For the least ordered type of pattern -- RANDOM PLACEMENT -- all orientations are equally probable for every element in the grid (this technique corresponds to group one in Holynski's classification scheme). Orientation is assigned through reference to a random number generator which produces a sequence of 16 random numbers with a range of one to four. An example of this category of stimulus is seen in Figure 3.

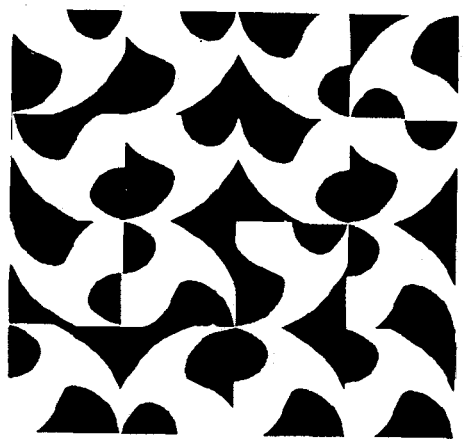


Figure 3: Example of Random Placement

To achieve the two more ordered types of patterns, element orientation was varied in a more systematic manner. Both begin with a four element quadrant where each element was randomly placed. A typical quadrant is seen in Figure 4. (These patterns relate to Holynski's group three.)

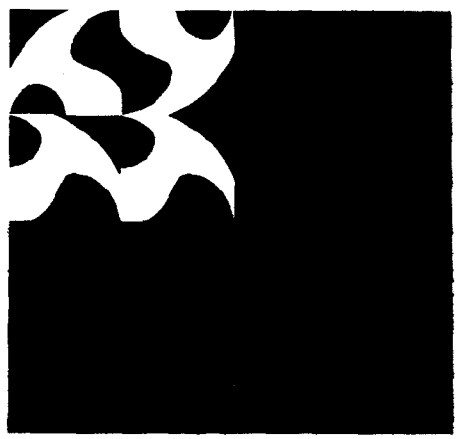


Figure 4: A Typical Quadrant

Patterns representing the middle level for order were produced by duplicating -- TRANSLATION -- the original quadrant to fill the remaining three. Figure 5 shows a typical translation pattern.

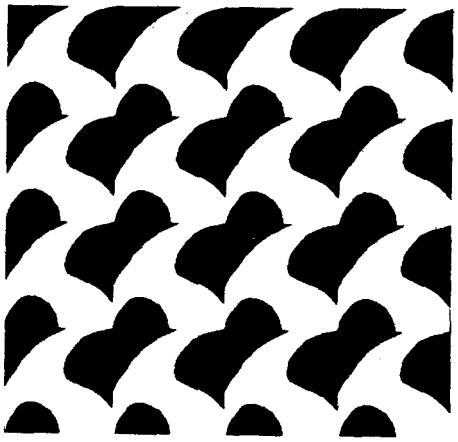


Figure 5: Example of a Translation Pattern

The most ordered patterns were produced by reflecting the original quadrant about both inside edges and the central point. This creates a symmetrical -- REFLECTION -- pattern as shown by Figure 6.

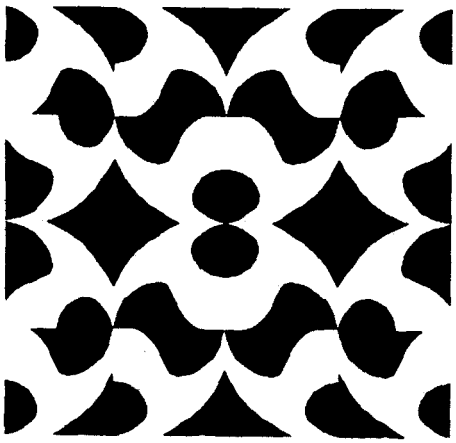


Figure 6: Example of a Reflection Pattern

Stimuli were produced on an AED 512 graphics terminal and recorded with 35mm film by a Matrix camera. Twenty-one slides were chosen to represent three levels of order (seven for reflection, eight for translation, and six for random placement). An additional dimension, color, was explored through three hues (nine white, six red, and six blue).

Testing

Forty-four Boston University students were shown each slide and asked to rate them according to preference using a ten point rating scale. They were told:

> You will be shown twenty-one slides. Each slide is a different abstract pattern. Please rate each slide with regard to preference by assigning a number between 1 and 10 to indicate how much you like a particular pattern. Assume 5 represents an average amount of liking. Assign a large number if you like a particular pattern very much; assign a small number if you don't like a particular pattern.

Stimulus patterns were shown for fifteen seconds each.

Analysis and Results

Mean (average) ratings were calculated for all preference ratings. When considered as a whole, the average preference rating was 4.829. When considered according to a particular slide's order value, some significant differences appear due to a stimulus' order level. A summary table follows.

code	value label	mean
1	REFLECTION	5.579
2	TRANSLATION	4.881
3	RANDOM PLACEMENT	3.886

As this table clearly shows, reflection patterns were most preferred. This suggests that viewers like more ordered patterns better than less ordered or random ones. These results, as expected, are consistent with previous findings (Lewis and Keith, 1983).

Although some differences were observed according to color (blue was most preferred with an average rating of 5.374, white next at 4.7131, and red at 4.459), these are not as significant. Further investigation on the effect of color may be an interesting focus for future study.

References

Bailey, Chris, "Graphic Standards are Emerging Slowly But Surely," Electronic Design, January 20, 1983, pp. 103-110.

Fleming, Jim and William Frezza, "NAPLPS: A New Standard for Text and Graphics, Part 1: Introduction, History and Structure," Byte, February 1983, pp. 203-254.

Holynski, Marek, Art and Computers, Wiedza Powszechna, Warsaw, 1976.

Lewis, Elaine, "An Effectiveness Measure for Visual Communication Media: Toward Definition of Visual Principles," Doctorial Dissertation, Department of Language, literature and Communication, Rensselaer Polytechnic Institute, 1981.

Lewis, Elaine and Brian Keith, "The Addition of Content and Consistency of Preference Ratings for Visual Structures," paper presented at 1983 annual meeting of the International Communication Association.

McClearly, George, "An Effective Graphic Vocabulary," IEEE CG&A, March/April 1983, pp.46-53.

Olenchuk, Bruce, "Graphics Standards," Computer Graphic World, August 1983, pp.56-60.

Schneeberger, Reiner, "Experimental Aesthetics with Computer Graphics: Analyses of Viewers' Impressions of Computer Graphics," Computer Graphics and Art, May 1978, pp. 20-28.

A Pantomation Interface for the Apple II

Tom DeWitt

Image Processing Laboratory
Rensselaer Polytechnic Institute, Troy, N.Y., 12181

INTRODUCTION

Pantomation is a process of tracking objects in the field of view of a television camera. The Apple II computer presents special problems for a video interface of this type, because its video circuitry does not conform to broadcast television NTSC standards. A video interface card has been constructed for the Apple II computer which takes standard television signals as input and makes the derived video information available for processing and display within the Apple's independent time base. In a typical application, the board derives the positional coordinates of a single tracked point each video field (1/60 sec.). The positional information can be used in the same manner as a joystick or other positional controller.

Pantomation has been described in the Proceedings of the Second Symposium on Small Computers in the Arts. The version of Pantomation created for the Apple II similar in design, but improvements have been made in interrupt service handling, key position averaging, and system timing. The current design is compatible with the ubiquitous Apple II, making a tracking keyer available to many more potential users.

THE PANTOMATION CONCEPT

Keeping track of information in a video signal is an awesome task if every pixel is to be examined. Pantomation takes the approach of throwing away most of the picture to ferret out selected tagged areas. This masking or keying operation is performed by external video circuits, typically analog comparators. The resulting signal is a simple binary pulse occupying a small portion of the video frame. The Pantomation interface card takes this simplified video input along with three timing signals: pixel clock, horizontal sync and vertical sync. The circuitry on the board derives the position in the raster at which the masked tag has occurred. This reduces the data from the video rate of 2,000,000 x,y pairs per second at 256 x 256 resolution to 120 bytes (60 x,y pairs) per second. At this rate of data acquisition, even the humble micro computer has plenty of time to process the incoming data.

There are devices called frame grabbers which accept an an entire video image in the time it takes for the incoming video source to write the frame. Pantomation takes another approach to digitizing video information. When a frame grabber stores the rapidly transmitted pixel information, the computer usually idles. After tens of thousands of pixels are stored for evaluation, image processing programs can peruse the pixels to derive positional information. These operations take place out of real time.

In Pantomation real time operation is achieved by a masking operation that takes place at the input. In television parlance, this masking technique is called Keying. Today television keyers have become quite refined. They can detect the presence of a specific color from a very wide palette. Masking based on color is called Chroma Keying. Many television studios include chroma keying as a standard effect, so Pantomation can be immediately implemented in this environment. However, unlike conventional chroma key applications, the Pantograph is not designed to mask out large areas and switch in a second television signal. In a Pantomation application, the chroma keyed area is made as small as practical, because it is resolved ultimately to a single recorded point per frame.

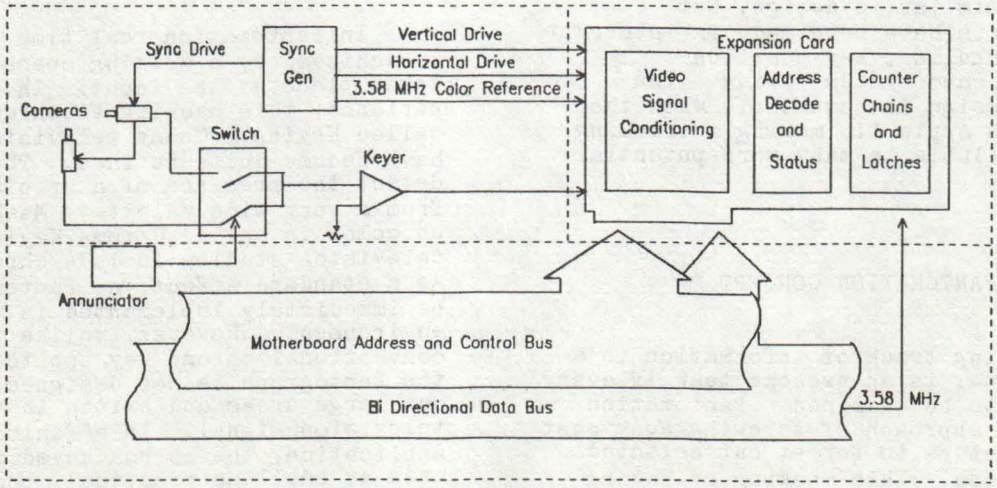
The Pantomation process is very similar in electronic design to the Light Pen, but uses a two dimensional scanner, a television camera, instead of a zero dimensional optical detector, a photo cell. As a consequence, the "pointer" is not restricted in movement to the surface of a CRT display, as are light pens. A tag can be made from a hand held flash light such as a pen light, so the process can feel like a light pen to the user. However, since the optical sensor is not separated from the light source by a CRT glass envelope, Pantomation has superior resolution compared to light pens. Moreover, as we will see, the pen light approach is easily extensible to three dimensional tracking, a feature impossible to realize with a light pen.

Three dimensional input and display are areas of continuing research in computer graphics. To achieve tracking in three dimensions, Pantomation uses multiple cameras. This configuration proves to be a trivial extension of the basic implementation. Stereo pair views can be obtained by using two cameras side by side, or greater accuracy of depth detection can be obtained by using orthogonally placed cameras. The positional coordinates tracked by each camera are obtained in time by switching between the cameras during sequential video fields.

SYSTEM LAYOUT

A block diagram of the Apple II based Pantomation system is broken into three sections: video, Apple motherboard, and Pantomation interface card. The video system must provide standard NTSC synchronization pulses: horizontal sync, vertical sync, color subcarrier (3.58 MHz), and whatever drives are required by the system's cameras. If more than one camera is used, a switching module is required. The Apple can select the current camera by toggling an annunciator on its game paddle interface, so the switching module should be a TTL compatible analog switch such as a CD 4066. In order to detect the tagged area in the video signal, the video system must have a keyer, indicated in the block diagram by an analog comparator. Designs for keyers vary widely and are beyond the scope of this discussion.

The Apple II micro computer is based on the 6502 processor. The motherboard has an architecture which extends the bi-directional 8 bit data bus and four bits of the address bus to each of eight expansion slots. Each slot is selected by a separate enable line decoded on the motherboard. In addition, the IRQ, interrupt request line, and the R/W, read/write strobe line, appear at each slot. Only one expansion slot has the Apple's internal 3.58 MHz color reference signal, but it may be routed by jumpers to any card. This approach was taken for the prototype Pantomation card, so as to leave the special slot available for planned expansion of the Apple's video output capabilities.

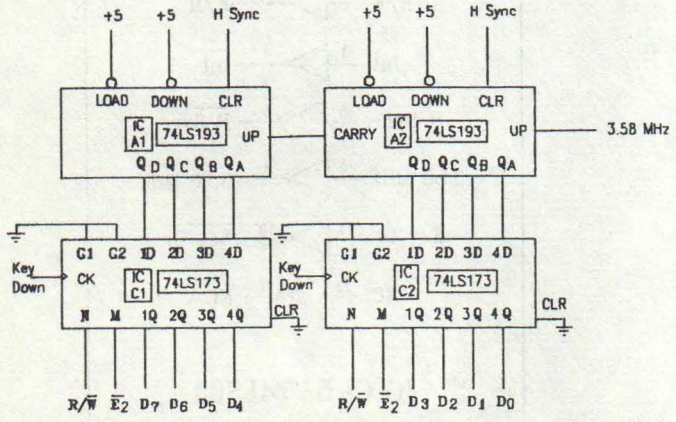


Block Diagram

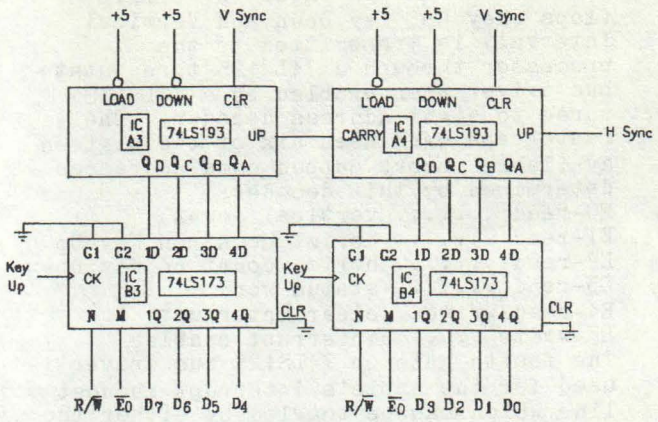
PANTOMATION INTERFACE CARD DESIGN

The Apple II based Pantomation system is constructed on prototype wirewrap card that plugs into the expansion slots on the back of the motherboard. There are four external signals to the board: horizontal sync, vertical sync, 3.58 MHz color subcarrier, and key pulse. These signals are buffered and conditioned to TTL levels by LM 319 dual comparators. A jumper is provided to select the Apple's internal 3.58 MHz color reference signal. This proves useful when the output display is the Apple's bit mapped high resolution screen.

There are two counter chains and three latches on the board. The counters are 74LS193's grouped in pairs for eight bit accuracy. The horizontal counters are clocked up by the 3.58 MHz color reference signal and reset by the conditioned horizontal sync from off the board. Two latches are provided for horizontal position: Key Up and Key Down. The distinction between Up and Down is based on the entry to and exit from a keyed area on a scan line. The Key Up is left hand position of the tag, and the Key Down is the right hand position of the tag. Both positions are latched for transmission to the CPU where a center point will be calculated by software. The latches are 74LS173's which are commoned to the Apple data bus and individually strobed by a 74LS138 address decoder during a read cycle prompted by the R/W line as buffered by a 74LS04 on the Pantograph board. The vertical counter is clocked up by horizontal sync and reset by vertical sync. It is latched by another 74LS173 clocked by the Key Up signal.

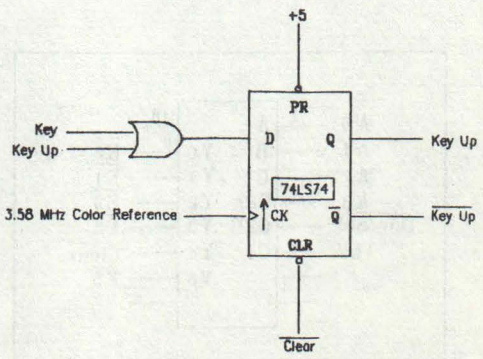


Horizontal Chain (Key Down)



Vertical Counter Chain

To insure that the horizontal count is transferred accurately to the associated latch, a pair of flip flops are used as synchronizing buffers between the incoming key pulse and the latch clock. This circuit locks the Key Up and Key Down transitions to the 3.58 MHz dot clock so the data on the counters is transferred to the latches after the counters have settled. This is an improvement over the first Pantomation system which stopped the horizontal counter at the first key occurrence in a scan line to assure accurate data transfer to the associated latches. The same flip flops store the Key Up and Key Down transitions for testing by the processor. The occurrence of the vertical interval of video is also latched by a flip flop.

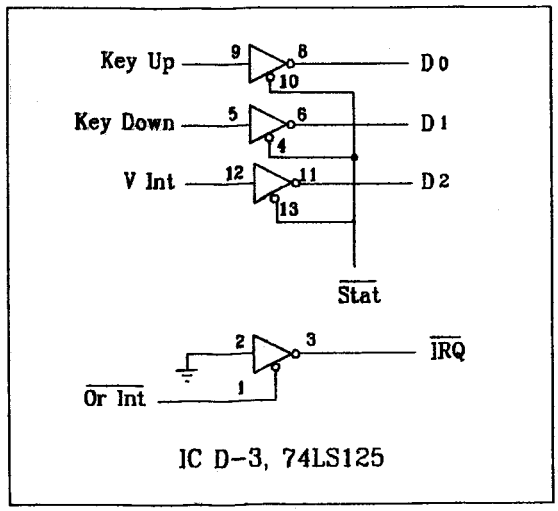


Synchronizing Buffer for External Key Pulse

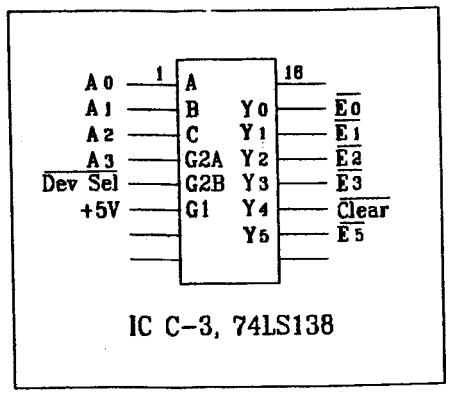
The status of these three flip flops (Key Up, Key Down and Vertical Interval) is transmitted to the processor through a 74LS125 three state bus driver when enabled by a 74LS138 three to eight address decoder. The Pantograph card uses six of the sixteen available memory mapped card addresses determined by this decoder:

E0-read ..... vertical count
E1-read ..... horiz. count on Key Up
E2-read ..... horiz. count on Key Down
E3-read ..... status word
E4-read/write . clear interrupt
E5-write ..... interrupt enable

The fourth gate on 74LS125 bus driver is used for the Apple's interrupt request line which can be toggled by either the Vertical Interval or Key Down. Key Up could be used to initiate an interrupt, but as it is followed immediately (in the micro second range) by a Key Down, this was deemed redundant and the line tied off.

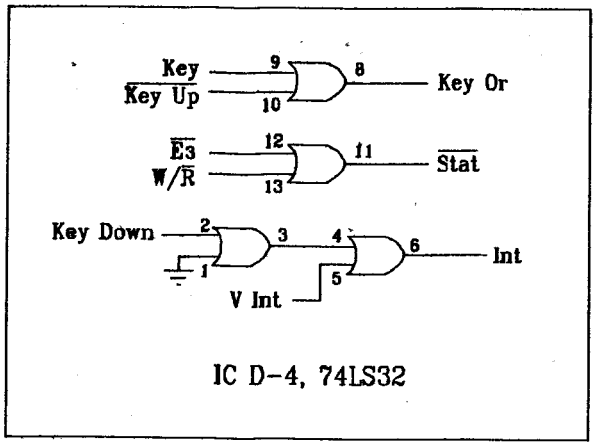


IC D-3, 74LS125

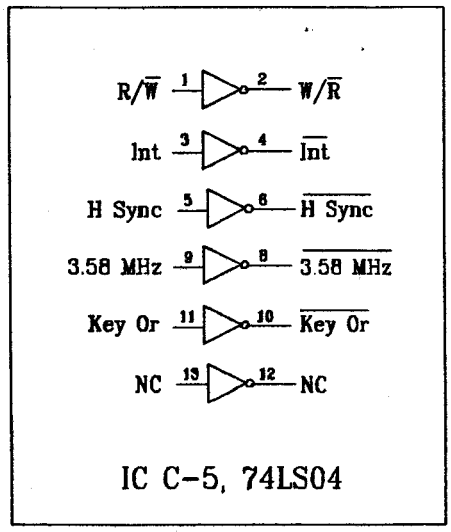


IC C-3, 74LS138

There are 19 integrated circuits on the wirewrap card. To conserve space, two Or gate packages (7432) are used for both And and Or operations, with the And operations being run in negative Boolean logic, converted to positive logic on the spare lines of a 74LS04 hex inverter that was required for other design needs. Component costs were about $100 from a local electronics retailer. All parts are commonly available off the shelf.



IC D-4, 74LS32



IC C-5, 74LS04

SOFTWARE

At the time of this writing, driver software is still in development. The initial set of programs, written in Basic, test the board's functions and registers. A demonstration program written by Eric Podietz was used at the Digicon 83 conference in Vancouver in August 1983. This program simply polled the board for the latest acquired tag values and did not use any of the board's interrupt features. Currently a set of programs are being developed for interrupt operation.

When a hardware interrupt occurs, the 6502 vectors to a service routine. The first operation of the routine is to disable the board's interrupt line. The status register is then tested to determine whether a key or vertical interrupt has occured. If a Key Down but no Key Up is detected, an error is assumed, and the the board is re-enabled and the routine terminated. Otherwise, the program jumps to the appropriate service routine.

The Key service routine picks up the acquired horizontal and vertical position values stored by the board. The x value of the tag is calculated by finding the midpoint between the Key Up and Key Down value. This is called New X, and the latest vertical count is called New Y. The program then looks to see which camera is currently being used and recalls the current recorded x,y position of the tag from that camera. These are called Cur X and Cur Y. The New values are subtracted from the current values to find how close they are to each other. During the course of a video frame, all New values are compared to the position of the current values and the closest New values are stored as Temp X and Temp Y. At the conclusion of the Key interrupt routine, the vertical interrupt flag is tested in case it was set while the key interrupt was in operation. If it was, the program branches to the Vertical interrupt, otherwise it re-enables the board and exits.

The Vertical Service routine takes the Temp X and Temp Y values determined during the video field, and tests them to see if they are within a "Jitter Window". If they are within a short distance from the current x and y values, the old x and y values are kept. This prevents dither from key sample errors that appear as jitter in the final output. Typical jitter window dimensions are 4 pixels on a side. Once the tracked tag moves outside the window, full eight bit resolution is restored to the data acquisition. Additional functions of the vertical interval program are toggling between multiple cameras and calculating the approximate velocity of the tracked tags.

CONCLUSION

Pantomation, a system for position tracking through video camera signals has been adapted to the Apple II computer. The interface is designed to work asynchronously with the Apple's internal video timing chain so that normal NTSC video sources can be used. The data acquisition rate is compatible with the Apple's processor speed so that real time operation and interactive functions can be realized. The hardware investment is low. Driver software is under development with demonstration scheduled for the the Third Symposium on Small Computers in the Arts on October 14, 1983.

ACKNOWLEDGEMENTS

Hardware design and construction for this board was the project of John Lattyak, Chief Operator of radio station WRPI. Software development is being pursued by Russell Brenner, an undergraduate student in engineering at RPI. Eric Podietz of Interactive Picture Systems wrote a demonstration driver program for the board and has generously provided his Painter Power program to help illustrate the board's use. The author is grateful to Ed Falk and Aaron Heller of the RPI Center for Interactive Computer Graphic for helping prepare the layout for this paper, and the author is indebted to his family for their emotional and financial support in bringing this project to fruition.

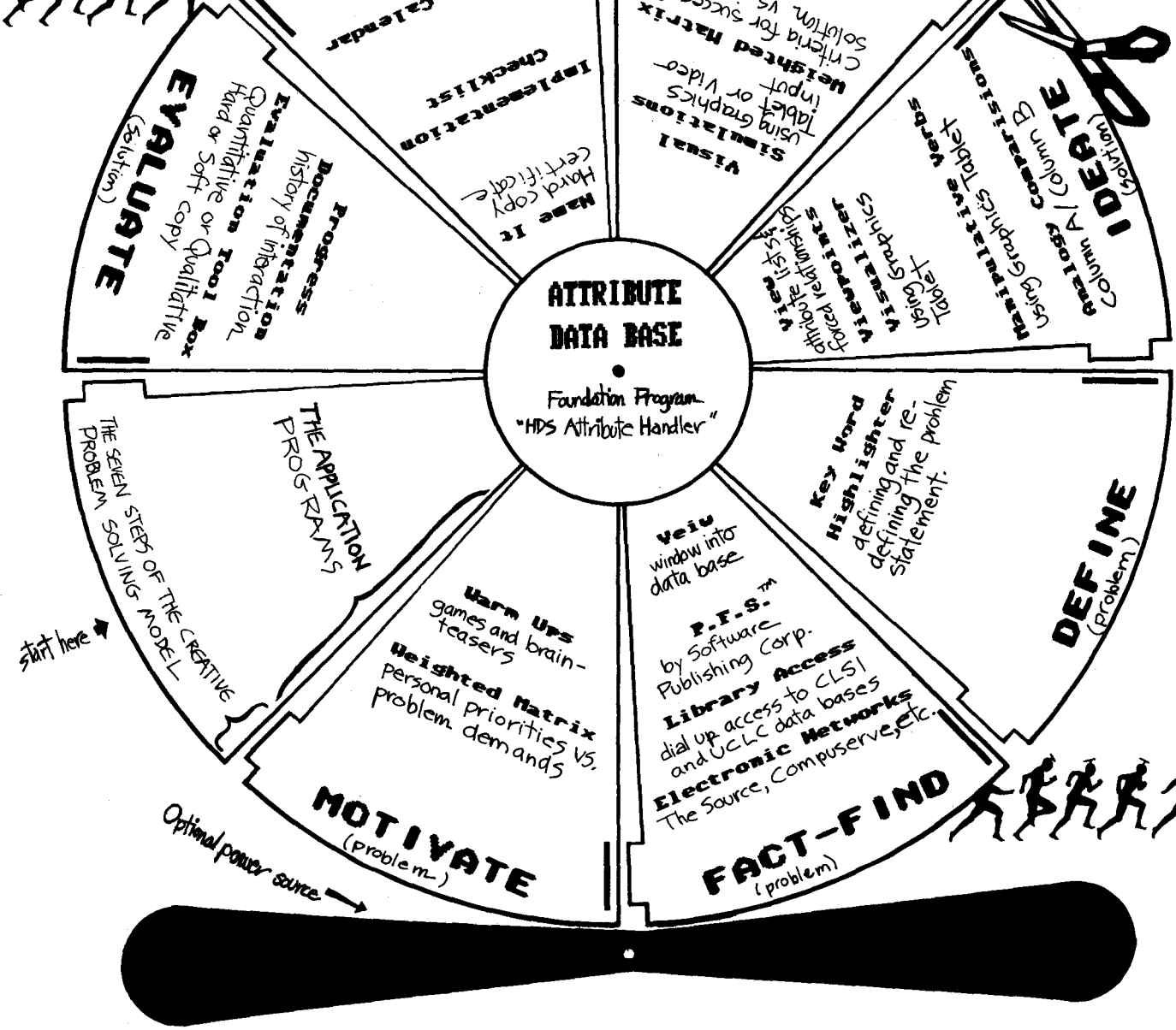
Electronic Thinking Cap

COMPUTER ASSISTED CREATIVE PROBLEM SOLVING

Susan Metros
Assistant Professor of Art/Graphic Design
Michigan State University

Directions...

1. Cut along solid lines
2. Insert tabs into slots
3. Find something to connect propeller (optimal power source) onto top of cap.
4. Read following page



Electronic Thinking Cap

COMPUTER ASSISTED CREATIVE PROBLEM SOLVING

is a playful visualization of a model for computer assisted Creative Problem Solving (1). Creative Problem Solving is an essential element in the Graphic Design curriculum at Michigan State University. The mission of our program is to educate students in the analytical and process phases of design. This kind of backround prepares students to successfully fulfill the demands of a competitive job market that seeks creative and innovative professional designers. The instruction stresses process over product by teaching the students to actively pursue creative solutions to design problems rather than passively assimilate knowledge.

Just like traditional Creative Problem Solving methods, the computerized method employs gaming techniques to playfully coax the user into manipulating information in ways that new combinations and fresh ideas surface. The advantages of the computer assisted method are as follows: 1) information can be logically and sequentially processed. 2) a quantity of alternatives can be generated in a relatively short period of time. 3) one-to-one interaction is always available. 4) an objective appraisal can be used to determine the ranking of idea options. 5) students gain important skills related to computer literacy.

The specific work being done at Michigan State University demonstrates how the micro-computer can be used to assist designers and artists in improving and learning Creative Problem Solving skills. The software being developed, along with commercially available software, provides the user with a systematic yet rapid approach to generating alternatives for specific design related challenges.

The system consists of a foundation program (the center of the electronic thinking cap) and numerous application modules that radiate out from the foundation program. The foundation program is an attribute data base called, "HDS Attribute Handler". It has the ability to maintain, modify,display, and eventually string together attributes entered by the user. The application programs are individual "games" or methods which playfully coax the user into manipulating information, either directly inputed or retrieved from "HDS Attribute Handler, in ways that new combinations, and fresh ideas surface. In some cases commercially available software is being incorporated and in other cases we are developing programs to fulfill specific needs. The panels of Electronic Thinking Cap list these programs and how they relate to the individual steps of Creative Problem Solving.

Note: The final step under "Directions" was omitted. It should read, "5. Insert head."

(1) The traditional Creative Problem Solving model used for this research is a synthesis of the work done by,
Don Koberg and Jim Bagnall,
The Universal Traveler (Los Altos, CA: Wm. Kaufmann, Inc.,1976)
Sid Parnes, Ruth Noller and A.M. Biondi,
Guide to Creative Action, (NY: Charles Scribner's & Sons, 1977)
and personal investigation.

State-of-the-Art Questions

Laurie Spiegel 175 Duane Street, New York City, NY 10013

August 1983

Among those of us who used to be an original lunatic fringe of artist-hackers, who got lost in the wilderness crevices of the computer-and-art intersection, a surprising percentage have by now been vindicated as avant garde instead of crazy, ahead of our times rather than parallel or right-angled to them.

This vindication has taken form in the integration of our work into commercial business contexts more often than by its integration into the mainstream worlds of music and art. This is understandable, as our work, like any other new art movement, breaks with "art world" traditions, in this case obscuring boundaries between such previously separated realms as art and music or science and art, and between such separate output forms as artists' tools and works of art per se.

But this one-sided vindication is also regrettable, as high tech's potential of wealth and glamour, and business's more abundant financial assets overshadow Art's nebulous (and dubious) attractions with other distracting images of reward. What seems to pose threats within the mainstream artworld offers profits in the world of business, and so our work as artists may be sidetracked by offers of homes other than those to which we had intended to go.

A surprising number of us are now welcomed by companies which pay us to do work which is remarkably similar to what we used to do on our own, and at our own expense, to be considered crazy for doing. Computer images and sound have changed from basement inventors' midnight insanities to viable business. Large companies can make large amounts of money by plucking the fruits of what was till recently only a rather bizarre underground rootwork. The Art gets lost in the picture, and possibly the artist too. Paradoxically, we also have unprecedented support available for our work.

The "state of the art" poses new and unforseen questions at every stage of its advance. Many of these questions were not asked within vector spaces whose dimensions were purely technical, structural and aesthetic, and whose incentives consisted almost entirely of intellectual, emotional, and artistic highs. These questions, new in a new context and for the toolmaker, gamewriter, and software artist, echo their counterpart queries as found in the traditional artworld, in which "commercialization, co-optation, and compromise" and many other all-too-familiar terms already refer to classic dilemmas.

Ironically, it is a mark of our new arts media coming of age that they have evolved to the point where such questions apply. Since they do apply, they must be stated, discussed, and thought through by each of us.

This readiness for a sort of conceptual adolescence strikes me as the most important recent development in the "state of the art" despite an amazing number of other advances. The computer arts are beyond their nursery, but far from being maturely placed in the world.

For those of us who are addicted to challenges, the greatest questions are those that can't be answered. ("Stupid questions" are those which are too easy.) The following are questions which I have asked myself or heard other people ask. I list them here to clarify, to help generate thought, discussion, and exchange, and to predict or forewarn for those who haven't encountered them yet. I also list them because they constitute an important aspect of the current state of these arts which needs and has received insufficient documentation.

My list bears no pretense of universality or comprehensiveness. Each of us would make a different list.

Computer Arts Businesses:

In the long run, the largest perspective, is it in the best interest of these new media to provide more people with less powerful lower cost tools, or to endow a smaller number of individuals with much more powerful creative and expressive potential?

If it seems you are getting paid at last for what you used to have to do alone at night at your own expense, is what you are doing now really what you were doing then? How is it different? Are you being paid, rewarded, supported for being yourself, or for not being yourself in some fairly subtle way?

Who "owns" the "rights" to our ideas, inventions, and creative output ("software art")? Those who do the creative thinking? Those who do the drudge work? Those who subsidize and provide tools, money, and distribution? Those people out in the world to whom our creations might really mean something or be really valuable?

How can we balance the amount of time we spend making computer arts tools, the time spent making art or music per se, and the increasing amount of time that successful commercialization of the field seems to require for the "3-D" reality of: Demonstration, Documentation, and Distribution?

How can we continue to openly exchange ideas, to educate and help each other, to synergistically raise the momentum of accomplishment together, as more and more of us affiliate with separate companies in the private enterprise sphere? Much that we create or learn from may be subject to "non-disclosure" agreements. Even if it weren't, business thinking encourages us to be afraid that the other guy will steal anything we mention. How can we prevent paranoia from destroying free open exchange?

If we decide to publish in the public domain, we can protect our ideas from ever being the exclusive property of any interest and guarantee their open access by all. It also undermines our own self-support for the furtherance of our work, and equally guarantees free access to our inventions by the same interests who have brought us Muzak and the "30 second spot." What are the pros and cons of private ownership of intellectual property?

If private companies are a new breed of patrons for the arts and sciences, if we don't safeguard our creations for them the way they wish, will they (or any other potential patrons) continue to be in a position to be able to provide needed subsidy for the creative research and development we believe in and want to do?

What are the differences between employers, patrons, and collaborators?

What are the trade-offs in working with tiny low budget computers while keeping total freedom and independence, versus losing the latter but getting further in in the work via tools we could never afford as individuals? What are these trade-offs for each of us individually? What are they for the whole field, technically and artisticly?

To what extent do our own personal and artistic interests coincide with those of the companies we work for (or start, run, and own ourselves)? To what extent do we find ourselves overlooking the difference because it is easier to be in situations where our work is wanted, waited for, rewarded, and well-equipped for progress, than to be free, independent, visionary, and broke, frustrated, and unwanted?

Can visionaries and idealists succeed in "business" without being somehow changed in ways that may not have been forseen or wanted?

Computer Arts Tools:

How do the best systems organizations for artistic use of computers differ from those which have been evolved for other purposes, such as business or science?

How do the purposes of computer arts systems differ from the purposes of other kinds of computer systems?

How does the process of artistic creation differ from other interactive processes to which computers have been adapted?

How can we convey these differences to other computer technical professionals who we may be working with or who may be hired to make instruments by companies, but who have no personal experience of doing music or art? (This may be especially problematic for those who have artistic backgrounds but are self-taught and/or without credentials or standard vocabulary in technical areas?)

How much more (and what) should a designer of computer-based tools for the arts know about these arts than a maker of canvas, brushes, or pianos needed to know to make the (simpler?) tools for these older arts?

How much does knowledge of established technique help, and how might it hinder, the development of new techniques? (How does knowing history improve the quality of our actions (works), and how does such knowledge inhibit us?)

How can a logical medium best accommodate the intuitive, the irrational, the emotional, the spontaneous and the unpredictable?

How do sequence and simultaneity intermix in the mind of the musician during the creative process? How can we make both of these dimensions equally accessible at each moment? (Most computer music systems push people toward the sequential dimension by defining musical voices ("tracks" or "channels") which must be entered sequentially as in multitrack tape, and also require to sequential specification of parameters which are to be simultaneously perceived, instead of allowing all sounds or sound parameters to be defined together if they are going to be heard together.)

How does the overview of a picture in progress intermix with concentration on detail in the mind of a visual artist? How is the "shorthand" of sketching used?

In creative processes which artists and musicians conceptualize in stages (or multiple "passes"), what are those stages? (Rough sketch, light and shade, edge definition, rendering, color; chord sequence or melodic line, voicing and orchestration, articulation and dynamics ...)

What do artists and musicians dislike or feel constrained by in traditional media? What would they be willing to learn new techniques to be free of (or free to do)?

What has prevented each of us from going further than we have in our own work in "conventional" arts media?

Why did we turn to computers in the first place instead of sticking with more traditional arts media?

How are the tools we create for others to use different from the ones we create for our own personal use in doing art or music?

At what points and in what ways do we choose to sacrifice generality and flexibility for specialized power in a creative system or tool? In what situations do we make the opposite trade-off?

If we design a tool dedicated to a highly defined application, how do the processes and data structures best for that specialty differ from those more general to the medium (musical composition versus spontaneous improvisation, design versus drawing)?

What kinds of individuals do we envision using the systems we produce? How much learning time do we expect them to put in? (In the old days anyone serious about the arts expected to invest quite a bit more than a single hour in learning to use that art's tools. But then again, few attained mastery.)

To what extent and in what ways might the tools we create be too "complex" for the "average" user? To what extent are they really just too personal or personalized? To what extent does a decision to reach more people with less of a custom fit, or to cut invested time or cost, or get a system out to "market" faster motivate a reduction in complexity or power (e.g. the number of options to learn and use) when we go from writing for our own use to writing for distribution? To what extent are we trying to make it easier to use, versus making it easier to create or to market?

To what extent can each person be equipped or expected to create their own art or music (expression, satisfaction) instead of relying on a small number of "talented" specialists who create images and music for all? (What do "talent", "masterpiece," and "amateur" mean? Is our goal to make masterworks more possible or to increase the amount of pleasure people get from the process of doing art? If both, how do we balance them against each other, and how might the tools for each differ?)

To what extent do we (or should we) judge what we create by its peer context versus judging in terms of some ideal? ("This is amazing for such a small computer" versus "How does this fall short of how we can envision an ultimate for human expression?" "More commands than others" versus "How would someone say this in natural language or gesture?")

Do we get further by starting simply and then adding features as we need them, or by designing a totality that would satisfy all our envisioned needs and choosing what subset we will implement? What complexity or quality of organization might make one of these options preferable over the other?

How can one avoid getting so involved in the beauty of the tool (program, etc.) one is creating that one gets out of touch with the purpose of the tool? If the two conflict with each other, which tends to get priority in a trade-off? Why?

Is distinction between tools to make art and actual 'artwork' still a valid distinction?

In what ways can new technologies reduce the need for extensive training in artistic composition or self-expression? Just in expediting the assembly and generation of material, or in the heightening and refinement of sensitivities as well? How else?

Computer Arts:

What will be the differences between the arts of the past and those of the future? What will be the same?

Will art and music continue to be fields of professional specialization in the future in this society if new tools permit easy generation of material without extensive (time-consuming and often expensive) training and physical coordination skills?

To what extent is one considered to be an artist or musician in this culture because of:

1. mastery of specialized technical skills
2. sensativity to a medium
3. completion and outputting into the world of finished works, regardless of their technical sophistication or of the sensativity or expression embodied in them?

Is it easier to rely on external considerations such as audience, market, or 'fashion' to structure our creative output, or to let inner directions and interests structure our work? For some, conflicts with the environment are harder to tolerate, and for others, conflicts with inner impetus are more intolerable.

Forget the computers. What is art? What is music? What do we need or want them for? What drew us to them when we first felt, heard, or saw them? What parts of us do they touch, intrigue, excite, or open up? What are the structures of these parts of us? How do we make new languages to describe and tools to generate what fits these parts of us?

What difference is there between graphics and art? What difference is there between audio and music?

Is there a difference between what we started trying to do once and what we're doing now? What is it? Have we evolved, or just forgotten?

What differences are there between my own computer music and art and the music and art that I do without computers?

What differences are there between my art and music as I have produced them and the internal visions and feelings that I have tried to embody in such forms?

How is my work different from the work of those who are doing things most similar to what I do? In what ways is my work similar to that of those who appear to do these things most differently from me? How do I want to be different? How do I want to be the same? How am I the same or different? From others? From how I see myself?

In what ways do I value the final results I complete and for how long? How do I value the process of doing, creating? Which is more important to me, doing or having done the software, music, or art? When is one more important to me than the other? What triggers the change back and forth?

To what extent am I creating for myself versus for others? When do I forget them and lose myself in the doing for long periods? How does the awareness of my audience (market) influence the work I am creating? How do immersion and self-consciousness alternate or mix? How do they compete or conflict? How do they refine or propel each other?

How can objectivity and subjectivity be maintained in balance during the creative process, so that neither overshadows the other?

What moves us emotionally in music and art? What excites us intellectually? What feels gentle, grating, or violent to the senses? What inspires us as beautiful? What reminds us of sadness? What textures, images, and archtypes do we see when we look inside ourselves instead of out?

What have we experienced in common but never had means to point to or communicate?

What are we trying to do that we have been unable to do?

What are we able to do now that we have never been able to do before?

What can each of us do beyond what we've already done?

SOUND SYSTEMS ON MICROS

Susan E. Jenkins

Instruction and Research Computer Center
The Ohio State University

Abstract

This paper examines microcomputer-based music systems and their applications for music theorists and composers. The examination is based on first-hand experience with the microcomputers located in the microcomputer laboratories at The Ohio State University, and during demonstrations of equipment at the Instruction and Research Computer Center workshops.

The use of computers is increasingly important in theoretical research, music analysis, instruction, and composition. Traditionally, composers had to have detailed knowledge of digital sound production in order to synthesize sound. Theorists had to become involved with job control language, system utilities, and high-level languages to use the computer for their research. Both composers and theorists either had to share a large system not designed specifically for their needs, or invest thousands of dollars in their own system.

Today, these circumstances are changing. Technology currently exists that enables a person to obtain useful results without being a programmer or an electrical engineer. Nor will cost continue to be as prohibitive a factor in computer use. The technology making this possible is that of the microcomputer.

INTRODUCTION

In this presentation we will be concerned with introducing music applications on microcomputers. There has been some research recently concerning voice synthesis on micros, but it will be covered only briefly.

Briefly about Voice Synthesis

Voice synthesis for micros is generally of a different origin than that of other sound generation systems. Presently there are three categories:

* Analog-digital converters like Codex
* Linear predictive modeling like Speak and Spell and Voice Synthesis from Texas Instruments
* Speech synthesis from phonemes like BS232 Votrex Box

Music and Sound

Music and sound capabilities on micros were considered frills until recently. Sound capability was often known to many as one of the micro's many "bells and whistles." In other words, having sound on a micro was something extra. Most will agree that this is true in many instances today. Nevertheless, it is possible to use seriously the sound capabilities of many micro systems. Even professional musicians are now admitting that micro systems have many uses, both musical and nonmusical.

REASONS FOR USING MICROS FOR SOUND

Some of the ways micros may be used for sound include:

* Composition and arranging - immediate feedback without having to wait for performance by other musicians
* Storage of melodies for protection against copyright violation
* As a data base for storing compositions (a library of your own works in electronic form)
* As music copyist and printer
* As an alternative to purchasing a piano or keyboard instrument
* In music education
  - drill and practice
  - music dictation/reading
  - developing aural skills
* Performance as an instrument itself
* Exploration of new sound capabilities
* In theoretical research
  - perception
  - psychoacoustics
  - signal processing
  - sound synthesis

WHAT IS A SOUND SYSTEM FOR A MICRO?

Combination of hardware and software

* Some method of inputing music from the beginning score; there are three types:
  - coded-entry
  - instrumental keyboard
  - non-traditional
* Some method of editing musical score files - may or may not be the same method as mentioned above
* Documentation and programming aids
  - manuals
  - preprogrammed songs
  - demonstration records and tapes
  - tutorials
* Graphics on monitor used to see what notation is entered; usually this is some sort of piano score
* Other things you can sometimes expect:
  - ability to repeat sections without re-entering notes
  - transposition of entire piece, voice or section
  - control of tempo and dynamics
  - sequenced playback - file merging, copying moving
  - control of 'instrument' sound or timbre
  - control of envelope
  - wide range between accuracy of notational capabilities on individual systems

A CLOSER LOOK AT SOUND SYSTEMS

PARAMETERS INVOLVED

* waveform
* voices (2-16)
* envelope - attack, duration, decay
* channel
* range frequency, pitch range
* amplitude control, volume
* tuning accuracy
* stereo
* filtering capabilities
* timbre control
  - you choose between preprogrammed "instruments"
  - you program your own "instruments"
* more advanced techniques (modulation, reverberation, etc.)

SAMPLE SYSTEMS

Casio Music Machine

Name:	Casio Music Machine
Use:	Melody instrument, for fun, performance
Manufacturer:	Casio
Price and Format:	Keyboard with chip $35-$300
Computer Model:	Several models
Documentation:	Short booklet
Audience:	Hobbyists, performers

Atari

Name:	Music Composer
Use:	Composition, arranging performance
Manufacturer:	Atari Inc.
Price and Format:	ROM cartridge $59.95
Computer Model:	Atari 400 and 800
Memory:	16K RAM
Documentation:	21-page booklet
Audience:	Hobbyists, composers

Commodore 64

Name	Music System
Use:	Composition, performing for fun
Manufacturer:	Commodore
Price and Format:	3-voice chip included with original system software

Computer Model: Commodore 64
Memory: 64K

Documentation: A chapter in the Programming Guide/Tutorial

Audience: Hobbyists, composers

Mountain Music System

Name: Mountain Music System

Use: Composition, arranging, theory, teaching

Manufacturer: Mountain Computer, Inc.

Price and Format: Music boards, software (diskettes), speaker cables, light pen $545

Computer Model: Apple II
Memory: 64K

Documentation: 2 manuals, updates supplied periodically

Audience: Composers, performers, music professionals.

AlphaSyntauri System

Name: AlphaSyntauri

Use: Composition, theory, in-struction, printing music performance

Manufacturer: Syntauri Corporation

Price and Format: keyboard, software, 4 octaves, $700; 12 octaves $1200; Apple II interface $30; Apple II cable $35; Musicmaster, Theory (diskette) $150; Draw Waves (diskette) $30; Composer's Assistant (diskette) $295

Computer Model: Apple II
Memory: 64K

Audience: Composers, performers, music professionals

Synclavier

Name: Synclavier II

Use: Music professionals wanting the most control over timbre and note creation, high resolution graphics, digital readouts of hertz and decibel levels, one-stroke transposition of key, double command erase button

Manufacturer: New England Digital

Price and Format: 16-voice 32K synthesiser and software $19,750 Add Winchester disk drive, dot-matrix printer, and graphics terminal to total $40,225

Computer Model: Dedicated system

Audience: Composers, professional laboratories, millionaires

GLOSSARY OF TERMS

amplifier - voltage-controlled amp is usually several signals added together in equal proportions; results in volume control

amplitude - is related to the height of a waveshape; measured in decibels (dB); strong relationship between amplitude and volume

analog-to-digital conversion - for the input of manual data; envelope follows, pitch trackers, filter banks, or audio signals converted to digital form; most micros use monostable multivibrator

analysis - thoroughly examining sounds determining the sounds' fundamental parameters and how they vary with time

analysis-synthesis - creating sounds as a result of understanding the relation between parameter change and audible effect

bandwidth - used to describe filtering techniques; refers to a specific range of frequencies

channel - the method many micro systems use for creating a stereo effect; system can have 2-16 channels

digital oscillator - offers two main advantages over conventional voltage-controlled oscillators; enhanced frequency accuracy and greater waveform variety

digital-to-analog converter - one of the most important parts of a digital sound synthesis system; converts digital data to an analog audio signal

display - most systems include some method of displaying the sounds during editing sessions or even during playback; some sort of graphics on the computer monitor

editing - method for inputing musical score or code on system; by keyboard, typing, or non-traditional methods

envelope - identification of changes in amplitude; what separates one note from another; consists of attack, sustaining of shape, and decay with time

filter - a device which acts directly on the spectrum changing the amplitude and phase of each wave component of the input by a predictable amount;

basic filter shapers: low-pass, high-pass, band-pass, band-reject

formant - part of the essential character of a sound; resonant peaks of a sound

Fourier transform - way of connecting the time domain (waveform and sample values) with frequency domain (amplitude and phases)

frequency - is measured in cycles/second according to the number of times a waveshape repeats itself (in hertz, Hz; or kilohertz, kHz); strong relationship between frequency and pitch

instrument - in "micro talk" commonly refers to a programmed sound with a certain timbre; most systems offer choices of preprogrammed sounds referred to as instruments

modulation - amplitude, using the amplitude of one waveform to control another; frequency, using the frequency of one waveform to control another

keyboard - most mid-priced to expensive systems include electronic keyboard for performance and editing; of various lengths and sizes; sometimes independent of CPU

noise generator - makes 'noise' analogous to white light, an even mixture of all frequencies

orchestra - sometimes (in MusicIV) refers to a set of sub-program calls generating a desired group or block of sounds or instruments

phase - shift of a particular frequency determined by comparing two waveforms on the oscilloscope face

reverberation - method of provisions for mixing delayed and undelayed sounds; audible effect-some kind of echo

sample - to convert sound in time to digital representation; an individual pulse or number which gives the waveform amplitude at a certain point in time

spectrum analysis - a kind of graph helpful in indicating timbre; horizontal axis is frequency, vertical axis is amplitude

stereo - not found on most small micro systems; separation and cross tracking of sound

teleharmonioum - one of the earliest musical instruments which produced sound by purely electrical means; built by Thaddius Cahill in 1903

timbre - an aspect of sound which assists in identifying it; sound quality

tuning - equal-temperment is used almost exclusively in micro systems; all half steps the same size

voice - in "micro talk" one way of referring to a single melodic line of music; not necessarily independent from other sounds occurring simultaneously

waveform - a method of visualizing sound as changes of air pressure with time

ANNOTATED BIBLIOGRAPHY

Adams, Christopher. "Sound Table: Fast Sound Effects from BASIC," Creative Computing, July 1983.
Assembler approach to making sounds.

Buxton, W., et al. "A Microcomputer-Based Conducting System," Computer Music Journal, Autumn 1980.
Description of system for performer to interpret or conduct precomposed scores.

Ciarcia, Steve. "Add Programmable Sound Effects to Your Computer," Byte, July 1982.
Includes instructions for hardware alteration and programs.

Edwards, John. "Highly Personal Music," Popular Computing, June 1983.
Good introduction to software for Apple, Atari, and Radio Shack computers.

Gold, Jordan. "Keyboard Charisma - Making Music by Computer," Personal Computing, May 1982.
Uses for personal enjoyment, includes buying guide.

Hammond, Ray. "Musical Computers," Music and Sound Output, September/October 1982.
Good survey of dedicated systems; aimed toward the performer/composer

Hiller, Lejaren. "Composing with Computers: A Progress Report," Computer Music Journal, Spring, 1982.
A review from one of the first to use computers for composition of his own works.

Haynes, Stanley. "The Computer as a Sound Processor: A Tutorial," Computer Music Journal, Spring 1982.
Good technical survey of work at Institui Recherche et Coordination Acoustique/Music and in France.

Jones, Kevin. "Computers and Musicians in Concert," New Scientist, 27 August 1981.
Good introduction for novice includes a brief history of computer music.

Lubar, David. "So You Want to Buy a Music Synthesizer," Creative Computing. Winter, 1982.
Sample systems and brief explanation of some terms.

Shore, Michael and McCain, Larry. "Computers Rock the Music Business," Popular Computing, June 1983.
Very good survey of commercial uses, and description of three types of entry systems.

Smith, Patricia. "Computers Make Music," Creative Computing, July 1983.
Investigates trends in computer music especially those in California.

Tubb, Phillip. "Pick and Choose," Creative Computing, Winter, 1982.
Somewhat biased but informative comparison of ALF, Micro, and Mountain systems; awkward conversational writing style

LIST OF SAMPLE SYSTEMS

Andex, Sirius Software, Inc., Apple II, $29.95.

Apple Super Music Synthesizer, Applied Engineering, Apple II, $159.

Compu-Music, Poland Corporation, Apple II Plus and IIe, TRS-80 I and III, IBM PC, NEC PC-6000 and -8000, cost?

Electronic Duet, Insoft, Apple, $29.95.

Emulator, E-Mu Systems, dedicated system, $6,400-$8,000.

Fairlight, CMI, dedicated system, $27,500.

Micro Music, Apple, collection of software and board, $50-$400.

McLeyvier, McLeyvier, dedicated system, $25,000.

The Music Box, Newtech Computer Systems, TRS-80, $149.

The Music Card, ALF Products, Inc., Apple II, MC-1 $195, MC-16 $245.

Music Maker, Sublogic Communications Corp., Apple II, II Plus, and IIe, $49.95.

Music Synthesis System, Micro Technology Unlimited, Pet, AIM, KIM, Apple, $50-$89.

Orchestra-90, Software Affiar, TRS-80 Model III, $149.95

Piper, Abacus Software, Vic 20, $22.95.

Prism, Kinetic Sound Corporation, dedicated system, $49,000.

Soundchaster Digital, Passport Designs, Inc., Apple II, IIe, $1250.

Roland CMU-800 and Compu-Music, Roland Corp. US, Apple II, NEC, $465.

Synthia, Adaptive Systems, Inc., dedicated system, $28,600.

Vic Music Composer, Thorn EMI, Vic 20, $39.95.

QUADSURF: A GRAPHICS SYSTEM FOR 3-DIMENSIONAL HALFTONE IMAGES

Steven R. Seidel

Mathematical Sciences Department
Virginia Commonwealth University
Richmond, Virginia 23284

ABSTRACT

A graphics package for producing halftone representations of 3-dimensional objects has been developed for the Apple II+. This package can display any quadric surface (a sphere, cone, cylinder, etc.) as it would appear when illuminated by a point source of light. The shadow cast by the surface is also shown. The halftone effect is obtained by constructing the surface of white points statistically distributed according to the proportion of light reflected from each element of the surface to the observer's eye. Within the limits of screen resolution this technique yields an image with appropriately highlighted and ambiently lit regions and smooth transitions of brightness between such regions. The package is written in UCSD Pascal and includes facilities for storing and retrieving complete images from disks, modifying images, and a limited form of animation.

0. INTRODUCTION

An interactive computer graphics package, called QuadSurf, has been designed that is capable of producing halftone images of certain types of 3-dimensional objects. This package was written for the Apple II+ and was designed for inexperienced and non-mathematically oriented users. The class of 3-dimensional objects the package can draw are called quadric surfaces. Common examples of such surfaces include spheres, cones, footballs, cylinders, and saddle-shaped objects. An illusion of depth is created by constructing the surface with individual points of light, where the density of points varies over the surface in a manner analogous to that of true halftone images. The surface is set in a simple background consisting of a ground surface that recedes to infinity and a "sky" above. The shadow of the object is shown as it falls on the ground surface.

The graphics package includes an interactive environment to facilitate creating and maintaining a collection of images. It provides for the "quick and dirty" creation of images so that image composition can be judged quickly and it has facilities for automatically rendering full-scale versions of images without user intervention. The latter facility is useful since a typical full-scale image takes several hours to compute and display. The system also provides for the high-speed sequential display of previously created images so that a limited form of animation can be obtained. The environment provided for the user requires no knowledge of programming or operating system usage. The package is written in UCSD Pascal and requires an Apple II+ microcomputer with the Pascal language option.

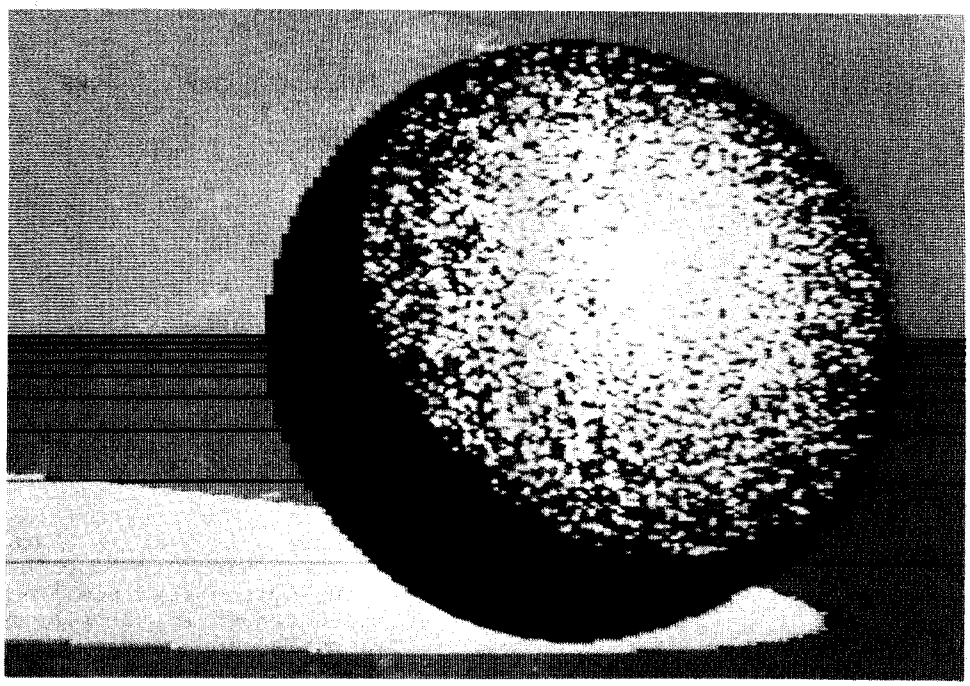
The following three sections describe in more detail the kinds of images that can be produced by this package, the interactive environment provided for the user, and the limitations of this system and its potential transportability to other computers.

1. IMAGES

This section informally describes the variety of images that can be obtained using the QuadSurf package. The name QuadSurf is a contraction of quadric surface, the mathematical name of the class of surfaces this package can produce. Throughout this discussion the word "image" refers to the entire contents of the graphics display screen. The word "surface" refers to the object shown in the image, namely, the quadric surface apart from its background. Figures 1 and 2 are examples of images produced by QuadSurf. In those figures each surface is composed of black and white points, the "ground" is green with black horizontal lines, the "shadow" is violet, and the "sky" is blue. The selection of alternate colors for the elements of an image may be be available in a later version of this package.

In general, the surfaces QuadSurf can

CH1930-7/83/0000/0041$01.00



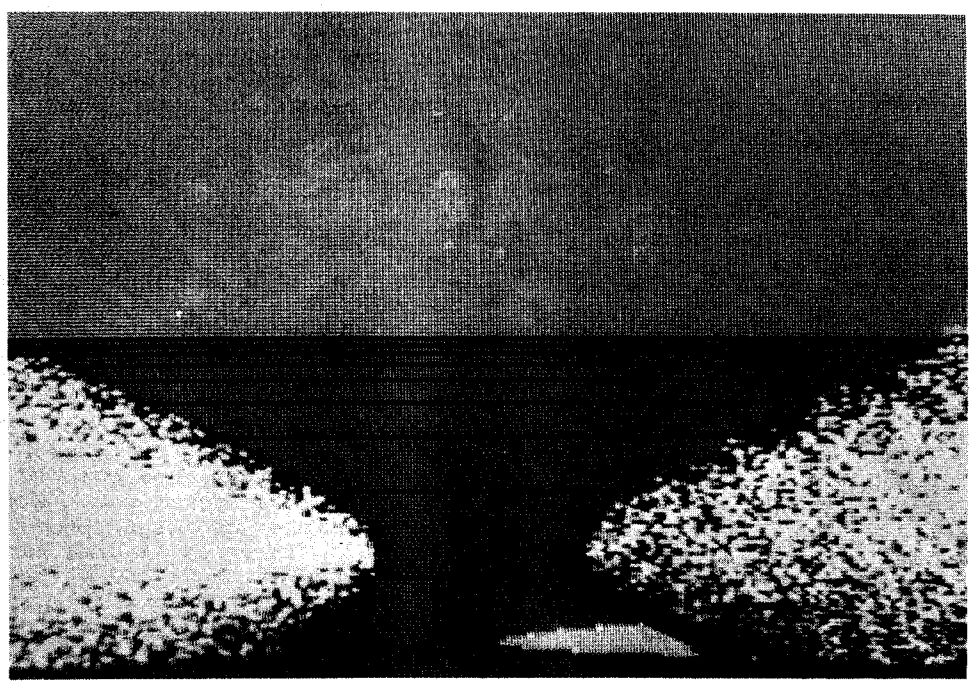


Figure 2

produce are smooth and very symmetric. A more detailed classification of quadric surfaces is given in the Appendix. Here it will suffice to think of them as cylinders (possibly smoothly gathered in at the "waist"), spheres, footballs, infinitely deep smooth-sided mixing bowls (perhaps with their mirror images), cones (perhaps with their mirror images), and "infinite" saddles. All but the last of these can be of circular or of ellipitical (oval) cross section.

The QuadSurf package can draw any one of these surfaces in a given image. The surface is presented in relation to a "ground plane" with horizontal lines across it to give an illusion of depth. Any part of the surface that lies below the ground is hidden from view. Figure 2 shows how the bottoms of the two "pods" are cut off by the ground. The upper half of the background is a simple "sky" that lies at infinity.

Figure 3 shows the observer's position with respect to the elements described above. The observer is centered behind a viewing window. The ground is perpendicular to the window and the bottom edge of the window lies directly on the ground. A light source above the ground illuminates the surface and casts its shadow on the ground. Only that portion of the surface above the ground and on the side of the window opposite the observer is seen by the observer.

The observer's environment is described to QuadSurf using the conventions of a 3-dimensional Cartesian coordinate system. As indicated in Figure 3 the x axis of the coordinate system runs right and left, the y axis runs parallel to the observer's line of sight, and the z axis runs up and down. The coordinate arrows in Figure 3 indicate the positive direction along each axis. The user may specify the location of the window anywhere in the x-z plane, that is, anywhere in space as long as the window is perpendicular to the y axis. No corner of the window need lie on the origin as might be implied by Figure 3. The user specifies the location of the light source by giving its x, y, and z coordinates. Finally, the user describes the surface to be displayed by giving numerical values for the ten coefficients for the mathematical equation of the desired surface. In practice, most of those coefficients are zero so the user need only reach an understanding of how a small number of them determine the shape of the surface. The effects of these coefficients are summarized in the Appendix.

As mentioned earlier, the observer sees that portion of the surface that lies above the ground and falls on the far side of the window. The shadow cast by the surface onto the ground is determined in a natural way by the location of the light source. The manner in which the surface appears to the user is determined as follows. The window, which corresponds to the physical graphics screen, is uniformly divided into equal-sized picture elements (pixels), the smallest discrete graphic elements that the computer can distin-

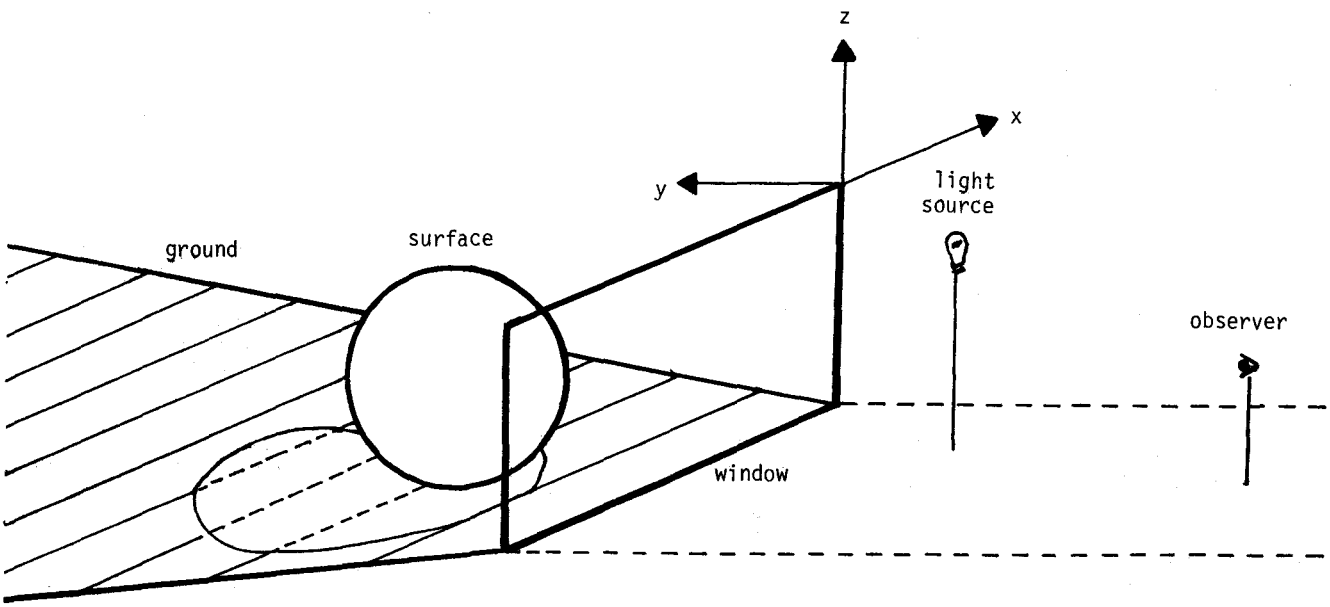


Figure 3

guish. For each pixel through which the observer sees a portion of the surface QuadSurf determines the direction in which a beam of light from the light source will be reflected off the surface at a point within that portion of the surface. The angle between the relected beam of light and the line of sight of the observer determines the brightness with which the observer will see that portion of the surface. Since the color values of pixels on the Apple graphics screen cannot be varied in intensity, variations in brightness over the surface are simulated by statistical means. Simply, the more nearly the beam of light is reflected to the observer's eye the greater is the liklihood that the pixel observed will be white. Since the resolution of the Apple graphics screen is so low (i.e. pixels are large) this approach yields a coarse stippling of the surface that only approximates a high quality halftone image. A discussion is given in Section 3 of the potential for transporting that portion of the QuadSurf package responsible for producing images to other, higher resolution graphics systems.

2. USER ENVIRONMENT

The user environment is the microcosm created by the computer for the person using the computer. Just as a video game creates a world of its own, every computer system and every program run on a computer system creates an environment based on the manner in which the user interacts with the program or system. QuadSurf is intended for inexperienced and non-mathematically oriented users and so was designed to be easy to interact with and forgiving in nature. The creation of such a user environment posed several problems. Although not all of these problems have been solved significant progress has been made since the earliest versions of this package were developed.

There were three main problems in creating the desired user environment. First, any surface to be drawn by the package is described by a complex-looking mathematical formula, a second-degree equation in three variables. Such an equation has ten coefficients whose values must be supplied by the user. Steps were taken to make this process as painless as possible. Second, the limitations on the computational speed of the Apple result in the fact that it takes several hours to generate full-scale images such as those shown in Figures 1 and 2. Such slow "response time" is discouraging to the typical user because it makes experimentation tedious and delays gratification. Finally, given the high cost in time of creating images it would be unacceptable if images could not be saved, only to have to be recomputed "from scratch" to be seen again. This section describes how these and lesser issues were dealt with in the design of QuadSurf.

Three features are provided to minimize the intimidation and confusion that might result from requiring the user to supply values for the coefficients of the quadric equation and a few other numbers that determine window dimensions and the location of the window and light source. First, default values that will yield an image similar to that of Figure 1 are initially provided for all of the variables. These default values give the user a secure base from which work can begin. The user is encouraged to change just one or two of those values at first in order to learn how those values affect the image displayed. This provides the opportunity to begin slowly and progress at a self-determined pace. As usage proceeds the default values are taken from the most recently displayed image.

Second, a technique called scanning on the fly [1] is used to guarantee that the user does not inadvertently supply values that will sooner or later cause program failure or other undesirable behavior. This technique prohibits the user, in a reasonably friendly way, from ever typing anything that might lead to unfortunate results. In fact, this technique is used throughout the package but its most significant use lies here.

Finally, the user is presented with a fixed-format screen for the purpose of entering the required numbers. This type of screen display does not roll up after a line is typed as is typical on many systems. The screen display consists of labelled fields that are to be filled in by the user. The default values described earlier initially occupy the fields. The user is free to jump to any one of the fields and alter its current contents by using a small collection of commands presented at the bottom of the screen. This design avoids the shortcomings of the usual scrolling of the screen since none of the numbers that determine the image ever disappear past its top edge but are always visible at the same location on the screen. In addition, one of the editing commands allows the user to flip the display between a page of instructions and the data entry page.

The second major difficulty in the design of the user environment was the unfortunately heavy computational effort the Apple must expend to compute a single image, typically several hours are required. To minimize the impact of this delay on the user a facility is provided for producing small-scale renderings of images. These are simply miniature versions of full-scale images and so contain proportionally less detail but take much less time to compute and dis-

play, typically a few minutes. The user is given the opportunity to determine the most suitable trade-off between the size of the image and the time it takes to compute it by choosing the relative sizes of these thumbnail sketches. When the user has created the desired image through the use of such sketches a facility for automatically generating a full-scale image can be invoked. This allows the user to have the computer work overnight or during other low demand periods. A future version of the QuadSurf package will allow the user to have a collection of up to 17 full-scale images computed automatically so that the computer can be left to do its heavy work, say, over the weekend.

Finally, in light of the high cost of computing a single image it would be unacceptable if there were no means of preserving images once they were computed. QuadSurf provides a simple facility for saving images (of any size) on a disk and for subsequently displaying those images whenever desired. It takes less than one second to transfer a stored image from a disk to the grahics screen so the user can easily review the fruits of his or her labor. A number of other features are provided to help the user maintain a collection of images. Of greatest importance among these is that the parameters used in the creation of each image are stored along with the image on the disk. Since about twenty numbers are used to determine the properties of an image it is essential that the computer itself maintains such a record. As mentioned earlier, QuadSurf uses the parameters associated with the most recently displayed image as the default values for the fields in the data entry portion of the package. In this way the user can recall and modify an existing image without having to re-enter all of the parameters associated with it. The user need only specify new values for those that are desired to be changed.

Given the speed with which existing images can be transferred from a disk to the graphics screen it was reasonable to provide a means of automatically sequencing through a collection of images in order to provide a form of animation. Since at most 17 images can be stored on one disk (and for other reasons) a "film" of at most 17 frames can be created. At a rate of 1.1 frames per second the film lasts only 15.5 seconds but the computer repeats the sequence of frames until the user tells it to stop. The imaginative user can use this facility to create interesting "film loops".

Some simple image management (i.e. file management) facilities are also provided for listing and changing the names of images stored on a disk and for deleting images from the disk. These duplicate some of the Apple Pascal operating system's file management facilities but they eliminate any need for the user to leave the QuadSurf program to perform such operations. Thus the user need not be familiar with the operating system in order to make good use of QuadSurf.

3. LIMITATIONS AND TRANSPORTABILITY

The QuadSurf package has several shortcomings that result from fixed design objectives or limitations of the computer system for which the package was designed. Two of these, the speed with which an image can be computed and the resolution of the graphics screen, have been mentioned earlier and will be discussed further. Three other shortcomings of QuadSurf are the small variety of surfaces that can be computed, the surface drawing algorithm is not particularly sophisticated (in contrast, see [2,3]), and no provision is made for drawing more than one surface in any one image. (Although the surface shown in Figure 2 has two disjoint parts it is a special case. What is meant here is that, for example, QuadSurf cannot draw two spheres in the same image.) Each of these shortcomings was forseen and accepted in order to maintain the feasibility of the project. A discussion of these issues is outside the scope of this paper. This section treats the issues of computational speed and screen resolution in terms of the potential for transporting QuadSurf to other graphics computers.

QuadSurf would be a much more useful package if it were available on a computer with a higher resolution graphics screen and with better facilities for floating-point arithmetic. Even though QuadSurf is written in Pascal, a fairly well-standardized language, transportability is still an issue for three reasons. First, UCSD Pascal, the flavor Apple supports, has a few nonstandard features which QuadSurf uses and so the program would require some localized modifications. However, this is the least of the problems. QuadSurf depends heavily on the particular way the Apple Pascal system does disk I/O and on the particular collection of machine-dependent graphics commands that QuadSurf employs to display images. These dependencies are too involved to discuss at length. It is enough to note that QuadSurf depends on disk format, disk directory format, the locations in memory of graphics display pages, and various other absolute addresses in memory that are part of the run-time environment of an Apple Pascal program. However, one of the design features of the package is that all of these dependencies are restricted to the user interface and file manipulation facilities provided by the package.

The QuadSurf package can be separated into two main parts of roughly equal size. The first of these consists of the outermost levels of the program and serves to create and maintain the user environment. This portion of the program would be difficult to transport because of the dependencies mentioned above. However, the second part is strictly responsible for computing images and itself does very little I/O and uses but a small collection of machine-specific graphics commands. (Indeed, this portion was first implemented on a Tektronix 4051, subsequently translated into Pascal because of the greater readability of that language and transported to the Apple.) This portion of the program is self-contained and could most easily be transported to another system. The implementor would need to create a new user environment to go along with the surface drawing algorithm but this environment could be as simple or complex as desired. Thus the computational and graphical limitations of this package can be overcome by transporting its core to a more powerful computer. The other issues remain as subjects of future efforts.

4. SUMMARY

The QuadSurf graphics package is a microcomputer-based facility for creating and maintaining collections of halftone 3-dimensional images and is designed for inexperienced and non-mathematically oriented users. The success of the package with respect to the last of these goals remains to be determined.

APPENDIX

The types of images QuadSurf can produce are now briefly cataloged and described. A more comprehensive treatment can be found in most any calculus text that contains a section on analytic geometry.

In general, QuadSurf can draw what are known as quadric surfaces. Such surfaces can be described by an equation of the form

$$Ax^2+By^2+Cz^2+Dxy+Exz+Fyz+Gx+Hy+Iz = J .$$

The choice of values for the coefficients A, B, C,..., J determines the shape of the surface that QuadSurf will draw. In the classification below at least six of the coefficients will be assumed to be zero. The user is left to experimentally discover the effects of nonzero values for those coefficients. The remaining coefficients in each case are assumed to be positive numbers.

1) Elliptic cylinder: $Ax^2+By^2 = J$.
   If $A = B$ then the surface has a circular cross section, otherwise it has an elliptical (oval) cross section.

2) Ellipsoid: $Ax^2+By^2+Cz^2 = J$.
   a) If $A = B = C$ this is a sphere of radius J (see Figure 1).
   b) If two of A, B and C are equal and the remaining coefficient is less than the other two the surface resembles a football.
   c) If two of A, B and C are equal and the remaining coefficient is greater than the other two the surface resembles the earth: a slightly "squashed" sphere.

3) Elliptic hyperboloid of one sheet:
   $$Ax^2+By^2-Cz^2 = J .$$
   An elliptical cylinder with a smoothly narrowed "waist".

4) Elliptic hyperboloid of two sheets:
   $$Ax^2-By^2-Cz^2 = J .$$
   This surface can be thought of as the surface of case (3) pinched so tightly that it separates into two pieces (see Figure 2).

5) Elliptic paraboloid: $Ax^2+By^2-Iz = J$.
   This is a surface resembling just one of the two pieces of the surface of case (4).

6) Elliptic cone: $Ax^2+By^2-Cz^2 = 0$.
   This is a surface resembling the surface of case (3) but pinched down just to the point of separating. It resembles two cones attached at their tips.

7) Hyperbolic paraboloid: $Ax^2-By^2-Iz = 0$
   This is a saddle-shaped surface extending infinitely in all dimensions.

Using the equation for an elliptic cylinder as an example, two observations can be made that also apply to the other surfaces described in this list. First, the values chosen for A, B, and J determine the cross-sectional area of the surface. For example, if $A = B = J = 1$, the cylinder has a radius of 1 unit. By increasing J or decreasing A or B the cross-sectional area of the surface will increase, and vice versa. Second, by exchanging the selection of coefficients from the general form of the equation the orientation of the surface can be altered. For example, an elliptic cylinder with equation

$$Ax^2+By^2 = J$$

will be drawn by QuadSurf as a cylinder standing on one end, that is, oriented along the z axis. The equation

$$By^2+Cz^2 = J$$

describes a similar surface but one lying on its side extending to either side of the viewing window and

$$Ax^2+Cz^2 = J$$

is an elliptical cylinder also lying on its side but extending through the plane of the window.

REFERENCES

[1] Seidel, S. R., Scanning on the fly: An approach to the user interface, Comput. Educ., (to appear).

[2] Potmesil, M., and Chakreverty, I., Synthetic image generation with a lens and aperture camera model, ACM Trans. on Graphics 1, 2 (April, 1982), 85-108.

[3] Blinn, J. F., A generalization of algebraic surface drawing, ACM Trans. on Graphics 1, 3 (July, 1982), 235-256.

DESIGN FOR AN ARTISTS' WORKSTATION FOR THE MICRO-COMPUTER

by David Cook

Digital Image Corporation
Grand Rapids, Michigan

ABSTRACT

In this paper, a design description for the software involved in an artist workstation is presented. This paper investigates the uses of scanners and functions incorporated into a highly defined user interface for the manipulation of graphic images. The results shows that complete graphics software systems can be developed for very low cost micro-computers which can rival, if not better, currently available and costly main-frame systems.

INTRODUCTION

In technology's onrush of computer graphics hardware and software we repeatedly hear that in order to have truly good, high quality graphics, we need massive number crunching computers to handle fast fourier transforms and other complex algorithmic strategies. While it is true that large machine running large programs can in turn produce very good graphics, the expense of such a system is much beyond most budgets. Three years ago, Digital Image Corporation, a new company, decided that for their first project they would attempt to produce high quality computer graphics for advertising, slide shows and television. Realizing that an equipment investment of $100,000 or more was required to buy a commercially available complete graphic system, it was also decided that the company would also design their own graphics system based around the Z80 eight bit microprocessor. Since time was a crucial factor it was decided that all the hardware should be current, off-the-shelf equipment, with the software being designed in-house.

COMPUTER EQUIPMENT

Cromemco Z2D, Z80 based computer system
  * 64K RAM - Main user memory
  * ITUART board (contains 2 parallel ports)
  * Two 5 1/4 inch floppy disk drives (386K each)
  * Beehive CRT
EPSON FX-80 graphics printer

GRAPHICS EQUIPMENT

Cromemco SDI graphics boards
  (16 colors out of map of 4096)
  * Two 48K RAM image planes
Via Video frame grabber with B&W RCA video camera
Houston Instruments HIPAD bitpad
Aydin Controls RGB monitor
3M color encoder (RGB to NTSC converter)
Matrix Camera (produces 35mm slides of RGB image)

After an initial six months of software development, the system was turned over to slide production with further software development occuring in non-production hours. At the time of this writing, we have just implemented version 4.0 of our software. One of the distinctions between our system and the larger comercially available 'canned' systems is that we had to produce a product using our own software. Having to eat our own code has made us painfully aware of our shortcomings in the initial design. This document will describe the basic problems encountered in designing a graphics system for an eight bit computer. We will deal in concept, inplementation examples will be in pseudo-code instead of Z80 assembler in order to make the ideas universal to all computers.

BASIC PROBLEMS

The basic problems we encountered in designing a graphics package for a eight bit processor were:

1) Resolution
   Would the product contain enough resolution to sell professionally in the television and 35mm slide markets?

2) Speed
   Would the hardware be fast enough to produce the required graphics in a time frame which was profitable?

3) Ease of use
   Would the system be useable without the operator needing two PhD's from MIT?

4) Versatility
   Would the system produce effectively without restricting the artists creative potential?

To solve these problems on a micro-computer required a tightly designed software system based around graphics theories which differ from the currently accepted standards.

THE SOFTWARE

The graphics software consists of several separate programs. FONT is designed to draw vectored stroke tables. This handles all lettering (fonts) and special images such as the 2D wireframe outline for a shape used to create a solid. SPECIAL PROGRAMS consists of several algorithms for mutating raster images such as tilting, skewing, block pixing, sphere generation etc. These are used, like FONT, to generate basic shapes or effects. SHOOT is a program which handles down loading an image to 35mm film. The final program, MENU, will be the topic of this paper. It is the hub of the system and is used to take all the basic components provided by the other software and modify them to create the final image. On the average, the user spends most of the time in MENU, therefore speed and efficiency are of prime concern.

GRAPHICS PRIMITIVE SET, BASIC DESIGN

Our computer included both Z80 ASSEMBLY LANGUAGE and FORTRAN IV. Speed and efficiency considerations made us choose assembly language for implementing the primitives. Since we wanted our primitive set useable in both languages , we endowed all primitives with FORTRAN call compatible parameter lists. Additionally, since our version of FORTRAN treats parameter lists containing more than three parameters differently than lists with three or less parameters, all calls contain no more than three parameters. Each parameter may be

either scaler or vector. Any routine requiring more than three parameters which can not be specified in a vector are usually specified through a second routine which makes the data available to the first routine through a global data area.

BASIC CONCEPTS

The graphics primitives are divided into the following four basic groups:

1) Environment maintenance primitives
2) Reality interface primitives
3) Image space manipulators
4) Color space manipulators

Group one, the environment maintenance functions, are simply commands to select work pages, display pages, stack usage and so on. Group one is highly dependent on the actual graphics hardware selected and therefore will not be discussed.

Environment Maintenance Primitives:

```
CPPAGE - Copy current page to alternate page.
DIFLIP - Display the alternate or flipped page.
DISP   - Display a particular graphics page.
DMAOFF - Turn graphics DMA off.
DMAON  - Turn graphics DMA on.
EXPAGE - Exchanges one page for another page.
FXDOT  - Write pixel via currently set function.
GRAFIX - Move FORTRAN stack for graphics use.
INIT   - Initilize general graphics environment.
NYBBLE - Set bit or nybble mapped resolution.
RES    - Set entire display to high or low
         resolution (high=756x482, low=378x241).
SCROFF - Turn the screen off.
SCRON  - Turn the screen on.
SETFUN - Set logical function for ISM commands.
SFUZZ  - Single neighborhood pixel average.
TARGET - Set target page and scanners ISM commands.
WINDOW - Open a window from page zero to page one.
WOFLIP - Work on the alternate or flipped page.
WORKON - Work on a particular graphics page.
XDOT   - Write pixel.
XREAD  - Read the color of any individual pixel.
```

Group two, reality interface functions, are composed of bitpad interfaces, frame grabbers and image-to-disk/disk-to-image routines. These fairly standard algorithms are also extremely hardware dependent. In a later part of this paper, the two major routines, BITPAD and BITGET, will be discussed.

Reality Interface Primitives:

```
BITGET - Get decoded data from the bitpad.
BITPAD - Get non-decoded data from the bitpad.
CURSOR - Display or inhibit display of the cursor.
FRAME1 - Grab a small frame from the video camera.
FRAME2 - Grab a big frame from the video camera.
FRAME3 - Grab a tilted frame from the video camera.
ITUART - Set parallel port for bitpad operation.
RESET  - Reset the parallel port for redefinition.
```

The final two groups image and color space manipulators, will constitute our main focus as these groups contain the most useable concepts.

IMAGE SPACE MANIPULATORS (ISM)

Each of the two image planes may be divided into any number of sub-image areas. These areas are called boxes and are defined by entering the lower left and upper right coordinates of the desired area. All of the ISM primitives require a minimum of the following information.

1) Source box coordinates
   The area containing the image to be modified by the ISM function.

2) Destination box coordinates
   The area which will receive the results of the ISM function.

3) Color vector
   The color vector has different meaning to different ISM primitives based on their individual functions. However, it usually specifies which colors in the source box are to be modified.

4) Mathematical/Boolean functions
   How the source box will integrate with the destination box. Choices are: REPLACE, ADD, SUBTRACT, XOR, OR, and AND. The function in effect at the time of the ISM primitive call will cause the result of the ISM to be ADDed, SUBTRACTed, XORed etc. into the destination area. Function is set by the routine SETFUN.

5) Box scanners
   Modifies how the source box and destination box are scanned. Currently we have eight separate scanners. Source and destination scanners need not be the same. This allows the user to mirror or rotate a raster image 90, 180, 270 and 360 degrees.

6) Source and destination pages
   Specifies that the source and destination boxes exist on the same or separate pages. This allows all ISM primitives to take an image on any page, modify it, and place the result (via the function and scanners) to any page. Both scanners and pages are specified by the routine TARGET.

The first three data elements are included in the ISM primitive calling parameter list in the format:

```
CALL ism (source box,destination box,color vector)
```

The remaining three elements must be specified through their corresponding routine before calling the ISM primitive.

ISM PSEUDO-DESIGN

Since most all of the ISM primitives are similar in design and differ only in the functions they perform, we can express them in a pseudo-language shell. The various functions can be placed interchangably in the shell below. Here then is the basic program structure for most of the ISM primitives:

```
ISM (source_box,destination_box,color_vector)
BEGIN
 SAVE color_vector_address
 SAVE destination_box_address
 CONVERT source_box TO memory_coordinates
 RETRIEVE destination_box_address
 CONVERT destination_box TO memory_coordinates
 RETRIEVE color_vector_address
 GET source_scanner_choice FROM global_data_area
 CONVERT source_scanner_choice TO
        scanner_calling_address
 GET destination_scanner_choice FROM
     global_data_area
 CONVERT destination_scanner_choice TO
        scanner_calling_address
 DO UNTIL NO MORE SOURCE SCAN POSITIONS
  SET display TO source_page
  READ source_color FROM
       current_source_scanner_position
  CALL source_scanner
       (move to next position in the source box)
  SET display TO destination_page
                    .
                    .
   particular ISM primitive function
        (see descriptions below)
                    .
                    .
  CALL destination_scanner
       (move to next position in destination box)
 END DO
END
```

As mentioned previously, the source_page / destination_page and source_scanner / destination_scanner are set prior to calling the ISM primitive through other primitives and are loaded globally by the ISM itself.

ISM PRIMITIVE DESCRIPTIONS

The ISM descriptions presented below contain the replacement pseudo-code for the ISM shell, a mathematical description and an English description. The following key contains symbols used to describe ISM primitive functions:

RP = Resulting Pixel: The resulting value from the ISM function for any given pixel within the scan area.

SP = Source Pixel: The current source scan position pixel value.

DP = Destination Pixel: The current destination scan position pixel value before execution.

[f] = Function: The currently set function, one of the following: REPLACE, XOR, AND, OR, SUBTRACT, ADD.

Here then are the ISM primitive descriptions:

COPY: COPY source box to destination box

```
FUNCTION:  RP = (DP [f] SP)

PSEUDO:
 IF source_pixel_color IS FOUND IN color_vector
  THEN PLOT source_pixel TO destination_box VIA
       function AND scanners.
 ENDIF
```

COPY takes each pixel in the source box and checks to see if the color is set in the color_vector. If not, no action is taken for that pixel. If the color is found, the pixel is written to the destination area via the function, with placement determined by the scanners.

AVE: Average source box with destination box

```
FUNCTION:  RP = (DP [f] ((SP + DP) / 2))

PSEUDO:
 IF source_pixel_color IS FOUND IN color_vector
  THEN ADD destination_pixel TO source_pixel
  DIVIDE result BY TWO
  PLOT result TO destination_box VIA function
       AND scanners
 ENDIF
```

AVERAGE will take each pixel in the source box which also exists in the color vector and calculate a mix, or average, between itself and the corresponding pixel in the destination box. The result of this calculation is written into the destination box via the function with placement determined by the scanners.

FADE: Slow average source to destination box

```
FUNCTION: IF SP = DP THEN RP = (DP [f] DP)
          IF SP > DP THEN RP = (DP [f] (DP + 1))
          IF SP < DP THEN RP = (DP [f] (DP - 1))

PSEUDO:
 IF source_pixel_color IS FOUND IN color_vector
  THEN IF source_pixel IS GREATER THAN
       destination_pixel
   THEN ADD ONE TO destination_pixel
  ELSE
   IF source_pixel IS LESS THAN destination_pixel
    THEN SUBTRACT ONE FROM destination_pixel
  ENDIF
  PLOT destination_pixel via function and scanners
 ENDIF
```

FADE is basically a slow AVERAGE. FADE takes each pixel in the source box and checks to see if it is set in the color_vector. If the color is found, FADE makes the corresponding pixel in the destination_box ONE MORE LIKE the source pixel. This allows two images to be faded from one to another in 15 steps.

CEQC: Color EQuals Color

```
FUNCTION: RP = (SP [f] value)

PSEUDO:
 IF source_pixel_color IS FOUND IN color_vector
  THEN PLOT source_pixel MODIFIED BY value VIA
       function TO destination_box
 ENDIF
```

CEQC is a raster oriented IF statement which allows the user to modify only particular colors in the display via a value predefined by the routine CEQCR (value). For example, the user may wish to turn all odd colors to even colors. To do this the user need only set the odd colors ON in the color_vector, set the function to ADD and the value to one. This will add the value one to all odd colors in the source rastor area and plot the results in the destination raster area. If the function were set to REPLACE, all odd colors would be replaced by the color one.

The following ISM primitives differ from the last four in that they perform more complex functions and may alter the format of the ISM primitive shell.

FUZZ: Pixel average (smoothing)

```
FUNCTION: RP = SP [f] ((SP[1,1]+SP[1,0]+SP[1,-1]+
               SP[0,-1]+SP[-1,-1]+SP[-1,0]+
               SP[-1,1]+SP[0,1])/8)

PSEUDO:
 IF source_pixel_color IS FOUND IN color_vector
  THEN SET destination_pixel EQUAL TO ZERO
  DO 8 TIMES
   MOVE TO next_neighboring_pixel
   SET destination_pixel EQUAL TO destination_pixel
       PLUS neighboring_pixel_color
  END DO
  SET destination_pixel EQUAL TO destination_pixel
      DIVIDED BY EIGHT
  PLOT destination_pixel VIA function AND scanners
 ENDIF
```

FUZZ allows the user to exercise several image enhancement algorithms such as smoothing, edge-detection and color wraparound on the source_box. Fuzzing is one of our most important effect generators and is responsible for improving resolution through smoothing. In using functions with FUZZ, glows, neons, basic textures and image cleanup may be performed. Basically, FUZZ extends an nine pixel neighborhood through the image area. As the neighborhood moves through each pixel in the scan area, the eight pixels surrounding the center pixel are averaged, and the result is placed over the ninth or center pixel. If the result is SUBTRACTED from the center pixel, the resulting image will be edge-detected. Other functions perform other useful modifications. For more information on implementing FUZZ on a micro-computer, please consult the JANUARY '83 issue of Creative Computing.

KEY: Pattern fill, complex copies

```
FUNCTION: RP = (DP [f] SP)

PSEUDO: (Note partial shell defination included)
 KEY (source_box,destination_box,color_vector)
  BEGIN
   SAVE color_vector_address
                .
                .
    Same as general ISM shell
                .
                .
   CONVERT destination_scanner_choice TO
           scanner_calling_address
   DO UNTIL NO MORE SCAN POSITIONS IN
     destination_box
    SET display TO source_page
    READ source_color FROM
        current_source_scanner_position
    CALL source_scanner (move to next
         position in the source box)
    SET display TO destination_page
    READ destination_color FROM
         current_destination_scanner_position
    IF destination_pixel_color IS FOUND IN
       color_vector
     THEN PLOT source_pixel TO destination_box
          VIA function
    ENDIF
    CALL destination_scanner (next position
         in the destination box)
    IF destination_scan AT START OF NEXT SCAN LINE
     THEN SET source_scan TO NEXT SCAN LINE
    ELSE
     IF source scan AT END OF CURRENT SCAN LINE
```

```
      THEN SET source_scan TO START OF CURRENT
           SCAN LINE
    ENDIF
   ENDIF
  END DO
 END
```

KEY, while looking like COPY, is actually COPY in reverse. While COPY allows the user to copy the color_vector colors from the source_box to the destination_box, KEY will copy ALL the colors from the source_box only ONTO the color_vector colors in the destination box. Thus, where in COPY, the user can specify which colors are to be copied, in KEY, the user specifies onto which specific colors the key is to occur. Furthermore, if the source_box is a different size than the destination_box, KEY will repeat the source_box within the destination_box until it is completely filled. If different scanners are specified for source and destination boxes, KEY will rotate the source area before plotting it into the destination area. This allows KEY to rotate any n by m image into a n1 by m1 area without smearing. The other ISM primitives require that the source_box dimensions be similar to the destination_box dimensions (ie source = n by m, destination may be n by m or m by n). If this is not the case, the source box is smeared into the destination area. KEY overcomes this effect by rotating each source scan line before plotting enabling it to take a n by m box, rotate it and plot it into a n1 by m1 area without smearing.

3D: Intensity as a function of space

FUNCTION: RP = DP [f] SP where RP[x,y] =
DP[((SP*xm)+x),((SP*ym)+y)]

```
PSEUDO:
 IF source_pixel_color IS FOUND IN color_vector
  THEN MULTIPLY source_pixel_color BY x_multiplier
  ADD destination_pixel_x_coordinate TO RESULT
  MULTIPLY source_pixel_color BY y_multiplier
  ADD destination_pixel_y_coordinate TO RESULT
  PLOT source_pixel_color TO destination_box VIA
       scanners AND function AND translated
       coordinates
 ENDIF
```

3D takes the desired pixels in the source box and adds their respective intensities to the x and y coordinates of the corresponding pixel in the destination_box. The user is allowed to specify a multiplier for both the x and y axes prior to the calling of 3D by calling PUT3D (xm,ym). If either multiplier is zero that axes is turned off and extrapolation occurs for that axes as if copy were called. Figure 1 shows a before and after image of a tiled surface consisting of two colors, one and 15. The after image shows that the color one tiles are shifted one pixel in the x and y (xm,ym=1) while color 15 tiles are shifted 15 pixels in the x and y. If a xm,ym of two were used, color one tiles would be shifted two pixels and color 15 tiles would shift 30 pixels. This type of effect has many uses. First, it allows a 2D image to be extrapolated into a 3D image by using color as depth information. Secondly, since this extrapolation is stronger for the higher colors shifting them further away from dimmer colors, it will leave shadows from the shifted position. The shadows will fall as predicted by where the hilights fall in the image. In fact, if more than one lighting source is hitting the image, multiple shadows can result. While this method is not nearly as controllable or accurate as raytracing, it provides a small, slow processor with a very high speed method of obtaining a similar result. It should be noted that the internal workings of 3D ALWAYS ADD the color to the coordinates. This means that in order to obtain proper treatment of the image, the user should specify both source and destination scanners be left to right-top to bottom, otherwise overprinting will occur. If the scanners are reversed the image will be plotted away from the viewer as opposed to towards the viewer. It should also be noted that it is undesirable to 3D map with color zero as this will often delete part of the resulting image (color zero will not shift in the x and y as it is a value of zero).

Finally, several other ISM primitives exist which allow the user to modify only a source area. These primitives do not use scanners or functions in the same way as the previous ISMs do.

SHIFT:

SHIFT is identical to COPY in that it will copy a source box to a destination area (only on the same page). It differs in that it replaces each source pixel with the background color after the shift. Shifts can occur in any direction at any jump factor but the user must specify the proper scanners (to avoid overwriting) before calling SHIFT.

FIND:

FIND is a method of locating a desired rastor image on the screen. FIND is called with the desired coordinates of a source_box and will attempt to find a non-background colored image within that coordinate space. If no image is found, the coordinates are returned unchanged. If an image is found within the source area, the coordinates are changed to fit the image on all four sides. This routine does not use scanners and perforas the search two pixels at a time making it extremely fast. The worst case find for locating a single pixel within the entire screen area occurs within 2 seconds. This is our fifth find algorithm and represents a speed improvement over the last algorithm of over 80 percent.

XORBOX:

XORBOX will take a source_box and XOR it with the value 15. This routine does not use scanners and is extremely high speed. This may be used for showing the user which boxes are currently in effect.

GETBOX/PUTBOX:

GETBOX and PUTBOX allow the user to store up to 96 separate box coordinates in a small unused portion of the image plane. Since our system consists of two image planes, a total of 192 box coordinates can be stored and retrieved. This allows programs to easily communicate box information back and forth.

COLOR SPACE MANIPULATORS (CSM)

Four basic CSM routines exist for manipulating colors. These routines are the basis for several higher level routines which perform shading and hueing. All data is supplied to the routines as one byte integers. Colors are specified as red, green and blue gun values where each value is a single byte with a value between zero and 15. Colormap positions are also single bytes with a value between zero and 15. Therefore, a grey scale would look like:

Color	R	G	B
0	0	0	0
1	1	1	1
2	2	2	2
.	.	.	.
.	.	.	.
14	14	14	14
15	15	15	15

While a map shaded from red to yellow with a blue background may appear as:

Color	R	G	B	
0	0	0	15	(Background)
1	15	0	0	(Red)
2	15	1	0	
3	15	2	0	
4	15	3	0	
5	15	4	0	
6	15	5	0	
7	15	6	0	
8	15	7	0	
9	15	8	0	
10	15	9	0	
11	15	10	0	
12	15	11	0	
13	15	12	0	
14	15	13	0	
15	15	15	0	(Yellow)

While the following four low-order primitives described below are extremely simple routines, they are the foundation for much more complex color manipulation routines as described further in this paper.

DEFCLR: DEFine CoLoR

```
FUNCTION:  cm(map+0) = RED        (cm = colormap)
           cm(map+1) = GREEN
           cm(map+2) = BLUE

PSEUDO:
 DEFCLR (map,rgb_stack)
  BEGIN
   DO WHILE VERTICAL_BLANKING
    SET color_map INDEXED BY map PLUS ZERO TO
        rgb_stack(0)
    SET color_map INDEXED BY map PLUS ONE TO
        rgb_stack(1)
    SET color_map INDEXED BY map PLUS TWO TO
        rgb_stack(2)
   END DO
  END
```

DEFCLR allows the user to set an individual color map position to a red, green and blue setting. 'Rgb_stack' must be a three byte stack containing the desired RGB values (each between zero and 15); 'Map' must be a single byte value between zero and 15 indicating the desired color map position to modify.

SETMAP: SET the entire color MAP

```
FUNCTION: cm(0-15,1) = RED(0-15)
          cm(0-15,2) = GREEN(0-15)
          cm(0-15,3) = BLUE(0-15)

PSEUDO:
 SETMAP (map_stack)
  BEGIN
   DO LOOP=16 TIMES
    DO WHILE VERTICAL_BLANKING
     SET color_map INDEXED BY LOOP PLUS ZERO TO
         map_stack(LOOP,0)
     SET color_map INDEXED BY LOOP PLUS ONE TO
         map_stack(LOOP,1)
     SET color_map INDEXED BY LOOP PLUS TWO TO
         map_stack(LOOP,2)
    END DO
   END DO
  END
```

SETMAP allows the user to immediately set an entire color map; 'map_stack' must be a 16 by 3 byte array containing the R,G and B values for each map position (0-15). This routine is extremely fast as it sets all 16 colors within one vertical blanking interval making it ideal for higher order color map animation routines.

GETMAP/PUTMAP:

GETMAP and PUTMAP allow the user to store up to 96 separate color maps in a small unused portion of the image plane. As with GETBOX/PUTBOX, if both image planes are used, a total of 192 maps may be stored. Map storage in no way affects box storage or image display. This is the only way for programs to communicate the color maps and box coordinates to each other.

MENU SOFTWARE

The power of the primitives can be seen by the fact that each ISM primitive can handle many separate functions. Just on functions alone, each ISM routine is capable of six different effects, one for each of the six possible boolean functions. When combined with source page/destination page and source scanner/destination scanner, the number of possible effects is greatly increased. However, in order to provide the user a sane way of using the primitives, an overall gridwork had to be laid over the primitive system. This gridwork consists of an extremely powerful user interface which couples the artist to the primitives allowing complete freedom to access all of the primitives powers. Menu, therefore, is the only program to use all the primitives.

Physically, menu is composed of three major components: the CRT screen which contains a continuously running environment monitor informing the user to the currently set parameters, the bitpad surface which contains the menu pad, and the RGB monitor which displays the complete image.

THE BITPAD

The bitpad surface is physically composed of three components: the digitizing surface, the button area, and the stylus. The digitizing surface is the area in which the user points and draws with the stylus. The button area is on the left of the bitpad and consists of four buttons which control the stylus. The first button is the RESET button and resets bitpad operations. The other three buttons are explained later.

The bitpad is controlled by two primitive functions; BITPAD and BITGET. A call to BITPAD will return the current x and y coordinate addresses of the stylus as two values between one and 10,700 each. Along with the x and y position, the bitpad returns the status of the stylus indicating if the stylus tip is touching the bitpad surface as well as the condition of three switches mounted in the bitpad surface.

BITGET, in most cases, is used instead of BITPAD as it performs several different functions besides calling BITPAD for the user. BITGET allows the user to treat the BITPAD surface in two separate ways. If the user is in STREAM mode (stream button on) BITGET will track the stylus with the cursor. If the stylus is depressed, the cursor will be turned off and the x and y coordinates will be returned. These coordinates will be extrapolated to fit screen coordinates (1-378,1-241). If the user is in POINT or SWITCHED STREAM mode, BITGET will not display the cursor and will treat the bitpad surface as if it contained 441 button areas divided into 21 rows by 21 columns. Depressing the stylus into one of these areas will result in their coordinates (1-21,1-21) to be returned. The difference between POINT and SWITCHED STREAM is that in POINT the stylus must be depressed for each button desired, while in SWITCHED STREAM, as long as the stylus is depressed, the button coordinates are fed continuously to the host. BITGET performs one further operation. A separate routine, BITAVE, can be called to toggle an averaging bit in BITGET. If the bit is off (.FALSE.) then BITGET behaves as mentioned above. If the average bit is set (.TRUE.) then each point in STREAM mode is averaged with the most recent point and any new points which are further than 15 pixels away from the most recent point are discarded. This allows the bitpad to be useful even in a noisy electromagnetic environment such as those often present around poorly designed RGB monitors.

MENU STRUCTURE

As mentioned earlier, MENU can be thought of as a controlling gridwork, or interface, between the user and the primitives. At the heart of this gridwork is a 441 word (882 byte) table divided into 21 rows by 21 columns. Each word is an address pointer to the routine which will handle the corresponding bitpad button (via BITGET). If the pointer is zero, no routine exists for that button. Otherwise, the address is retrieved and jumped to. This direct approach to the jump table allowed us to easily layout the bitpad buttons until a comfortable position was found for each command. This also allows more than one button to reference the same command making the layout more versatile by limiting the amount of operator arm travel around the bitpad surface.

MENU divides the bitpad buttons into several categories (figure 2a and 2b):

1) Keypad commands
2) Color commands
3) Box commands
4) Pen commands
5) Control functions
6) Other functions

The CRT is controlled by MENU and contains several areas. Important information in each area is displayed as full intensity, or 'bright'. Any unimportant information is always displayed at half intensity, or 'dim'.

The five areas provided by MENU on the CRT screen are (figure 3):

1) Keypad vector
2) Color vector
3) Page zero vector
4) Page one vector
5) Control indicators

The last line on the CRT is reserved for a computer-to-user message center. Errors, warnings and notices are placed here by MENU for the user. In designing the current version of menu we decided not to give the user audible feedback of button hits. In older versions, the CRT 'beeped' everytime the user hit a valid key. However, after eight solid hours of use, most operators went beep crazy. MENU now only beeps when it wants the user to look at the message line. To further enhance the operator-computer interface, MENU requires the user to type into the keyboard only for filenames. This keeps the operator positioned at the bitpad for most of the session, instead of bouncing back and forth from keyboard to bitpad.

KEYPAD COMMANDS - THE KEYPAD VECTOR

The keypad section is divided into several sections:

1) Numeric keypad
2) Sequence keys
3) Control keys

These keys are clustered in the center of the bitpad surface for accessability as these keys are the most frequently used. They function in two ways. First, by allowing the user to specify how commands are to be executed. Secondly, by allowing the user to use the keypad vector area as a temporary storage stack for complex operations. However, to the computer this area is meaningless until applied via one of the vector key operators discussed below.

NUMERIC KEYS

The numeric keys include the numbers '0' thru '15' and one non-numeric key- 'FF' for Full Frame. Depressing any of these keys once causes the corresponding legend under the keypad area on the CRT to be bright. Depressing the key again will cause it to toggle dim. This allows the user to specify numeric keys quickly as well as perform corrections quickly. The FF key will toggle the same way with the corresponding FF legend in the keypad vector. To the computer, this key correspondes to the value 16, to the user, it means the entire screen image, or full frame. How these keys are used will become apparent further in the paper.

SEQUENCE KEYS

The sequence keys consist of '1-15', EVEN, ODD and '-' (THRU). These keys allow a fast setup of certain sequences. '1-15' will automatically set the numbers 1 2 3...15 bright under the keypad vector. If one of the numbers is already bright, it will be inverted to dim. Thus, if 5 thru 8 is already bright when '1-15' is issued, 5 thru 8 will toggle dim and 1-4 and 9-15 will become bright. The EVEN key will react the same way, inverting only the even number keys (0 2 4 6...14). The ODD will invert only 1 3 5...15. The '-' or THRU key will allow the user to input a consecutive sequence. When touching THRU, the computer will prompt the user to 'ENTER FIRST NUMBER' in the CRT message area. The user then touches the button corresponding to the desired value. After this first value is entered, the message 'ENTER SECOND NUMBER' is displayed. The user then enters a second value after which the areas between and including the two values are inverted.

CONTROL KEYS

The control keys consist of 'CE' (CLEAR ENTRY), 'SKIP' and 'QUIT'. The 'CE' key will set all legends under the keypad vector to dim (off). This effectively erases any current entries in this vector. To understand 'SKIP' and 'QUIT' it must be first understood that besides the keypad vector, some commands also use the numeric area to input values for other operations. These include setting scaling factors, jump values and colors. The 'SKIP' key allows the user to skip entering a particular value. The 'QUIT' key allows the user to abort any operation which is currently requesting information via the bitpad. Neither the 'SKIP' or 'QUIT' keys affect the keypad vector.

COLOR KEYS - THE COLOR VECTOR

One of the easiest ways to apply the keypad vector is to use it to manipulate colors. However, before any color may be modified the user must instruct the computer WHICH colors are to be modified. To do this, the user sets the keypad vector so that only the desired colors are bright. To apply the keypad vector to the color vector, the user need only press the 'C' key. This will brighten all the R,G and B colors which correspond to the bright keypad vector numbers. Only bright colors are modified. Two other keys aid in manipulating the keypad vectors and color vectors, the '<C' key will load the keypad vector from the color vector. In other words, what is dim in the color vector becomes dim in the keypad vector and what is bright in the color vector becomes bright in the keypad vector. No inversion of the keypad vector takes place in this instance. The '<C>' key will exchange the contents of the keypad vector with the color vector. This allows the user to keep secondary color selections in the keypad vector and swap them back and forth with the contents of the color vector. Two other keys, 'C+' and 'C-', will rotate the color vector up or down allowing the user to manipulate the color vector without going through the keypad vector for every operation. Note that only what is bright and dim rotates, the actual colors do not rotate nor do their settings.

To the right of the keypad area are the color keys. These keys actually modify the color of a given area in some way. All color key commands use the SETMAP and DEFCLR primitives to perform their individual functions. The following is a description of each key in the color area:

'GREY RGB','GREY R','GREY G','GREY B':

These keys do not require the color vector to contain bright areas. They are built to be single key commands with no setup required. The 'GREY RGB' key will set the entire color map black and white. 'GREY R' only sets the red gun to a linear scale between zero and 15 (black and white on red gun only). 'GREY G' and 'GREY B' act as 'GREY R' on their corresponding guns.

'+ RGB', '+ R', '+ G', '+ B':

The '+ RGB' key will increment the R, G and B colors for all bright colors in the color vector. Dim area remain unchanged. For any given color, if it increments past 15, it is automatically wrapped around to zero. '+ R', '+ G' and '+ B' only increment their respective color guns. Therefore, if color five had a RGB of 5, 7 and 15 and a '+ RGB' followed by a '+ G' was issued, the result would be a RGB of 6, 9 and 0.

'- RGB', '- R', '- G', '- B':

These keys act similarly to the above PLUS keys with the exception that they decrement instead of increment. If a color is decremented past zero, it is wrapped around to 15. Therefore, if color two has a RGB of 0, 5 and 15 and a '- RGB' is issued, the result would be a RGB if 15, 4 and 14.

'? RGB', '? R', '? G', '? B':

These keys will randomly color all RGB guns or any individual gun affecting only those colors that are bright in the color vector. The randomly picked

values are between zero and 15. Dim areas are not affected.

'ROLL DOWN', 'ROLL UP':

These keys will rotate all bright colors in the color vector. The action is to physically rotate the selected colors through the color map. Only bright colors are rotated. Therefore, if only one color is selected, no action occurs. If two colors are selected, they are swapped. If three or more colors are selected, they are rotated in the desired direction. This command has the ability for colormap animation if images are designed with color rotation in mind.

'ROLL RIGHT', 'ROLL LEFT':

The RIGHT and LEFT ROLL will rotate all bright colors in the color vector left or right. In other words, if rotating right, red is moved to green, green to blue and blue to red.

SET RGB:

The SET RGB key allows the user to specify exact R,G and B values for each bright color. After pressing SET RGB, the computer prompts the user to enter the RED value for the first bright color. The user then presses the desired value on the keypad (0-15) which is followed by the prompt for GREEN and BLUE. After all three components are entered, the color is changed and the R,G and B for the next bright color is requested. If the SKIP key is pressed when requesting a value, that component (R,G or B) is skipped and the next component is requested. If QUIT is hit, SET RGB is terminated.

LITE:

The LITE key allows the user to determine what colors exist in the image currently on the screen. Upon touching LITE, the entire colormap is turned to black. Then each color is displayed, one at a time in white, with a slight pause between each one. Along with this, the terminal beeps with each color enabling the user to count the colors as they are displayed. To further aid the user, the bottom line of the terminal contains the current color being shown. After all bright colors are shown, the original color map is assumed.

LITE ALL:

The LITE ALL key is similiar to the LITE key with the exception that it does not sequence through the colormap, but rather shows all bright colors as WHITE at the same time while turning off those colors that are dim in the color vector. This allows the user to see which part of an image contain a combination of colors. For example, if the user has as bright colors 3,5 and 9 and requests LITE ALL, all colors except 3,5 and 9 are made black then 3,5 and 9 are made white and are displayed this way for about three seconds before restoring the display to the original color map.

1/2:

The 1/2 command allows the user to use one color map or two color maps. Since the user has two image pages at their disposal, the user may specify that one color map is for both pages, or that each page contains its own color map. If one map is chosen, the same map is applied to both pages. If two maps are chosen, the maps are changed every time the page is changed. Along with two maps, the system also keeps two color vectors indicating which colors are bright in both maps. These independent color vectors are only in use when two maps are chosen. To inform the user which mode is currently in effect the computer will brighten the arrows ('<---' and '--->'), which appear to the left and right of the color vector legend, as to which page the current map is being applied to. If both arrows are bright, the map is applied to both pages (one map). If only one arrow is bright, then two maps are in effect and the current map is pointing to the current page.

'HUE', '1 HUE ?', '1 HUE 15':

The HUE key allows the user to interpolate color values between two previously set values. If colors one and 15 are bright and all other colors are dim, and the user hits HUE, colors two through 14 will become a linear interpolation between colors one and 15. In other words, if color 1 is RED and color 15 is YELLOW and the user hits HUE (with 1 and 15 bright and all other dim) then colors two through 14 will become colored from RED to YELLOW in linear steps. If more than two colors are bright, the computer will perform a linear interpolation between the first and second, second and third etc., to the last. '1 HUE ?' will assume that color one is bright whether it is or not. This allows the user to specify a color such as 15 to be bright, random color it and shade to color one without having modified color one or having to redefine the color vector. '1 HUE 15' acts as '1 HUE ?' except one and 15 are assumed bright. This allows the user to 'keep a theme' in color shading. If color one is set to RED and color 15 is set to YELLOW and color eight is bright with all other colors being dim, the user may then random color only eight and shade from one to eight and eight to 15. This would keep a red to yellow theme while throwing in another color to augment the combination.

OTHER COLOR KEYS:

In order to aid the speed by which one can setup the color vector, two keys were included as presets and map control. The '0 C' key which is positioned in the upper left hand corner of the keypad vector will automatically set only color zero bright and all other colors dim. The '1-15 C' will automatically set only 1-15 bright and zero dim.

BOX KEYS - THE PAGE 0 AND PAGE 1 VECTOR

To request the computer to modify an image in ways other than color, the user must apply the KEYPAD VECTOR to either the PAGE 0 or PAGE 1 vectors. The page zero vector is to the left of the color map, the page one vector to the right of the color map. An individual box in either vector is selected by first going to the desired image page (via the 'P' or 'PSA' keys), selecting the desired boxes to work on in the KEYPAD VECTOR as described above in COLOR and hitting the 'S' key. This will SELECT the desired box for work. If, for example, the user depresses the zero and one keys brightening their values in the keypad legend, and then hits the 'S' key, the user will observe the symbol '<' for page zero, or a '>' for page one appear next to the desired box. (Note that the symbol POINTS TOWARDS the desired box.) This means that the box has been selected for work. As with the color keys, the '<S' key will load the keypad vector with the contents of the current page vector. The '<S>' key will swap the contents of the keypad vector with the contents of the current page vector. Once a box has been selected, it may be grabbed, found and manipulated. Since the ISM primitives allow a source and destination box, the 'S' keys define the source. To define the destination box area, the user has several choices. If a box is simply selected ('S'), then it is both source and destination. If, the destination is desired to be on the alternate page, the user may hit the 'A' key. This key will set an arrow ('<' or '>', depending on which page they are on) POINTING AWAY from the current page vector. This indicates that a box is selected (the select arrow) and is pointing to the other page (the alternate arrow). Along with the A key is the '<A' and '<A>' key which act just as <S and <S>. If it is desirable for a box to point to another box on the SAME PAGE, then the FORCE ('F') keys may be used. 'F' allows the user to input a value from the keypad (number between zero and 15) which will be entered between the '<' and '>' positions on the current page vector. Along with the F key is the 'ALL F' key which will set ALL selected boxes to the same force value. The 'ZAP F' key will remove ALL force settings from ALL boxes regardless of selection. Therefore, the following setup is possible:

```
                Page 0
0) 20  40  100  241 ( 81x 202) <
1) 55 100  120  150 ( 66x  51) < 5
2) 35 110  140  200 (106x  91) <  >
3)
4) 78  40  200  240 (123x 201) <FF>
5) 35 110  140  200 (106x  91)
```

In the above example, box zero contains x,y,x1,y1 coordinates of 20,40 by 100,241 and is a box sized 81 by 202. It is selected and the destination box is itself. Box one is selected and points to box five on the same page. Box two is selected and points to box two on the alternate page, box three does not exist (no coordinates) and box four is selected and points to the FULL FRAME on the alternate page. Box five exists but has no select arrow and therefore is not active.

'S+', 'S-', 'A+', and 'A-' all act the same way as the 'C+' and 'C-' keys by rotating the select and alternate arrows up and down. The 'F+' and 'F-' keys react similarly by rotating the FORCE values up and down.

Two keys at the lower right hand portion of the bitpad allow the user to save even more setup time. The 'ZERO S' key will automatically remove all select arrows and then select only box zero. The 'LEGAL S' key will remove all select arrows and then only select boxes with a legal coordinate displayed. LEGAL S will not select full frame. Both keys only effect the current page.

One final key, 'PSA', allows the user to instantly duplicate select and alternate arrows from one page to another when paging. Similar to the P key, which will be explained later, PSA will go the the alternate page and will then duplicate all select and alternate arrows till both pages are identical. Coordinates are not changed, nor are force values. This is useful when the same commands are to be executed from page to page. On the first page, the user can save setup time by PSA and from then on, P.

BOX DEFINITION COMMANDS:

Immediately below the keypad are the box definition commands. The first of these commands 'BOX GRAB' will prompt the user to enter the lower left and upper right coordinates for each SELECTED box, one at a time. The user may enter either a screen coordinate by placing the bitpad into STREAM mode and using the stylus to indicate the coordinate, a number from the keypad indicating the coordinates from an already existing box, or a combination of the two. If for example, the user is defining the location of box zero, and wants the lower left to be somewhere from the screen but wants the upper right to be the same as box one, the user may hit (after BOX GRAB) the STREAM mode button and then point the stylus to the desired lower left position and then hit the POINT mode button followed by the '1' key. This will assign the screen coordinate to the lower left of box zero and the upper right from box one to the upper right of box zero. If 'CE' (Clear Entry) is entered for a box coordinate, its coordinates are removed from the current page vector. If the 'SKIP' key is depressed, that particular coordinate is skipped in definition. If 'QUIT' is depressed, BOX GRAB is terminated. Once a box has been defined, the computer will display the coordinates next to the select arrow (to the left if page 0 and to the right if page 1). At this time, the boxes dimensions will also be calculated and displayed to the right of the coordinates. The dimensions are computed by the display processor as opposed to the GRAB routine.

To the right of BOX GRAB may be found the 'AUTO GRAB' button. This is useful when working with many boxes. For example, if the user wishes to center many lines of text, they need to count the lines, select the proper number of boxes and BOX GRAB them. Instead, however, they may hit the 'AUTO GRAB' key. AUTO GRAB first erases ALL box coordinates for the current page. AUTO GRAB also erases ALL select arrows for the current page. Alternate arrows and forces are not affected. AUTO GRAB then begins asking for coordinates at box zero. After each set of coordinates are entered, the coordinates are placed on the screen and the next box is requested. The 'QUIT' key is the ONLY way to terminate this command if less than 16 boxes are desired. After all 16 boxes are entered, or the QUIT key is used, the computer will place SELECT arrows next to each legal coordinate. In this way, the user is not required to count and select the boxes, just grab them.

At times, it becomes useful to duplicate box coordinates in other box positions. Uses for this include positioning a box in the same position as another box on another page, creating a box in another location on the same or different page which is the same size as another box etc. To the right of AUTO GRAB are three keys which aid in box coordinate duplication. The 'S-->D' key will force the source coordinates to the destination box. The 'D-->S' key will force the destination coordinates to the source box. The 'DESTINATION SQUARE' key will force the upper right coordinates of the destination box to be the same distance from the lower left of the destination box as the upper right of the source box is from the lower left of the source box. For example:

```
PAGE 0
55  78 100 210 ( 45x 132) <  >
```

If the user hits 'S-->D', then box zero on page one will contain the same coordinate as the above box. If the user has forced the value five, then box five on page one will contain the same coordinate as the above box. If the user hits DESTINATION SQUARE, then the destination box (zero or five depending on the above example) will have its upper right coordinate moved till it is the same distance from the lower left as the source box. This will make both the source and destination box dimensions the same while allowing them to be in different places.

To the left of BOX GRAB is the 'FIND' key. This command uses the FIND primitive and allows the user to find the image within each selected box on the current page. In other words, each box coordinate will be adjusted until each side of the box touches part of the non-background colored image contained in the box. If no image is found, the box coordinates are unchanged.

Below the BOX GRAB key is 'AUTO FIND'. Depressing AUTO FIND once will place the word 'FIND' on the top line of the CRT. This indicates that the FIND command is automatically invoked for each BOX GRAB or AUTO GRAB thus saving the user an extra key hit for box assignments. Depressing AUTO FIND again will turn the mode off.

BOX ADJUSTMENT COMMANDS

To aid the user in creating box coordinates which are useful, a set of box adjustment commands have been built in. These commands are located in the box adjustment keypad to the left of the numeric keypad. Two types of box adjustments are allowed: coordinate manipulation and rastor manipulation. Centered on the bottom and lower right of the keypad are the two coordinate manipulator keys 'ADJUST' and 'MOVE'. ADJUST allows the user to adjust the exact size of a box definition without modifying the image it contains. Upon depressing ADJUST, the first box selected is xored with the value 15 allowing the user to see the box while working on it. The user is then prompted via the CRT: 'Adjusting box nn - Enter GROW or SHRINK' where 'nn' is the current box number. At this point, the user may pick either the GROW key, to the left of ADJUST, or the SHRINK key to the right of ADJUST. After indicating the mode, the user may hit any of the eight arrows appearing in the center of the keypad. In GROW mode, the box will GROW only on the edge(s) specified by the currently depressed arrow. In SHRINK, the box will shrink only on the specified edge(s). In either mode, if GROW of SHRINK is again depressed, all edges of the box will GROW or SHRINK. At any point during adjustment, the user may push either 'SKIP' or 'NEXT' which will automatically move to the next selected box. 'QUIT' will stop adjusting and touching any other command will cause execution of that command to begin.

The MOVE command will begin by xoring the first selected box. After this, pressing any of the eight arrows will result in the box coordinates moving in that direction so that the whole box moves. NEXT, SKIP and QUIT work as with ADJUST. Note that ADJUST and MOVE do not effect the actual image. Only the coordinates are effected and the positioning of the inversion. The inversion (xor) is removed when ADJUST or MOVE are terminated. Both MOVE and ADJUST utilize the XORBOX primitive to first invert the box, and then to invert only the sides which are being modified.

On the left of the adjustment keypad sits three centering keys. The top is 'CENTER IMAGE'. CENTER IMAGE will take each box selected on the current page and center them, one at a time, in the center of the page. Coordinates and image area are both effected. The area originally occupied by the image before centering is replaced with the background color. Below CENTER IMAGE is 'H CENTER'. H CENTER will center the image in the horizontal direction. Only the x coordinates of the image is centered which results in the image becoming centered left and right. Below H CENTER is 'V CENTER'. V CENTER will center the image in the vertical, or up and down direction. All the center keys compute the amount of offset for the image to move and then call the SHIFT primitive to do the movement.

On the right of the adjustment keypad are 'SHIFT' and 'JUSTIFY'. JUSTIFY allows the user to position any edge of a selected box on the current page on the same edge of any other box on either page. Source and alternate arrows as well as force values are read to determine which box to justify on. The user is prompted for each selected box to enter the desired edge for justification. The eight arrow keys are used to specify the desired edge. NEXT and SKIP will automatically move to the next selected box without changing the current box. QUIT will abort justify and hitting any other legal command will execute that command.

Above JUSTIFY is the SHIFT command. SHIFT will shift any selected box a given number of pixels in any of the 8 arrow directions. After depressing SHIFT, the user is prompted to enter the direction. Touching an arrow will result in the box shifting in the desired direction. The number of pixels shifted can be adjusted by seven other keys surrounding the keypad. '1 JUMP', '5 JUMP', '10 JUMP' and '50 JUMP' will set SHIFT to move an image one, five, 10 or 50 pixels. 'XY JUMP' will prompt the user to enter a number of pixels to shift the box from the numeric keypad. If the value 127 is desired, the user pushes '1', '2' and '7' to enter the value. The '10' through '15' keys are disabled. Since the user may request one of the four angled arrows, X and Y jumps may be specified independently by using 'X JUMP' and 'Y JUMP' which request the number of pixels in the same way as XY JUMP. Thus is is possible to move a box n pixels vertical and m pixels horizontal. JUSTIFY and SHIFT both use the SHIFT primitive to move the image.

SCANNERS

Below the box adjustment keypad is the scanner keypad. The scanner keypad consists of two rows of five buttons each. The left most button on the top row is the 'SOURCE SCAN' button. Touching this button prompts the user to enter the source box scanner. The user may hit any of the 8 possible source scan keys. Under SOURCE SCAN is 'DESTINATION SCAN'. Hitting this key followed by a scanner key will enter the appropriate scanner as with SOURCE SCAN. On the upper left hand corner of the CRT display, the user may notice two legends:

```
S SCAN:  B-T..L-R
D SCAN:  T-B..R-L
```

These are updated each time the user enters new source or destination scanners. The eight possible scanners are:

1) B-T..L-R = Bottom to top, left to right
2) T-B..R-L = Top to bottom, right to left
3) T-B..L-R = Top to bottom, left to right
4) B-T..R-L = Bottom to top, right to left
5) L-R..B-T = Left to right, bottom to top
6) R-L..T-B = Right to left, top to bottom
7) L-R..T-B = Left to right, top to bottom
8) R-L..B-T = Right to left, bottom to top

In reading the legend, the first two directions (B-T, R-L etc.) are the first portion of the scan to be executed. The second portion of each legend (..L-R, ..B-T etc.) is executed after each scan of the first portion. In other words; B-T..L-R indicates that the scan travels from Bottom to Top first and then that entire scan pattern moves from Left to Right.

Commands effected by scanners are: BOX FADE, BOX AVE, BOX COPY, BOX KEY BOX FUZZ, BOX CEQC, 3D, PAINT FADE, PAINT AVE, PAINT COPY, PAINT KEY, PAINT FUZZ, STAMP FADE, STAMP AVE, STAMP COPY, STAMP KEY and STAMP FUZZ. Scanners are set by a call to TARGET at the issue of a ISM command.

Above the box adjustment keypad is found the ISM COMMAND AREA. This area consists of four lines. The lowest line is the BOX COMMAND AREA. The second line contains the FUNCTION area, the third line is the STAMP COMMAND area, and the forth line is the PAINT area.

FUNCTION AREA

The function area contains six buttons which read from left to right:

ADD, SUBTRACT, AND, OR, XOR and REP

Touching any one of these boxes will set that mode for BOX COMMANDS only. The 'Box F' legend on the upper left hand portion of the CRT will change to reflect the new choice. All box commands are effected along with disk load commands. To set the function for STAMP and PAINT commands, the user touches the 'PEN FUN' to the right of the ISM COMMAND AREA. The computer will prompt the user to 'Enter paint function' after which the user points to one of the six functions. The new choice is entered under the BOX F legend in the 'PEN F/C' legend. This enables the user to keep two functions at one time. One for box commands and one for paint/stamp commands. Setting of the function occurs when an ISM primitive is executed by first calling SETFUN with the currently chosen function.

BOX COMMAND AREA:

Below the FUNCTION AREA are a series of six keys. Each key corresponds to one of the ISM primitives. Hitting one of the keys will result in the execution of the corresponding ISM primitive using the currently set function and scanners as well as the boxes specified by the source and alternate arrows as well as the force values. Therefore, if a user wants to copy a box from one page to another, the proper boxes, function and scanners should be set up as described eariler and then the COPY box command should be issued. When the current page vectors contain more than one selected box, each box is executed one at a time. The computer will always display the DESTINATION PAGE for each box while the command is executing on that particular box. Therefore, if box zero points to the alternate page and box one points to the source page, the alternate page will be displayed while the command is executing box zero, then the source page will reappear while the command is working on box one. Display is always returned to the source page after completion of the ISM command. In this way, the user always see the result of each operation while the operation is executing. The allowed box commands are FADE, AVERAGE, COPY, KEY, FUZZ and CEQC. For the CEQC key, the replacement value for the CEQC primitive is specified by using the PEN COLOR key described below under PEN COMMAND AREA. To the upper right of the CEQC key is the 3D key. This is a seventh box command which allows the user to 3D a source box to a destination box. To the right of 3D, are 3D X and 3D Y. Both of these keys allow the user to enter a multiplier for the 3D command. Hitting one of these keys will result in a prompt for the multiplier value. The user enters the value (between zero and 255) via

the numeric keypad. The 10-15 keys will not function, so the number 123 must be entered '1' followed by '2' and '3'. A multiplier in either or both of the axes of zero turns 3D off for that axes. Any multiplier larger than 128 becomes negative causing the function to be reversed for that axes.

Two other box related commands can be found two keys below FIND and AUTO GRAB. The first, 'ZAP BOX', allows the user to erase with the background color any selected (bright) box on the current page. Alternate arrows and force values do not have any effect. Because ZAP BOX is destructive, it requires a confirmation of hitting the key twice. To the right of ZAP BOX is 'INVERT BOX'. This command will invert (xor) all selected boxes on the current page with the value 15. Alternate arrows and forces are not used. Both ZAP BOX and INVERT BOX exist to save the setup required to do the same effect through CEQC or COPY.

STAMP COMMAND AREA:

Above the FUNCTION AREA is the stamp area. The stamp area allows the user to paint on the current page. The user is allows to specify up to 16 paint brushes by defining boxes around each of the desired brush shapes. Upon touching one of the STAMP keys, the user is prompted to point to a brush. After setting the bitpad to STREAM mode, the user points to a brush which must be currently assigned to a box and made active with a select arrow. Upon identifying the brush, the user may then paint with that brush by pushing the stylus anywhere else on the page. Each time the stylus touches the bitpad surface, the currently selected brush/box will be copied via the chosen ISM primitive. The box will be copied so that the center of the box is at the stylus position. If the stylus is dragged over the bitpad surface, the chosen brush/box will be dragged along with it leaving copies at each point where the stylus touches the bitpad surface. Setting the function for this command is accomplished by 'PEN FUN' followed by the appropriate function. It should be noted that if the user overwrites the brush while painting, the brush will be modifed by the paint function which will inturn will modify itself. STAMP functions include: FADE, AVERAGE, COPY, KEY and FUZZ. COPY is the most used function allowing the user to make quick and easy duplicates of any shape on the screen. For example, in correcting word copy, the user may wish to change a letter without leaving MENU. If the letter exists elsewhere on the screen, the user may define it as a box, hit STAMP COPY and touch the cursor anywhere within the box area defining the letter as a brush. Then moving near the area to receive the correction and touching the stylus again will place the letter on the screen centered under the stylus. The user would then find the stamped letter and shift it into position. This type of edit takes less than a minute to perform and can be applied to any shape. The user may even define the entire page to stamp and stamp it at an offset to itself producing infinitely repeating patterns.

PAINT COMMAND AREA:

To provide a more general system for painting, the PAINT commands were developed and added to the STAMP commands. Paint commands include: FADE, AVERAGE, COPY, KEY and FUZZ. In the paint commands, one page is kept as the brush page. The other page becomes the canvas page. To select a brush the user would first select all possible brushes on the brush page. Then, going to the canvas page, the user places the ALTERNATE ARROW pointing to the desired brush(es) on the other page. Drawing is the same as with stamp. Only boxes pointed to by the alternate arrow will be drawn. If more than one alternate arrow exists, each brush will be drawn centered on each other. In other words, if the user selects three brushes, goes to the canvas page and points alternate arrows at all three, all three brushes will be used, each overlapping the other to produce a composite brush. If AVERAGE PAINT is used with three brushes, each will appear transparent allowing the viewer to see each image through the others. FADE painting using a brush which is a very small metallic box allows the user to 'buff' an area. This effect produces very good smoke and clouds and can also be user for buffing smooth an edge. Since the commands are treated in the same way as their BOX counterparts, scanners and function (PEN F) are in effect. This also means that the color vector is treated identically as with box commands. For example, the KEY PAINT command allows the user to paint only on a certain color. Therefore, a full color image may be brushed onto any part of the canvas page without touching other portions of the image. If the user KEY paints onto the background color only, the brush will appear behind any existing image, even the most recent brush. This allows the user to paint away from themself as opposed to towards.

PEN COMMANDS AREA:

The STAMP and PAINT commands border two categories, that of box commands and that of pen commands. STAMP and PAINT use boxes hooked up to the pen to enable brushing on the screen. To the right of the BOX/STAMP/PAINT area is the PEN COMMAND AREA. The top left key of this area is the PEN key. Touching this key will put the user in 'PEN MODE'. In this mode, the user may paint with a single colored pixel onto the current page. The pixel is placed using the function specified by the 'PEN FUN' command (discussed above). To the right of the 'PEN FUN' key is the 'PEN COLOR' key. This key allows the user to specify the color of the pixel drawn by the PEN command in two ways. After touching PEN COLOR, the user is prompted to 'ENTER COLOR'. The user may either touch one of the numeric keys, 0-15, indicating that color, or the user may put the bitpad in STREAM mode and touch the stylus to some portion of the image. The color under the cursor point will be used as the chosen color. PEN COLOR then performs several functions, first allowing the user to specify a color to be used by the pen, and secondly, to allow the user a quick way of determining the color of a particular area on the screen without having to resort to the LITE commands, and thirdly, to allow the user to specify a color replacement value for the CEQC box comand (described above) and the PEN FIX command (described below). The current pen color is displayed to the right of the 'PEN F/C' legend on the screen. PEN COLOR is always bright. If the user is painting with a pen color of one and a pen function of ADD, the value one will be added everywhere the user points. To facilitate quick color changes (useful for MACROS described in CONTROL COMMANDS below), two keys 'PC +' and 'PC -' have been developed. 'PC +' will increment the current pen color and PC - will decrement the current pen color. If zero or 15 are exceeded, the pen color is wrapped around. PEN uses BITPAD, CURSOR and FXDOT to do the actual drawing.

To the right of the PEN key is the PEN FUZZ key. PEN FUZZ uses the SFUZZ primitive (which only fuzzes a single neighborhood) to only fuzz a eight pixel neighborhood. The ninth, or center, pixel is centered under the stylus point. Scanners, function and color vector all have meaning. This allows the user to go into an existing image and smooth edges only where desired by the user. Where the FUZZ BOX COMMAND will smooth an entire box, even the portions which do not require smoothing, PEN FUZZ will only smooth where the user points to.

To the right of PEN FUZZ is PEN FIX. PEN FIX is CEQC hooked up to the stylus point. By specifying the colors to modify in the color map, a function via PEN FUN and a replacement color via PEN COLOR, the user will only alter the colors specified by the color map with the current pen color. This is very important in cleaning up an image. The user may wish to remove only color five from a complex area. To do this, the user would brighten color five in the color map, specify a replacement value of zero in with PEN COLOR and specify a PEN FUNCTION of REPLACE. Then any area the user points to while in PEN FIX will convert only color five to color zero.

CONTROL COMMAND AREAS

The bitpad consists of five separate control command areas:

1) RESOLUTION/WINDOW CONTROL
2) CAMERA CONTROL
3) PAGE BACKUP CONTROL
4) DISK CONTROL
5) MEMORY CONTROL

RESOLUTION/WINDOW CONTROL:

Above the PEN COMMAND AREA is the RESOLUTION and WINDOW CONTROL AREA. Our particular hardware allows sets of eight pixels to be declared as high resolution. While the rest of the screen remains low resolution. The first key, 'HI RES', will place the entire screen in high resolution (756 x 482). To the right is 'LOW RES' which places the entire screen in low resolution (378 x 241). To the right of LOW RES is 'RES BOX'. RES BOX takes all selected boxes on the current page and fits an inverted resolution area into them. Therefore, a selected box when RES BOXed will flip from its current resolution to the inverted resolution. The area which is affected is as close to the box size as possible within the eight pixel per resolution area rule. To the lower right of RES BOX is the 'RES ON/OFF' key. This key allows the user to specify the entire display or the currently set high resolution areas to be displayed in low resolution. This allows the entire screen to be displayed differently than it is configured. To the right of RES ON/OFF is 'WINDOW'. WINDOW takes each high resolution area on the screen and opens a window to the alternate page in its position. Therefore resolution areas may also specify windows depending on the mode. Both RES ON/OFF and WINDOW are hardware dependent commands and make use of the NYBBLE and WINDOW primitives. Windows are only effective from page zero to page one. Windows may not be opened on page one. The top line of the CRT contains two positions devoted to RES ON/OFF and WINDOW. If the option is not enabled, the corresponding legend is blank. If an option if enabled, the word 'RES ON' and/or 'WINDOW' is displayed bright.

CAMERA CONTROL:

Above the PAINT keypad is the camera keypad. The leftmost key is the 'SMALL' command. This corresponds to the FRAME1 primitive. If this key is hit, the user is prompted to confirm by hitting the key again. Once confirmed, a 256 x 241 camera area is scanned onto the center of the screen. To the right of SMALL is 'LARGE'. This takes a 189 x 120 camera area and displayes it twice the size over the entire frame. The third key is the TILT camera load. TILT takes 189 in the X axes and blows it twice the size but takes all 241 in the Y axes making the X axes appear tilted 45 degrees.

PAGE BACKUP CONTROL:

To the upper right of the numeric keypad appear three page backup keys. These keys allow the user to quickly backup and restore the current page with the alternate page. The first key is 'COPY S TO D'. This key requires a confirmation of hitting the key again. Once confirmed, the entire current (or source) page is copied to the alternate (or destination) page. Included in the backup is the entire source page resolution area and colormap/box storage area. The second key is the 'COPY D TO S' key which allows the destination (or alternate) page to be restored on the current (or source) page. This command also requires confirmation. The third key is 'SWAP S WITH D'. This key requires no confirmation and will perform a high speed swap of both image planes. All three commands are extremely high speed, completing their operations in under a second. Colormaps, select/alternate arrows, force values and functions are all ignored. These keys use the CPPAGE and EXPAGE primitives to perform their functions.

DISK CONTROL AREA:

At the top of the bitpad towards the center are the eight disk control keys. The first 'LOAD' allows the user to load an image from disk. The command prompts the user to enter the filename from the keyboard. Once entered the corresponding image is loaded using the currently set BOX FUNCTION onto the current page. After the load, the colormap the image was saved with is assumed ONLY IF the currently set function is REPLACE. Once an image is loaded, the filename of that image is placed in the upper left hand corner of the CRT. If the file is a write protected file, the symbol '(P)' is placed after the name. A image filename may be up to eight characters long.

To the lower left of the LOAD key is the 'LAST LOAD' key. This key will load the last file saved or loaded. In other words, this key will load the file indicated by the name at the top of the CRT. This saves retyping in the filename and is very useful when ORing a file in and XORing it out to produce a mask.

To the right of LOAD is the 'SAVE' key. Touching this key will prompt the user for the filename to save the current image on. If the file already exists or is protected, the user is asked to confirm the action by touching SAVE again. To the lower right of SAVE is the 'LAST SAVE' key. This key acts as LAST LOAD except it resaves the current image on the last file loaded or saved.

Below LOAD and SAVE are two keys 'INC' and 'DEC'. These keys help to keep the user from going to the keyboard more than necessary. Most slides have more than one associated image file. For example, a single slide may be composed of four separate exposures. Each exposure is saved in a separate image file. Thus, a slide named 'CHART1' may be made up of CHART1A, CHART1B, CHART1C and CHART1B. INC allows the user to increment the last letter of the current filename on the upper left hand corner of the CRT. DEC allows the user to decrement the last letter of the filename. This allows the user to move from CHART1A all the way to CHART1D and back without ever going to the keyboard.

Below INC and DEC are 'A: DIR' and 'B: DIR'. These two keys allow the user to receive a directory of disk A and disk B. The CRT display is erased and the directory is displayed. The user is then prompted to enter any character from the keyboard at which time the CRT is again erased and the original display rewritten.

MEMORY CONTROL AREA:

To the upper right of the numeric keypad is the memory keypad. The memory keypad consists of five keys: 'MEM', 'SET', 'DEL', 'LOOP' and 'RUN'. The memory feature allows the user to have the computer watch-and-learn a particular sequence. Once the sequence has been entered, the user may request it repeated up to 999 times. Up to 16 different sequences may be remembered at one time by the computer. Each sequence may consist of as many instructions as needed until the stack is used up.

The MEM key turns memory on and off. Touching the key once will place the 'MEMORY' legend to bright at the upper right hand corner of the CRT. When memory is on, each keystroke entered is executed and remembered by the computer. If an error is entered, the user may hit the MEM key again. This will turn memory off and delete the entire sequence allowing the user to start over. If a sequence is entered correctly, the user must assign the sequence to one of the macro spaces. To do this, the user hits the SET key. SET allows the user to set the current memory to a macro number. The user is prompted to enter a number from the numeric keypad. Legal values are zero thru 15. Once the number is entered, the macro is stored under that number and the memory legend is erased and turned off. To run a macro, the user need only hit the RUN key. The computer will request the macro number from the numeric keypad. Upon receiving the number, the macro will be executed once. To run a macro more than one time, the user should hit the LOOP key. The computer will request a value from one to 999 be entered from the numeric keypad. Once entered, the loop value will be displayed to the left of the MEMORY legend area. The user may then use the RUN key to specify the macro which will be repeated until the loop value is zero. The DEL key will delete any macro by allowing the user to specify the undesired macro from the numeric keypad. Any macro which uses ADJUST BOX or MOVE BOX will inhibit the inversion of the box to improve speed. Macros are extremely useful for

repeating large boring tasks and for creating automatically run animation sequences.

OTHER FUNCTIONS

Included in the bitpad menu layout are several other function keys which aid the user in manipulating the display and environment.

As mentioned before, to the right in the numeric keypad is the 'P' key. This key will flip the display to the alternate page each time it is hit. The 'PAGE 0'/'PAGE 1' legend will alternate from bright to dim to indicate on which page the user is currently. If two colormaps are in effect (1/2), the proper colormap is assumed. The P key also handles the internal switching of source, alternate and force vectors from the page zero vector set to the page one vector set.

The upper left hand corner of the bitpad contains the 'CRT' key. Touching this key will cause all information on the CRT to erase and then reappear. This allows the user to verify what appears on the CRT. This is helpful for problems due to data transmission error or keypunching error.

To the right of the CRT key is the 'IO' key. Touching this button once will erase the CRT and display the message 'HIT CRT OR IO TO REDISPLAY'. This mode will inhibit the computer from sending any information to the CRT. In this mode, the user is operating blind as far as box coordinates and colormap values and other legends are concerned. This command is used to speed up the system, especially when modifying colors. When IO is inhibited and the user is rolling the colormap, color will rotate with speeds in excess of 64 colors per second. This command will also allow macros to run faster by inhibiting all display of new coordinate updates, etc.

To the right of the IO key is the 'ZAP' key. This key requires the user to confirm by hitting the key twice. Once confirmed, the current page is cleared with color zero. To the right of ZAP is 'ZAP ALT'. After confirming this key, the alternate page is cleared with color zero. This allows the user to delete the alternate page without having to go to the alternate page before executing a command which will result in the alternate page being modified.

To the upper left of the bitpad under the CRT key are two keys, 'FAST' and 'SLOW'. These keys will set the bitpad (via BITAVE) to fast or slow operation. In fast, the bitpad coordinates are simply returned to the user when in SWITCH STREAM mode. In slow mode, each point is averaged with the last point and any points further than 15 pixels away from the last point is discarded. This allows more accurate drawing with the stylus. This only effects the bitpad when the SWITCH STREAM switch is in effect.

The final key is in the upper right hand corner of the bitpad. This key physically contains the area of four keys grouped two by two. This key is the 'EXIT' key and will exit the user to the operating system.

FUTURE FUNCTIONS

Our software system is forever being modified and improved. Because of this, we have several functions which currently exist as FORTRAN test programs that are waiting to be converted into ASSEMBLY LANGUAGE primitives and incorporated into MENU. Several such functions require mentioning. The first, FILL is a adaptation of the SMITH algorithm as presented in the SIGGRAPH '79 proceedings. This is a very high speed horizontal scan line flood fill algorithm which uses less than a 2K stack for worst case fill. MENU currently has a 2K fill stack reserved for this function.

The second routine to be included is a smooth zoom algorithm. This routine is basically FUZZ in reverse. An image is enlarged by plotting each pixel leaving a background pixel in between. In this way, a 2 x 2 pixel area is converted to a 3 x 3 pixel area. A neighborhood scanner is then envoked to average the intensities of the four existings pixels in order to create the missing data. This has the effect of zooming up a raster image while keeping all edges smooth and defined. Instead of duplicating pixels when zooming, this routine interpolates the five missing pixels to determine the proper data which should exist there. This keeps the edge and prevents the zoomed image from becoming blocky. The following example will help clarify the effect:

```
Before zooming:
  PP  =  1,9      - Four pixels valued 1,9,4 and 6
  PP  =  4,6        in a 2 x 2 area.

First zoom step:
  PbP =  1,0,9    - Image is zoomed leaving
  bbb =  0,0,0      background pixels between
  PbP =  4,0,6      the original pixels

Last zoom step:
  PPP =  1,5,9    - Background information is
  PPP =  2,4,7      interpolated by averaging the
  PPP =  4,5,6      endpoints.
```

Finally, no provisions have been included for saving multiple maps using the GETMAP/PUTMAP primitives. These will be included with a quick recall method allowing the user to be able to ripple through all 96 stored maps.

OBSCURITY

Because the menu system allows the user to have extreme amounts of control over the graphics environment certain measures have been included to insure ease of use.

As described earlier in the paper, the CRT is divided into many bright and dim areas. Only bright areas are of importance to the user. This allows the user to quickly focus the eyes on the desired area of the CRT without having to search around. Secondly, all legends output to the CRT are always cursor addressed as apposed to scrolled. This always keeps information in the same place on the CRT reducing the amount of searching required. The CRT cursor is turned off to keep it from becoming distracting. Display updates are all handled by assembly language routines which talk at 19.2K baud to the CRT. This makes legend updates extremely fast and allows the entire screen to be rewritten (CRT key) within five seconds.

To keep a rapid flow of commands from the user to the computer, all commands which keep you in a mode (ie. SHIFT, ADJUST, SET RGB etc..) are abortable by using the QUIT key or by just hitting another key to immediately execute another command. Keys which require confirmation, such as ZAP, ZAP BOX etc., only require the key to be hit twice. The CRT always pauses at least two seconds before requesting a confirmation via the legend '<<CONFIRM?>>'. Confirmations are only allowed in the POINT mode which means that the confirm button won't accidentally stream in via the SWITCH STREAM mode.

However, no matter what precautions one takes in designing a software package, some amount of obscurity always occur. In our system, the weak point is in the complexity of the commands. Operators often forget that smear scanners are in effect, or that XOR is set instead of REPLACE. To help reduce the amount of incorrect keystrokes, no command is executed if the basic components for its execution are not present. For example, if colors are to be set and no color is bright in the color vector, the color command will beep and display a 'missing color map' error on the CRT.

Another problem which presented itself to us was the fact that the system was too powerful. It takes a new operator about two months to become effective on the computer (working around production). The reason for this is the large number of possible effects for each command. You can tell a potential operator the purpose behind taking a rastor image, assigning it a box, edge detecting only color 15 and xor plotting it back on itself rotated 180 degrees, but it still takes them time to experiment with the effects and all their functions before they realize the use of such a

sequence. The obvious solution to this problem is large amounts of high quality documentation expressing all useful controls and functions using pictures to convey the actions.

PROGRAMMING PHILOSOPHY

In designing computer graphics software for small systems, one must be careful as to the approach taken with the actual programming. While structured programming is a good practice, keeping to all the structured programming rules will result in a program which runs slowly and poorly. The method we adopted for our system was that of a powerful primitive system which allowed the primitives to call themselves. To handle global events such as scanners and functions, each primitive which uses them, when called, calls $SCAN$. $SCAN$ is called by placing in the A register the value zero (for source scanner) or the value one (for destination scanner). The HL register should point to the position in the primitive itself where it calls the actual scanner. The $SCAN$ function then writes into the address pointed to by the HL register the address for the proper scanner. Therefore, each primitive when called immediately calls $SCAN$ to 'write' the proper CALL address into the primitive code itself. In the same way, the function specified to the SETFUN routine is converted by SETFUN to a one byte instruction (ie NOP; ADD A,D; SUB A,D; XOR A,D; OR A,D; AND A,D) and stored in a global location labeled $$FUN. When a primitive is called which requires a function, it loads the function from $$FUN and writes it into the area needed in itself. This area is reserved with a NOP instruction. This means that when each ISM primitive is executed, it actually rewrites itself to perform for the currently set environment parameters. This saved a large amount of code from ever being written. However, self modifying code is also a bad programming practice. It must be seen that when programming for an application which requires a tight hardware/software integration, certain programming practices must be weighed against factors such as speed and efficiency.

CONCLUSION

Our design has shown that to force the most amount of work out of the hardware, each software primitive must be designed to allow the most number of functions possible within that primitive. Great care must be given to the computer/user interface to help ease the possible obscurities which may result. Care must also be taken in using each device to its greatest potiential. Functions such as screen area should not be duplicated as many manufacturers do by placing the screen area in the bitpad and displaying the menu choices in the image plane. This is a great waste of both speed and space and should be avoided by careful planning of the available devices.

Finally, it can be seen from the presented system, that an extremely advanced graphics software package can be implemented on a small computer system. The advent of secondary sources of image boards designed for small computers such as the APPLE II and IBM PC etc., allows programmers to create highly advanced artist workstations for less than $4,000 dollars. Together, the new hardware combined with new software can make for a complete product which will outperform much of the currently, commercially available equipment that costs hundreds of thousands more.

ACKNOWLEDGEMENTS

I would like to thank Deborah Sellers, Andrea Swetish and Walter Wright for their valuable time spent in suggestions and corrections on this paper and Bill Romanowski for fresh ideas including the 3D algorighm. Also, thanks to our EPSON printer for printing this paper and diagrams many times without breaking down. And a final thanks to J. D. Foley and A. Van Dam for their wonderful book 'Fundamentals of Interactive Computer Graphics' which has provided valuable insights on all the algorithms that could never run on a micro-computer.

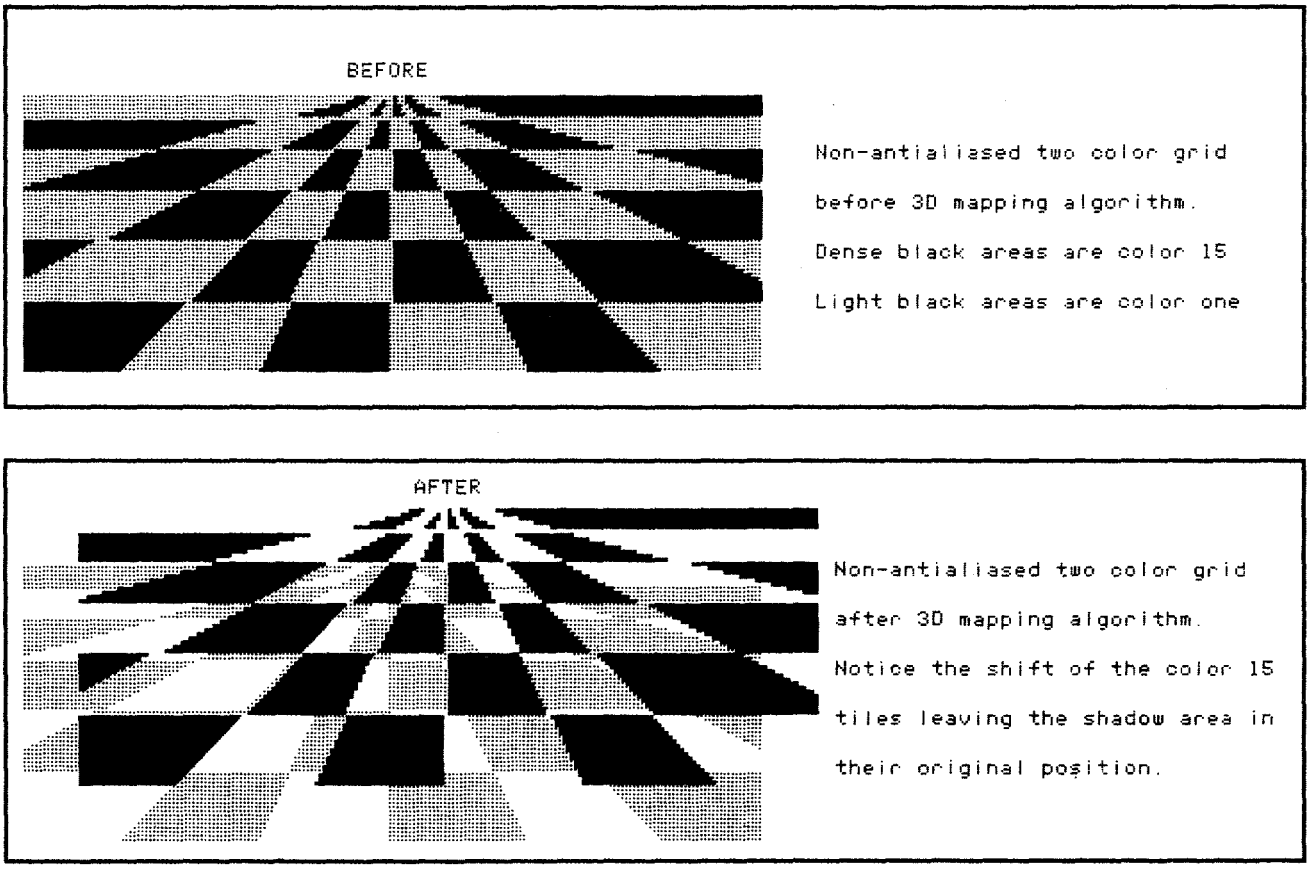


FIGURE 1 - Tiled surfaces before and after 3D treatment

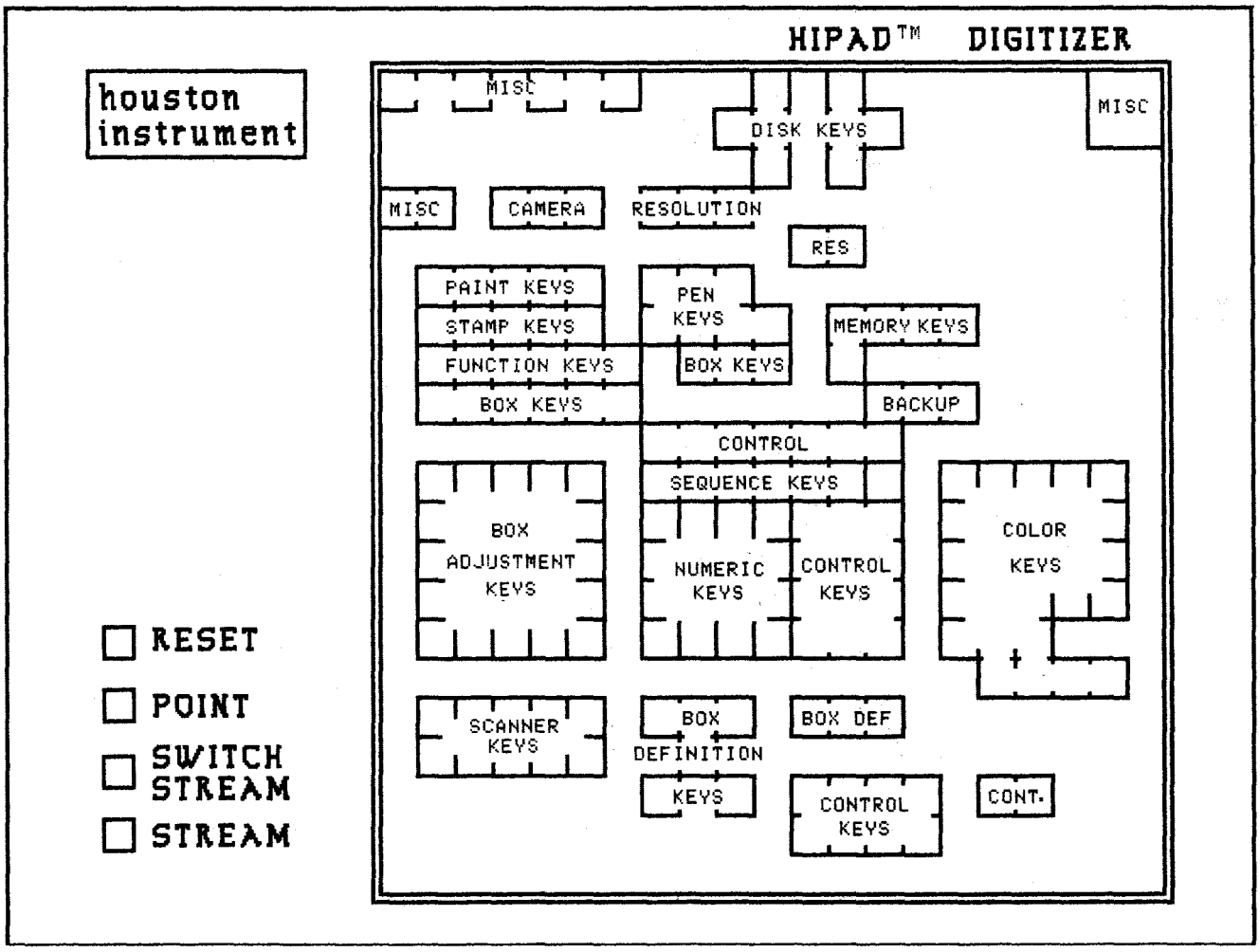


FIGURE 2a - Bitpad surface: Buttons and menu/screen area

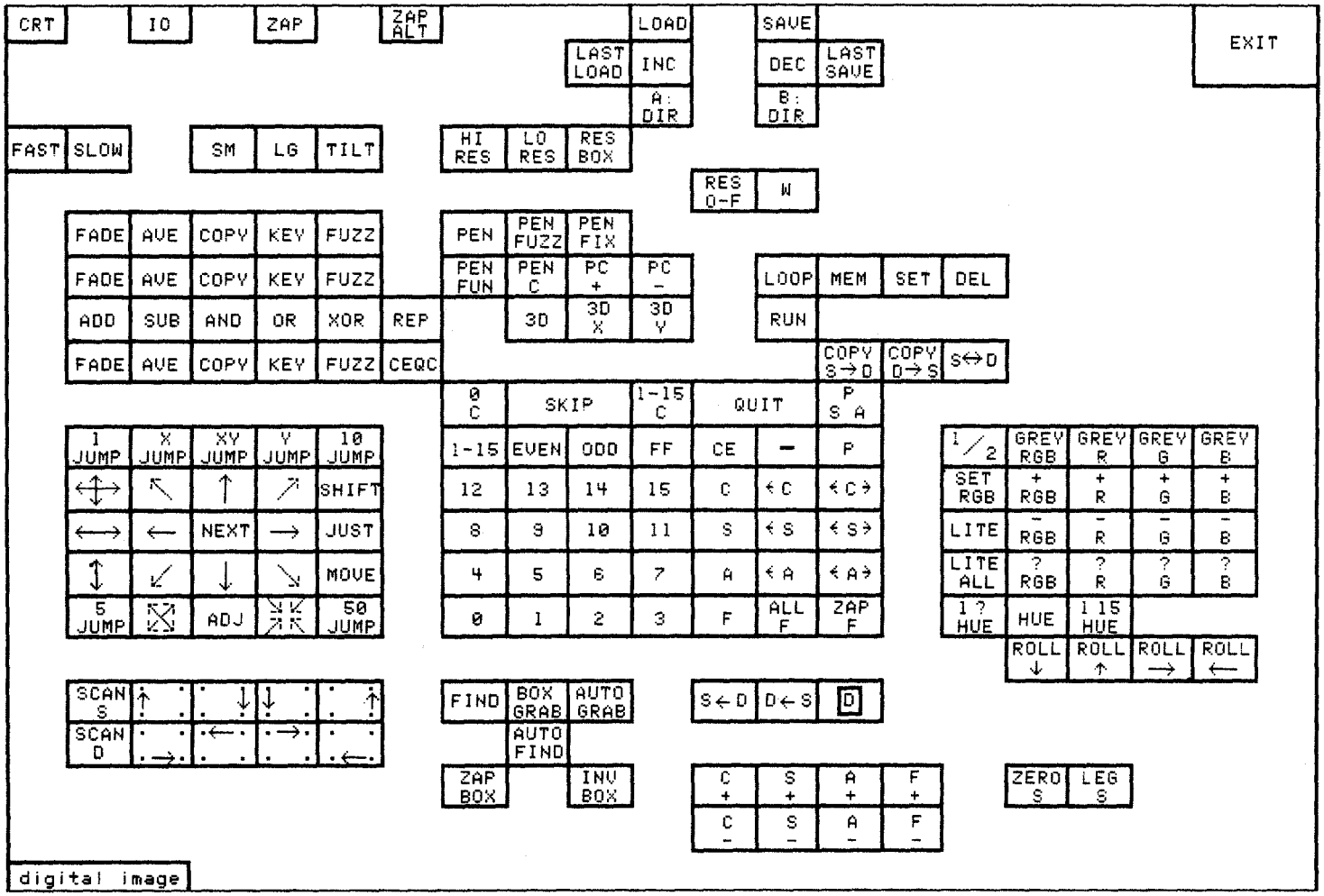


FIGURE 2b - Bitpad surface: Menu key closeup

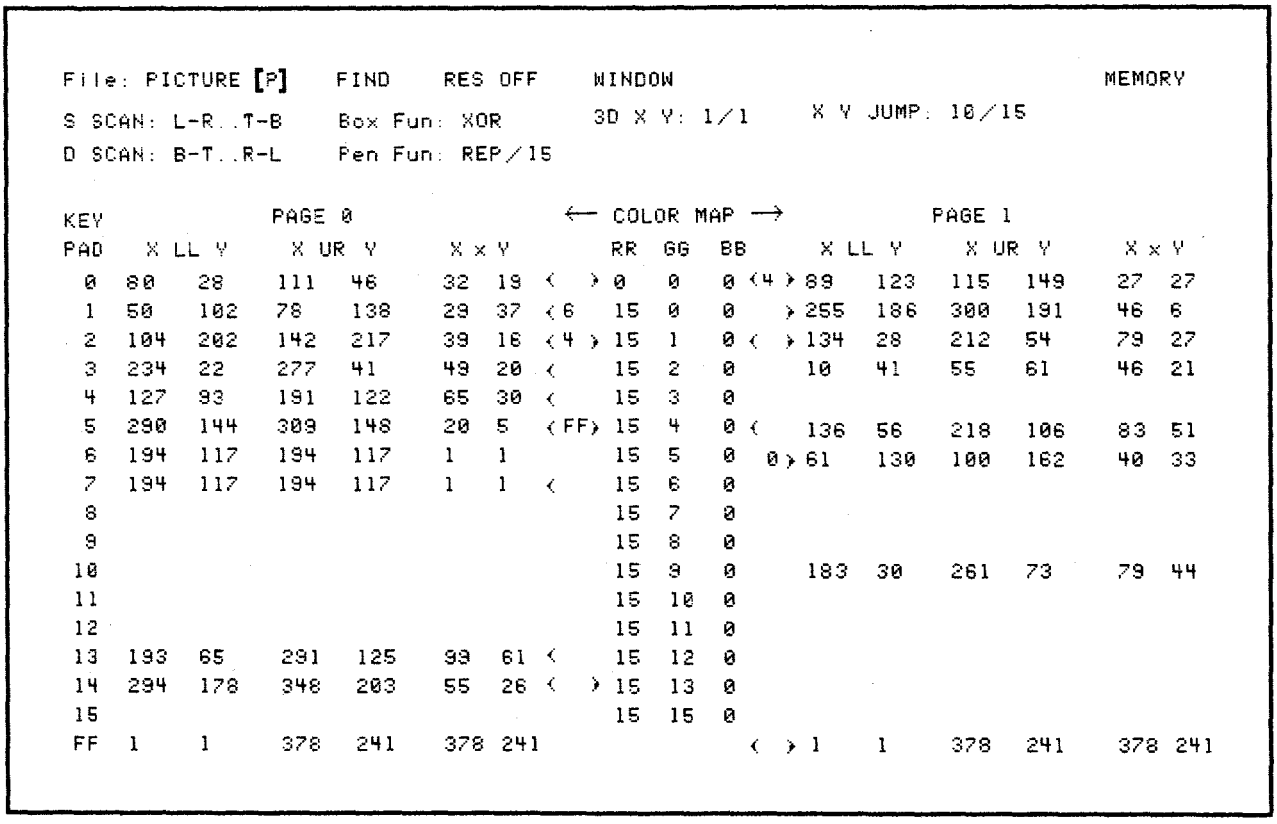


FIGURE 3 - Menu CRT display

DEVELOPING A 3-DIMENSIONAL ANIMATION SYSTEM FOR THE DIGITAL 11/23 MICRO COMPUTER

Marla Schweppe

Art Education Department
The Ohio State University
Columbus, Ohio

The animation system currently in use in the Art Education Department at The Ohio State University will be the topic of this presentation. The hardware configuration, the software or animation tools which are currently in use and the new projects under development will be discussed. This discussion will include data generation, the creation of 3-dimentional scenes, the process of animation, and the final product. The courses offered here in computer animation will be briefly outlined and animation produced by students working on the Digital 11/23 will be shown. The possibilities and limitations of doing animation on the 11/23 will be discussed in depth.

This presentation is designed for people with a general understanding of computer graphics.

The computer animation group in the Art Education Department at The Ohio State University has a Digital Equipment Corporation PDP 11/23 microcomputer system. Students also have access to a VAX 11/780 through the facilities Computer Graphics Research Group. The 11/23 animation system is basically a limited version of the animation software available on the VAX. The 11/23 is used primarily in the introductory computer animation class. The environment is a research one. This means that new ideas are constantly being tested. The animator therefore has an opportunity to make contributions to the development of new software. However, this also means that the software is in a constant state of flux.

Hardware Configuration

For the first two or three quarters students design their animation on the PDP 11/23. This 11/23 has a 16 bit micro processor, floating point, memory management, a quarter mega byte of memory, and four serial interfaces. The interfaces go to three terminals, two VT100's with VT640 retrographics boards and an Zenith H-19. The fourth serial interface is used for communication to the VAX (mail and shipping files back and forth). The VT100's with 640 x 480 line drawing capability and the H-19 which gives a very coarse image are the principal display devices. There are two, double-density, floppy drives with a half a mega byte of memory each and ten mega bytes of memory on four surfaces of hard disc.

Software/Animation Tools

The system uses UNIX Version 7 DEC modification 2.1, the C programming language with a math library and the Bourne shell. The animation software was written locally, originally for the VAX and was transported to the 11/23. The local software includes scn_assmblr[2], which allows for the manipulation of objects and view parameters and a data generation program[1].

Data Generation

Data or objects can be created using either the data generation program developed by Wayne Carlson and modified by Paul MacDougal or by hand. In creating data by hand the animator must specify points in x, y, z space and connect those points to specify polygons. This is done in an ascii format. The data is then transformed into a binary format for later use. We also use an ascii description file which contains necessary information for the animator. It includes information like the name of the binary file, a comment, the display algorithm to be used and the bounding box of the object. Currently there is only one display algorithm which outputs to the VT100 retrographics display or the H-19. Later, with a color display and therefore different display algorithms, the data description file will also contain some information about color. Since the ascii data files take up a fair amount of space in memory, these are usually deleted leaving the binary version of the data and the ascii description file. If for some reason the ascii data file is needed again it can be converted back into the ascii format. It is also possible to have more than one ascii description file pointing to the same binary file. For instance, different display algorithms or color maps could be specified for the same piece of data.

Three types of data can be created using the data generation program: solids of revolution, tubular objects, and projected objects. The data generation program (DG) uses menu selection in combination with input from the terminal. In the

2d drawing mode the user specifies paths and profiles for solids of revolution, 2d paths for tubular objects, and profiles for projected objects. There are menus for each type of object that allow the artist to adjust various parameters. When an object is output from DG, the binary data file and the ascii description are created. The animator can convert this binary file to ascii format if s/he wants to change the object by changing the coordinates of some of the points.

Scene Description

In order to describe scenes we use a program called scn_assmblr which was developed by F.C. Crow for use on the VAX and transported to the 11/23 by Michael Girard. Scn_assmblr allows the animator to instance a piece of data as many times as s/he wants by simply calling it any number of times. It then allows all objects called to be placed, rotated and scaled independently in three dimensions. Once the scene is created the animator can decide where the eyepoint should be and in which direction it should be pointed. The animator can also control the view angle, which gives the effective control of a zoom lens going from wide angle to telephoto. The user can ask to see the scene from any direction at any time and can then make the necessary adjustments to get the desired effect.

Scn_assemblr can be used in an interactive mode or it can take input from a file. In the interactive mode, the commands are typed in line by line. This mode is useful when initially creating a scene. There is a command which causes the current status of all parameters to be output to a file for later use.

The mode which allows input from a file is useful both for viewing of a single frame which was created in the past and stored in a file, and for doing animation. Animated movement is created by moving an object or objects some amount from frame to frame. Therefore it is preferrable to write a script for input into scn_assmblr which will describe one frame after another, display the frame on the screen and send a signal to click the camera. If the animation is to be recorded on 16mm film as we do, 24 frames per second must be generated. This means there are 1440 frames for a minute of animation. There are several systems available on the VAX which generate these long files that describe each of the frames. One is a key framing system. One is a skeleton animation system and another simply does linear interpolation for independent objects. For the present the scripts for animation are being created by an adhoc method. The animator writes a C program with lots of print statements and arithmetic expressions that generates a file in the proper format for scn_assmblr. This method requires that the animator have a fair amount of programming experience. Animation students have also put their programming skills to use writing subroutines that perform various functions like ease-ins, ease-outs, acceleration and deceleration.

The Development of the System

The main objective of the development is to create an environment where students can become familiar with the tools for computer animation in a way that is easily transferrable to later work on the VAX. At the same time, the system must include enough tools for the beginning student to create interesting animation and to transfer work done on the 11/23 to the VAX for later use. For this reason, the software currently implemented on the VAX was transported to the 11/23. In order to transport scn_assmblr the limit of objects allowed was lowered from 64 to 16 and the number of points and polygons per object was lowered from several thousand to 700. In order to transport DG, the option to create objects by lofting was eliminated. The mode of interaction was transferred from a Megatek to the VT100 with keyboard control of the cursor. The code for DG also had to be rewritten into modules that would fit onto the system at the same time as the data being created. Many programs available on the VAX will not be transported to the 11/23 due to size limitations, but others are under evaluation. The current display algorithm includes clipping, perspective and the removal of backfacing polygons.

Future Improvements

A Peritek Frame Buffer and a Barco monitor will provide a 512 x 512 x 4 display with 16 colors chosen from 4096. Most of the user interfaces now available on the VAX will be transported. TWIXT, a 3d keyframing system written by Julian Gomez, is one example of such an interface[3]. This system allows the user to specify key frames of an animation and the intermediate frames are generated through linear interpolation. A version of TWIXT works from a keyboard, and will be transported to the smaller machine. TWIXT on the VAX interacts with a Megatek in near real-time display. This capacity allows for fast feedback and debugging of animation. Only single frames can be displayed on the 11/23. The only way to provide real-time playback for animation done on the 11/23 is through film. The skeleton animation (SA) system developed by David Zeltzer[4] is also planned for transport.

Outline of the Beginning Computer Animation Class

The beginning Computer Animation class is designed for students with a background in art and computer science. Students are also required to take a survey course in Computer Graphics which familiarizes them with the history, state of the art and basic terms of computer graphics. The introductory class presents a wide variety of information. During the first quarter the student must become familiar with the system and local software, basic film techniques, the theory of additive versus subtractive color, temporal-spacial skills and programming skills. Relative to system software, the student must learn the UNIX operating system, the editor, and the C programming language. They must become

familiar with the application of basic film techniques like cuts, pans, and zooms through control of the eyepoint and direction of view. Conventional animation is studied in depth in order to develop a sense of timing and motion control. Most students must switch their thinking from using subtractive color as with pigment to using additive color.

The student first creates a scene using primitive objects (balls, cubes, cones, pyramids and cylinders). The student draws the scene in orthographic projections before beginning work on the computer. Two objects in this scene must be in contact with each other in order to demonstrate control over placement. Three, well-composed views of the scene must be displayed on the screen. This demonstrates control of the eyepoint and direction of view. This assignment familiarizes the student with scn_assmblr without having to know any other programs.

The second assignment requires that the student build three objects using three possible options of the data generation program. This assignment familiarizes the student with the kinds of objects that are easily built in DG.

Assignment Three requires that the student build an object by hand. The object must be drawn in orthographic projections. If it is a complicated object a rough model may need to be built as well. The object must be one which cannot easily be built using the methods of DG. This avoids wasting a lot of time building an object by hand that could have been done more quickly in DG.

Controlling movement is crucial to animation so the last two assignments require the student to demonstrate motion control. The first requires that the student use one or two objects in 10 seconds of animation. A storyboard of the animation is required. This gives the student practice in visualizing an animation from start to finish and forces them to control the movement and timing in a structured rather than an arbitrary way.

The second animation allows the student to use up to the maximum number of objects (16) and encourages the student to create an environment as well as have moving objects.

Summary

A microcomputer can be a powerful tool for doing animation. The animation will not compare to that utilizing complex display algorithms and large data bases generated on larger systems. However, if compared to the many conventional animation techniques, the animation that is possible on a micro should entice many artists. Much, state-of-the-art computer animation done today is designed to produce special effects. The micro, with current capabilities, is less useful in creating animation for these purposes. The relative low-cost of micro systems however, should make them more available to artists to develop standards and an aesthetic based on the system's potential.

References

1. Techniques for the Generation of Three Dimensional Data For Use in Complex Image Synthesis, W. Carlson, Ph.D. Thesis, The Ohio State University, September, 1982.

2. "A Comparison of Antialiasing Techniques," F.C. Crow, IEEE Computer Graphics and Applications, January, 1981.

3. TWIXT User's Manual, Julian Gomez, Computer Graphics Research Group, Columbus, Ohio, September, 1983, Unpublished.

4. "Motor Control Techniques for Figure Animation," D. Zeltzer, IEEE Computer Graphics and Applications, Vol. 2, No. 9, November, 1982, pg. 53-59.

3-D Computer Graphics for Artists

Judy Sachter

Department of Art Education
The Ohio State University
Columbus, Ohio 43201

Artists need to understand the techniques and limitations of computer graphics in order to fully utilize this new medium. While graphics software may differ from system to system, this paper addresses basic concepts in 3-D computer graphics that are independent of a particular language or graphics device.

1. Introduction

Computer graphics opens up a totally new medium for the artistic community. In order to fully utilize this new medium artists need to understand its techniques and limitations. The nature of computer graphics demands a variety of both technical and aesthetic skills. Artists have to learn where previous artistic skills and concepts will or will not transfer to this new medium. It is important for them to become comfortable with the computer environment, both the hardware and the software. They need to familiarize themselves with the use of the computer, the operating system, the text editor, and the graphics language as well as the input and output devices available on the particular system they are using.

As artists see the world (real or imaginary) they create an image of this world in their minds. This image is transformed mentally into a visualization, and rendered in a particular medium either two or three dimensionally by the artist. In 3-D computer graphics it is necessary to create a mathematically model of the world in the computer memory. Creating objects in this 'digital' world is similar to creating sculpture or architecture. The artist needs a good understanding of space and form. These objects can be painted and manipulated by changing their size, position in space, and their orientation (rotation) in space to create a scene. This scene is similar to setting up a stage with actors, props, and background for film or theater. The light or lights are placed for the best effect. Then the artist chooses a window from which he will view this world and where he is looking. This building of models, and organization of space, color, light and view are where the artist has control, the rendering of the shaded 2-D image is all done by programs or algorithms.

Artists consider the level of realism of their work in relation to the subject matter, the materials used, the amount of time they have, their skills at rendering realistically, the amount of detail necessary, the expressive content, and how it will be perceived by an observer. This is also true in computer graphics. Computer scientists must take into consideration the level of realism needed for a particular application, the amount of detail recorded in the model, the processing time required of a computer to generate the image, the capabilities of the computer and the hardware display, and the perceptual effects of the image on the observer. This has led to choices in how the artist and the computer scientist model the world in a way that is appropriately believable for a particular purpose.

Graphics software may differ from system to system; there may be more options on some systems than others and the graphics or animation language may be different on each system. This paper addresses the basic concepts in 3-D computer graphics that are independent of a particular language or graphics device.

2. Illusion of Depth

The basic problem addressed by visualization techniques in 3-D computer graphics is depth cuing[2]. Artists have used various techniques to create an illusion of 3-D space on a 2-D surface using an intuitive knowledge of space and the human visual system.

Largeness of scale can be interpreted as nearness, and conversely, smaller scale is interpreted as spatial distance. The position of a form near the lower edge is perceived as near spatially and anything above the horizon line or the center is interpreted as farther away.

Linear perspective is a geometric system which uses the spatial indication of size, position and converging parallels and converts size and distance into a unified spatial order as seen from one viewpoint. This visual logic of linear perspective can be programmed into the computer.

Overlapping planes or volumes are a powerful indication of space and take precedence over other depth cues. An object covering the visible surface of another object is assumed to be nearer. Overlapping, one of the strongest depth cues, is handled by what is called a hidden-line or hidden-surface algorithm. This eliminates

CH1930-7/83/0000/0068$01.00

surfaces or parts of surfaces that are not seen from the designated view. Transparencies can also be synthesized by the computer but are expensive computationally. Light is a significant indicator of volume of an object in space. Light reveals the form and surface qualitites through a gradation from light to dark. Other surface qualities such as color, texture, degree of reflection are the result of the play of light on the surface. Shadows indicate both the form of solid objects as well as position of the light source. [3] [6]

The qualities of light can be modeled in the computer in various ways. Diffuse shading can be easily simulated in the computer. Some effects create more realism, such as cast shadows, transparencies, and reflections, but are computationally very expensive. The construction of the human visual system doesn't allow the eye to see near and distant objects at the same time with equal clarity. Distant objects lose their sharp edges, have less detail, and colors tend to be lighter and less brilliant. Artists call this phenomenon aerial or atmospheric perspective [3,6].The depth cue of atmospheric effects might be programmed with a haze factor and adjustment of colors in the distance. The techniques necessary to incorporate most of these depth cues into computer graphics algorithms are worked out [5], but when programing depth cues the scientist has to take into consideration the amount of computation time for an effect.

3. The Cartesian Coordinate System

The world is modeled in a digital computer mathematically in a form understandable to the computer. Each 3-D model must have a complete 3-D definition in space. The frame of reference for this space is the cartesian coordinate system, which gives us a standard mathematical reference for specifying 3-D space. There are various conventions used in reference to this system. The center of the system is called the origin. The X axis runs horizontally with positive X (X) to the right and negative X (-X) to the left of the origin. Y is the vertical axis with Y going up and -Y going down from the origin. The Z axis is the depth axis, with positive Z forward in space and the -Z going back in space. This gives us a right-handed system, which is important to know when it comes to rotating an object in space. I will go into this in more detail later.

4. Data

Each coordinate point in this space can now be defined as an (X, Y, Z) triple. Connecting these points or vertices with lines creates a wireframe view of an object. If this object is to be shaded, the surface not just the edges needs to be defined. One way of approximating this surface, is by defining a planar polygon or face. To define the face the coordinate of each vertex is listed. In a consistent order around the periphery of the polygon. For example, a triangle would be made up of three vertices and a square would be made up of four vertices. A face has two sides that need to be distinguished from each other. A consistent order in describing the vertices that make up the polygon is important to determine if the polygon is facing towards or away from the viewer. A clockwise order as seen from the observer position or the outside is used to define the front of the face. The reverse order as seen from the eyepoint would indicate that the polygon is facing away or backfacing. These backfacing polygons are identified and not displayed. [2] [4] [5]

This ordered list provides a description of the face normal. The face normal is a vector perpendicular to the plane of the face directed outward from a front-facing polygon. This face normal is used to determine hidden-surfaces and used for shading [5]. This system speeds up the hidden-surface calculation. An object is a collection of adjoining polygons. Since neighboring polygons share vertices along common edges, objects can be easily and compactly defined by listing each vertex once and polygons are defined as a list of pointers to the vertices they use. Another way to speed up calculations is to be sure that all polygons are convex. This means that a point is not closer to the center of the polygon than either of its neighbors. [2] [4] [5]

The model is a simplification. The closer the approximation of computer data to reality, the more complex. Realistic objects may become very complex, which tends to increase the computation time needed for display. Sometimes having the same object with different levels of detail can be helpful. An object displayed in the distance does not need to be as complex, and thus take as much time to display, as an object close up. There are several things to consider when planning and creating a piece of data. One thing to keep in mind, is that a polygon has to be a plane. A triangle by definition is a plane, and if there is any doubt about a surface then it should be broken down into triangular polygons. Backfaces can be a problem in debugging a piece of data, as well as a design consideration. Backfacing polygons can leave holes in an object. If the eyepoint is placed inside a box, the box would not be visible because all the polygons would be backfacing. To eliminate this problem all the polygons can be defined twice, once clockwise and once counter-clockwise. Another method is to turn off the culling of backfacing polygons for the boject. Purposely designing in backfaces allows one to play with space and the object. Another thing to keep in mind when designing data is the fact that it can be viewed from any point in space. As in sculpture, the data may need to be interesting from multiple viewpoints.

There are various methods for planning an object and picking the coordinates of the vertices. One method that is very useful is orthographic projection. This involves plotting 3-D points on a piece of 2-D graph paper using the cartesian coordinate system. The front or plan view of the object is projected onto the XY plane. A side view is projected onto the YZ plane. A top or bottom view is projected onto the XZ plane. These plans are similar to an architect's blueprints.

Folding these plans into a box with the side and top folded back provides us a model of this object in 3-D space. These plans assist in finding the coordinate of a point in each axis, X, Y, Z by marking the origin and counting spaces on the graph paper. One consideration is the location of the origin in relation to the object. Most the time the origin is in the center of the object. Then it is simply a matter of connecting points in a clockwise order to define the polygons.

Many systems have a library of pre-defined geometric models or primitives (cube, cone, sphere, pyramid, and cylinder, etc.). These primitives can be scaled, rotated, and placed in space to build new objects. This is the building block or combinatory approach to data generation. Since a sphere can become an ovoid, a cylinder can change from a long thin rod to a flat coin. Many possible forms may be created using this method. Another thing that can be done with primitives is warping. Some of the points making up a sphere can have the value of their coordinates changed. In effect, we can 'push' and 'pull' points to create a new shape. Projection is a fairly simple way to create data. It is somewhat like a cookie cutter approach. The points for a shape are plotted in the XY plane and then repeated with a changed Z value in each vertex. Solids of revolution are very useful in plotting the points for a symmetrically curved surface, such as a vase. The data is created for a 'piece of the pie' or the profile and then rotated an appropriate number of times at the correct intervals to form the entire object. Lofting is very much like defining the contours for a topographic map, where a series of cross-section of an object are defined. This method can be very simple or become very exacting. It is similar to creating a sculpture by cutting it into many thin slices and serially plotting the points. The slices are then connected and defined as polygon surfaces. [4]

Many of these methods, as well as others, have been implemented into data generation programs that are interactive, in such cases the artist is usually able to input points with a 'drawing' or digitizing tablet. Polygonal data is a linear approximation of a curved surface. Other software may have methods of creating data that can more closely represent curved surfaces. An example of these higher-ordered approximations are B-sline and Bezier patches [5].[4].

5. Object Transformations

Once various objects are defined in the computer there are several things one can do with each object to create a scene. They can be made larger or smaller or scale it. They can be moved around in space or translate it. They can be turned in space or rotate it. The manipulation of an object is performed on a copy or instance of the original. In this way one can have several instances of the same original object and manipulate each individually. Generally the copy can be given a name so that it is easier to keep track of the many objects in a scene. These manipulations of the object we call transformation. These transformations can be expressed mathematically in matrix algebra by a single entity called the transformation matrix. Complex or several transformations of an object can combined or concatenated into one matrix [2]. Matrices are usually invisible to the user. It is important to only understand what they do.

5.1. Scaling

When scaling each coordinate is multiplied by a scale factor [2]. One is free to scale independently in each axis to change the shape or proportions of an object. Objects can also be scaled equally in each axis and maintain a larger or smaller version of the same object. In this way it is very easy to turn a cube into a board by scaling it up in X by 4 and scaling it down in Y by .50. For example here are the coordinate points for a cube:

	X	Y	Z
1.	1	1	1
2.	-1	1	1
3.	-1	-1	1
4.	1	-1	1
5.	1	1	-1
6.	-1	1	-1
7.	-1	-1	-1
8.	1	-1	-1

Each point in the X column would be multiplied by 4 (4 x 1 = 4 and -4 x 1 = -4). Each point in the Y column would be multiplied by .5 (1 x .5 = .5 and -1 x .5 = -.5). The Z coordinates would remain the same. Thus we have changed an object with the outer limits or bounding box of -1 to 1 in X, -1 to 1 in Y, and -1 to 1 in Z to -4 to 4 in X, -.5 to .5 in Y and -1 to 1 in Z.

Another thing to keep in mind in designing data is the location of the origin of the object. If an object is designed with coordinate points with a zero value in any axis then any multiplication of zero is still zero. This can be used creatively if controlled, but can be difficult to control because it may not only scale an object it can also move it. The scaling can be used very effectively for instance in animation to incrementally scale an object down to the point where it disappears or up to the point where it takes over the screen. In fact scaling an object by a negative number can turn it inside out and reverse the order of the polygon description. If an object is designed for this it can actually become another object on the inside.

5.2. Translation

In translation a factor is either added or subtracted from each coordinate to move it around in space [2] [5]. As with scaling this can be done independently along each axis or in more than one to place it where desired in space. In the example of data for the cube listed before, the object has its origin at 0 X 0 Y 0 Z (0,0,0). If this object is placed at 0,0,0 it will not have moved. If it is placed at -2, 1, -3 values

will have been added or subtracted respectively from each coordinate of the object. This cube will have just moved to the left in X up in Y and back in Z. A shorthand method for thinking about this that the origin 0,0,0 of object is actually placed at -2, 1, -3. Using this method it is fairly easy to figure out where the object is placed.

Translations are very useful in organizing and placing things in space. Incrementally placing the same object at various intervals can be used to build more complex images, such as a steps or bars on a crib. In animation this can be used to simulate motion. If one displays a box, takes a picture of it with a movie camera (stop motion), then moves the box a little more and takes another picture of this frame, we have animation. Translation is very important in animation. By varying the distance between moves one can either speed up or slow down the motion.

5.3. Rotation

Rotations are mathematically more complex. They are done through matrix multiplication using sines and cosines [2] [5]. Artists need to only understand how to control rotations. In translation a positive number in X moves the object to the right and a negative number moves it to the left. In scale large numbers expand the object and small numbers contract it. In rotation one has to think of the direction of the rotation. As stated previously, these examples will use, a right-handed space. With the right hand one grasps the axis with the thumb pointing towards the positive end of the axis, your fingers will curl in the direction of a positive rotation. When looking down an axis from the positive end towards the origin a positive rotation will be counter-clockwise [2]. Rotations are usually specified by the axis of rotation and degrees of rotation either positive or negative for the direction.

A simple rotation is a rotation around a single axis in the coordinate system [2]. This is a rotation on the origin of the object in a specific axis. The order of rotation is important. This concept is not immediately obvious but must be understood. The reader can demonstrate this principle by holding an object and performing these operations. Rotate an object on its X axis, then on its Y axis. Then repeat the same operation in reverse order. You should see that the resulting orientations in space are different due to the order of rotations.

A concatenated rotation is a rotation about more than one axis at a time [2]. If an object is rotated in X and then in Y, a concatenated rotation would be rotating it in X and Y at the same time. The order of the rotations are still important even when rotations are concatenated. Keeping track of rotations can become very complex and requires skill and a good sense of spacial orientation.

Rotations off the origin of the object are different from on-origin rotations. This can be thought of in terms of the solar system. An on-origin rotation is like the earth rotating on its own axis, and an off-origin rotation corresponds to the earth orbiting around the sun. It is important to realize that several rotations on-origin and off-origin can be going on at once increasing the complexity. A common error is to rotate an object so that it moves completely off the display screen. The beginner is often unable to understand why the object can't be seen.

Rotations, like scaling and translating, can be used effectively in animation. By incrementally rotating an object it can appear to spin, tumble or roll. The speed of this motion depends on the size of the increments. For example, if it only rotates by .5 degrees per frame it will appear to rotate very slowly.

6. View Transformation

Once all the objects are arranged as desired in the scene, an eyepoint to view the scene must be selected. The eyepoint is placed by locating it in the coordinate system through an X, Y, Z) triple. The point in space at which one is looking must also be specified. This point is the center of interest and is placed by defining its X, Y, Z position. These two points in space create a line of sight. The viewing angle, which is the angle on either side of this line makes up the viewing pyramid. The eyepoint is at the apex of the pyramid. Anything outside this view pyramid is clipped and not displayed. A view angle of 90 degrees indicates a 45 degree view on either side of the line of sight. Depending on the system, this angle is either fixed or can be changed, but it is important to know. Setting the viewing angle is similar to chaning the focal length of a camera lens, in that the size of the angle of view controls how much of a scene will be seen in the display. Similarly, a long focal length telephoto lens gives a narrow field of view, while a short focal length wide-angle lens, of course, gives a very wide field of view.

Once this view pyramid is defined, all the coordinates of the objects in the scene are transformed to eye coordinates. The view that appears on the display device seems to be centered along the specified line of sight. These new X Y Z coordinates of each of the objects in the scene are projected onto a 2-D picture plane by one of several known mathematical techniques for generating perspective, orthographic, or other kinds of projections. The perspective projection is the one we are most familiar with, because it remarkably is similar to the photographic image. In any case, the 3 X Y Z coordinates of the objects have been transformed into eye space, and then into the two dimensional X and Y coordinates of the objects have been transformed into eye space, and then into the two dimensional X and Y coordinates of the display screen. Thus the 3-D eye space is transformed to a 2-D screen space and completed the view transformation [2] [4] [5].

The eyepoint and center of interest can be used in animation to simulate filming conventions or techniques, such as cuts, pans, dolly, trucking and crane shots. By changing the view angle,

a wide angle or telephoto lens can be simulated. One unique property of computer graphics is the ability to arbitrarily move your 'camera' or use views impossible in other medium.

7. Color, Light and Shading

The color raster display device has individual video input of red, green, and blue (R G B). The three guns excite the phosphor dots which make up each picture element or pixel. Each gun is individually controlled for intensity. The eye blends these dots of R G B into the visible colors of the spectrum [4]. This color phenomenon was used by the Pointillist painters. The light reflection model that describes both the color and distribution of reflected light in computer graphics is similar to the way some artists approach a painting. The illumination and the color of an object are painted separately. The entire painting is done achromatically in white, greys and black creating the shading gradients, then local color is applied with glazes [1].

Now that we have calculated the visible surfaces of our image, we need to compute the correct intensity value for each pixel in the shaded image. A light reflection model that describes both the color and the distribution of reflected light. This intensity value is determined by the position of the light source, the polygon orientation and the surface properties of the object. The shading gradient is determined by the angle of incidence of the light source to the normal vector, which is perpendicular to the plane of each polygon [4]. The closer this angle is to zero the brighter the face, and conversely the greater the angle the closer to black the face will be. These value gradients are usually stored in a color look-up table. If these tables can be accessed the artist can play with reversing light and dark, 'bumping' the dark values, and change the hue of highlights and shadows, etc.

Now that each polygon has a value, the image can be displayed. Each polygon would be seen as faceted, or one value for each polygon. Gouraud and Phong shading models create more realistic images of curved surfaces of smooth shading, even if the data is polygonal [4] [5]. Gouraud shading is a fast approximation of smooth shading. The normal is calculated for each vertices and the intensity is then linearly interpolated across the surface of the polygon. Phong shading is a closer approximation to reality. This algorithm computes a normal at each pixel to obtain the intensity, but is computationally more expensive. Ray tracing is the most exact model, it takes into consideration highlights, shading, cast shadows, and reflections, of course it takes a very long time to compute an image [4]. These shading algorithms are invisible to the user. The artist must only understand the principles so that he may achieve the desired effect by placing the light source and coloring their objects. The point light source is placed at an X Y Z coordinate in space in relation to the position of the other objects in the scene. Depending on the system, the artist may have access to more than one light source. He may even be able to scale and color the light source. The point light source is usually thought of as a sphere of light with the scale factor a radius. Thelight model for directed light as modelled after stage lighting could be very exciting, but is not yet used.

Color in computer graphics is made up strictly of the three primary colors red, greem and blue (R G B), due to the limitations of the raster display. This system is modeled after the light primaries and is an additive color system. Colors are specified numerically with a discrete intensity value for each component. Colors can be specified in two ways. In the H L S (hue, lightness, and saturation) model, hue is specified as a given angle of a circle, lightness is from black to white, and saturation is used to specify the degree the color differs from grey.

The R G B system is modeled after a color cube with black at 0,0,0, and white at 1,1,1. White is made up of full intensity of each primary and black is the absence of any color. This color can be more readily visualized by creating a color wheel with red, green and blue as primaries and yellow, cyan, and magenta as intermediary colors.

	R	G	B
red	1	0	0
yellow	1	1	0
green	0	1	0
cyan	0	1	1
blue	0	0	1
magenta	1	0	0
white	1	1	1
black	0	0	0

The higher the number the brighter the color. A bright blue would have the value of 1,0,0 and a darker blue would have a value of .2,0,0. The artist has several adjustments to make in using color on a computer. Colors are specified with numbers, and additive color systems is different from substractive color system which the artist is used to. A colored light can drastically change the hue of a scene. The artist has to practice using a particular system and a particular display device to be able to anticipate hue and saturation of a color mixture.

The possibilities for artists in computer graphics are endless. Presently the design of hardware and software is geared to science and technology. The artist needs to become more familiar with computer graphics and form a partnership with computer scientists. Through collaboration of these two fields, computer graphics can become a more natural, dynamic and artistic medium.

References

1. Arnheine, R., Art and Visual Perception, University of California Press (1974).

2. Crow, F.C., Three-Dimensional Computer

Graphics, Part I, Byte, Vol. 6, (3), p. 54 March, 1981.

3. Knobler, N., The Visual Dialogue, Holt, Rinehart, and Winston,Inc. (1966).

4. Greenberg, D.P., An Overview of Computer Graphics, Addison- Wesley (1982).

5. Newman, W. and Sproull, R., Principles of Interactive Computer Graphics, 2nd Edition, McGraw-Hill, New York, (1979).

6. Ocvirk, O., Bone, R., Stinson, R., and Wigg, P., Art Fundamentals Theory and Practice, Wm. C. Brown Co. Publishers (1975).

A PROFESSIONAL QUALITY DIGITAL AUDIO PERIPHERAL FOR SMALL COMPUTERS

by Hal Chamberlin

Micro Technology Unlimited
2806 Hillsborough St., Raleigh, North Carolina

ABSTRACT

During the past couple of years the development of low-cost digital audio systems has proceeded at an astonishing pace. Already announced, and in many cases available to consumers, are encoders/decoders which turn a video cassette recorder into a digital audio recorder, and "compact disk" digital audio playback systems. The ability to offer such devices at reasonable cost (under $2000) is due primarily to advances in semiconductor processing (for the 16 bit DACs and ADCs involved) and willingness of large, established audio equipment suppliers to invest in LSI controller ICs to handle the difficult synchronization and error control logic involved. While these developments are a boon for the critical audiophile, they have not so far been exploited in the design of a professional quality digital audio computer peripheral suitable for use in small systems. This paper describes the design features of such a device which is flexible, is easily interfaced to any small computer, and is much lower in cost than previous units intended for use with minicomputers or mainframes.

Introduction

Digital recording and playback of audio signals is currently the hottest topic in audio, both in the professional recording industry and in high-end consumer products. It is difficult now to find an audio journal or magazine without at least one article or product review on digital audio equipment. The advantages of using digital techniques in what has been the weakest link in the audio chain are indisputable (although a handful of ill-informed critics nevertheless try to dispute them). This fact, coupled with recent advances in IC and laser disk technology, has encouraged established audio equipment manufacturers to make the investment necessary to manufacture digital audio equipment at low cost and in large volumes.

While these developments promise to revolutionize the quality of recorded and reproduced sound, they so far have not been applied to any great extent in synthesis and processing of sound material with small computers. When one realizes that sound in digital form is completely accessable to manipulation by computer (just as tape recording made sound accessable to manipulation by hand 35 years ago), one also realizes that a whole new world of creative possibilities has been opened up. With today's powerful personal computers, inexpensive mass-storage peripherals, and a suitable digital encoding/decoding peripheral, even small institutions and individuals with limited expertise can now explore this world which heretofore has been limited to large organizations.

To date, computer processing of digital audio has been performed on three fairly distinct "planes" of sophistication. On the highest plane are the professional recording studio equipment suppliers who design, manufacture, and program the entire system including a special purpose computer and cost-is-no-object A-to-D and D-to-A converters. The cost of such systems is accordingly high while their flexibility is usually limited to the job at hand such as equalization (tone control) and multi-track emulation.

Next down are large research organizations such as music departments at major universities. They typically use a campus mainframe or large departmental mini-computer and either buy the A-to-D and D-to-A conversion equipment off-the-shelf (if something suitable can be found), or more frequently design and build one unit in-house. While cost is usually an important factor (relative to the commercial plane), these computer music installations nevertheless represent several tens of thousands of dollars worth of equipment.

On the lowest plane are small organizations and individuals using personal computers and relatively crude commercial or home-made conversion equipment. Much has been accomplished on this plane in the short history of personal computers (1, 3, 4) but the lack of reasonably priced, high quality conversion equipment has restricted serious artistic application of the results. While the participants on this plane may be fully capable of designing and implementing audio signal analysis, processing, and synthesis software; the expertise needed to integrate an IC data converter into a high quality conversion peripheral is lacking. It is this need that the device to be described is designed to fullfill.

Desirable Characteristics

Since this project was untertaken as a product design to be manufactured by a small company, there was the initial task of determining what features were desired by potential customers. Of course there is always the issue of cost since a unit that incorporates every desired feature at the highest possible accuracy level would clearly be too expensive for the intended market. Nevertheless the final design is remarkably free of compromises.

The first issue that seems to come to any prospective user's mind is sampling rate. While data throughput considerations in the host computer often limit the attainable system sample rate (2), nearly everyone felt that the conversion peripheral should have as high a sample

CH1930-7/83/0000/0074$01.00 © 1983 IEEE

rate capability as possible, at least up to 45-50KHz in order to cover the accepted 20Hz-20KHz audio range. Frequently noted were requests to provide the "commercial standard" sampling rate of 44.056KHz and the "professional standard" of 48.000KHz with zero error. Besides these, a continuously programmable rate was also desired with particular emphasis on 1/2 the commercial rate (22.028KHz), 1/2 the professional rate (24.000KHz), and typical "speech rates" of 8 and 10KHz.

Equally as important as sampling rate was the issue of wordlength or the number of bits in the converter. Again, almost reflexively, 16 bits was desired. The desire persisted even after extended discussions of what was likely to be audible when an analog recording device was in the chain (as it would likely be to hold the initial input or receive the final output) and the impact on cost, particularly of A-to-D conversion. In many cases, the ability to operate with shorter words, such as 12 bits or even 8 bits, when acceptable was desired to minimize data storage requirements.

Closely associated with the sampling frequency is the issue of anti-alias filters. Since low noise, sharp cutoff filters are difficult to design and build and are expensive to buy off-the-shelf, most respondants preferred that they be supplied along with the converter. An additional problem with filters is that high quality with selectable or programmable cutoff frequency entails a very high cost. Conversely, it was undesirable to closely integrate the filter with the converter and thus force selection of a fixed sample rate when the unit is built. The use of fixed-frequency plug-in filter boards was found to be an acceptable compromise in most cases.

Both analog-to-digital operation (recording) and digital-to-analog operation (playback) were desired. When the considerable cost increment to provide 16 bit A-to-D conversion was discussed, its acceptance as an extra-cost optional feature greatly increased. Sometimes simultaneous A-to-D and D-to-A operation was also desired for applications such as real-time digital signal processing. However, after discussing the difficulties and expense involved, most indicated that the use of two lower cost units, one A-to-D only and one D-to-A only, would be acceptable. Most everyone preferred at least the option of stereo. Those most interested in analysis were the least likely to require stereo while those primarily interested in synthesis universally wanted stereo. Occasionally, even more than two channels was desired for non-audio data acquisition applications.

The question of interface technique to the host computer proved to be especially sticky. The first experimental conversion peripheral constructed in 1979 was a stand-alone device interfaced via a parallel I/O port to the host. There were several design problems related to the lack of synchronization between host-converter data transfers and the sample rate but that method did have the advantage of being non computer specific. The second experimental converter (2) was implemented as a plug-in bus peripheral for the MTU-130 computer which solved those problems but added some others such as digital noise pickup from adjacent boards. With the current proliferation of powerful 16 bit systems it was clear that a successful unit must again be non-computer specific. Additionally, 16 bit operation essentially mandates packaging in a separate, shielded box (the experimental units were 12 bits). Also, it was deemed desriable in some cases to provide for DMA transfer of data through a DMA port as well as programmed transfer through a parallel I/O port.

The last issue was buffer size. Potential users weren't overly concerned with the buffer as long as it was possible to perform continuous digital recording and playback to or from disk. Since sample rates, particularly with 16 bit samples, have to be rock steady, at least one buffer register is always required. The earlier prototypes extended this minimum to a more generous 256 and 1024 samples respectively which simplified record/playback programming by allowing up to several milliseconds of host inattention before data loss (recording) or sound gaps (playback) is experienced. However, when disks are used for digital data storage, considerably more buffering is required (2). This is usually provided by the host computer's main memory and a fairly sophisticated buffer management program. With today's low cost memory ICs, it should be feasible to provide this very large buffer right in the converter and thus simplify programming even further.

Given these market inputs then, the design task is how to best satisfy them at a reasonable cost. Additionally, to be a feasible product for a small manufacturer, the design should only use standard components. The following sections briefly describe the reasoning behind design decisions in each of these areas and more fully describe the final unit.

Converter Selection

Great strides in low-cost monolithic and hybrid A-to-D and D-to-A converters have been made in the past 2-3 years. In particular, units designed for audio have appeared to meet the projected need in consumer digital audio devices. This is significant because in audio applications absolute accuracy (percent of full scale) is not very important while relative accuracy (linearity) is quite important. While both are difficult to achieve at the 16 bit level, removing concern over absolute accuracy has a very favorable effect on cost.

When selecting the D-to-A converter, it turned out that quite a variety were available. This is probably due to the expected sizable market for "compact disk" playback equipment. Major contenders in the US merchant market include the Hybrid Systems HS3160, Intersil ICL7134, Analog Devices AD7546 and DAC9331, Harris HI-DAC16, Analogic MP1926, and Burr-Brown PCM51. There are undoubtedly others available in Japan or under license in large quantities.

The HS3160 is a CMOS device which uses a segmented technique with 16 segments of 4096 steps each. Although rated for 1.5uS current output settling time, the large output capacitance characteristic of CMOS converters makes the practical value several times longer. Cost is about $50. The DAC9331 is also a CMOS device but uses an R-2R ladder for identical differential and integral linearity. Its cost is about $80. The AD7546 too is CMOS and uses segmentation to attain 16 bit monotonicity. In practice it needs 3 external op-amps and the total settling time is about twice the amplifier settling time or typically 10uS. One unique feature is a built-in deglitching switch. It costs about $45. The ICL7134 is unique in that an on-chip PROM is programmed after manufacture to correct for nonlinearities. The PCM51 is a bipolar R-2R unit with a settling time of just 350NS current output. It costs only about $25. The MP1926 is probably the best unit available specwise (it uses sign-magnitude architecture) but its cost

of nearly $200 limits use to the very best professional equipment.

In a stereo system there is the decision between using two DACs, one for each channel, and using a single DAC which flip-flops between channels. The former places the least demand on converter (and other circuits) speed while the latter should be lower in cost. Another consideration is that the flip-flop arrangement introduces a slight differential delay between the left and right channels. At 50KHz this would amount to 10uS which is about 1/8 inch in air and so was dismissed as being insignificant.

Given these considerations, the Burr-Brown PCM51-V was chosen as the best all-around value. This unit has an on-chip reference and an on-chip amplifier to provide a +-5 or +-10 volt output voltage with a 5uS settling time. Spec sheets just received from Burr-Brown indicate that a new model, the PCM-53, is now available at a slightly lower cost and with slightly better low-level distortion specs (i.e., better zero-crossing linearity). This was achieved by switching from the brute-force R-2R ladder used in the PCM50/51 to a segmented architecture. Since it is essentially plug-compatible with the older unit, it will probably be used when production starts.

The range of possibilities for 16 bit A-to-D converters was much more restricted. Since the potential market for consumer digital audio recorders is much smaller than compact disk players, there is less incentive to develop very low-cost A-to-D converters. Most units evaluated were relatively slow at 30uS (30KHz maximum sample rate, single channel) or very expensive large potted modules; all intended for data acquisition use. One designed-for-audio standout was the Burr-Brown PCM75 which has a 17uS conversion time (8uS with reduced specs), small 1"x2" size and acceptable cost of $250. Another possibility was constructing an ADC from one of the Burr-Brown DACs, a successive approximation register, and a high speed comparator. Although likely to be much lower in cost, the thought of shielding a comparator with 150uV sensitivity and 100NS speed made the integrated unit seem much more attractive.

Note that at 17uS the PCM75 is not fast enough to perform 50KHz stereo digitizing at full 16 bit linearity with just one unit. According to the spec sheet however, one looses just about one bit of linearity when sped up to 10uS. By using a separate sample-and-hold for each channel, 10uS is fast enough for stereo at 50KHz and the slight loss in linearity was judged to be an acceptable tradeoff for the substantial reduction in cost from two units.

Sample-and-Hold and Deglitcher

One of the secrets of successful audio D-to-A conversion is the sample-and-hold circuit used for deglitching. Monolithic SAH circuits are woefully inadequate (2) while hybrid modules are expensive and usually undergo distortion inducing slewing when switching from hold to sample modes. Analogic does make a device designed specifically for audio reconstruction but again its cost is beyond acceptable limits. The only alternative seems to be constructing the deglitcher from descrete components.

In the past the author would have used Signetics SD211 DMOS transistors or an SD5001 DMOS transistor array for the analog switches due to their extremely high speed (10NS) and very low gate-to-drain capacitance (0.2pF). Unfortunately these devices have been discontinued. Harris however makes a nice, fast (30NS) CMOS integrated quad switch-driver array known as the HI201HS. For stereo deglitching, four switches are assembled into a non-slewing sample-and-hold arrangement illustrated in figure 1.

The sample-and-hold circuit used ahead of the A-to-D converter has a different set of requirements. Slewing when going from hold to sample modes is not harmful but a fast, uniform transition from sample to hold is important since the input signal is changing. The same HI201HS switches proved adequate in that application as well. The stereo sample-and-hold arrangement illustrated in figure 2 is not very sophisticated from a data acquisition point of view (the droop rate is fairly poor, there is a noticable hold-step, and no attempt is made to cancel amplifier offsets) but it does hold the signal long enough to be digitized and introduces very little excess distortion over that of the ADC itself.

Sample Rate Clock

Normally, the sample rate clock would be a simple, straightforward crystal oscillator and programmable divider. However, the desire to provide both the commercial standard (44.056KHz) and professional standard (48.000KHz) sample rates without error proved to be somewhat troublesome.

First it was necessary to determine how the 44.056KHz number was arrived at. This was initially done empirically and later confirmed by consulting audio industry trade journals. It seems that 44.056KHz is equal to 4/325 of the NTSC color subcarrier frequency of 3,579,545Hz. How's that for an arbitrary choice, at least from an audio point of view! To actually generate this frequency, one would divide the output of a 14.31818MHz crystal oscillator by 325.

Next, some common factor analysis was performed to find a crystal frequency that when divided by integer values could generate a whole host of "round" sampling frequencies without error. In particular, accurate rates of 8, 10, 12, 12.5, 15, 20, 24, 25, 30, 40, 48, and 50KHz were desired. The lowest reasonable frequency that satisfied all of these was determined to be 14.4MHz.

Since the least common multiple of 14.31818MHz and 14.4MHz is somewhere up in the light frequency range, the final sample rate clock uses two crystal oscillators to generate two basis frequencies. When selecting a sample rate, the desired basis frequency must also be selected. With a 12 bit programmable divider, the lowest sample rate that may be selected is about 3.5KHz. For additional flexibility, an external sample rate clock can also be selected.

Sample Buffer

As mentioned earlier, having a very large first-in-first-out buffer can vastly simplify programming of the host system for continuous transfer of data between disks (both floppy and rigid) and the conversion peripheral. With a large enough buffer it should even be possible to do this on a system that uses a programmed I/O disk controller (as opposed to a DMA disk controller).

Since currently available FIFO buffer chips are far too small to consider (at most 128 locations by 8 bits), the buffer had to be implemented as a dual-port random-access

memory with two address pointers (counters), one to steer data going into the memory and one to select data coming out. There were essentially three choices for the actual RAM chips themselves: 16K static (2Kx8 6116 type), 16K dynamic (16Kx1 4116 type), and 64K dynamic (64Kx1 4164 type). Since a truly large buffer was desired and the cost of 64K dynamic RAMs is down around $5, they were chosen. Using 16 of them would have created a 64Kx16 sample buffer, enough for a second or more of sound at all but the highest rates. This was scaled back to 64Kx8 in the final unit and "page mode" cycles used to access the high and low byte of each sample in quick succession.

When actually programming data flow to or from a device through a FIFO buffer, one needs to know when the buffer is empty and when it is full. To avoid having to check its status after every sample is read or written, you would also like to know just how full or empty it is at the moment. With such a "fullness register", buffer status need be checked only every block of perhaps 1024 samples. One could also determine whether the buffer status is such that a retry could be attempted in the event of a disk error.

During the circuit design, however, it turned out that detecting the exactly full and exactly empty conditions was very costly in terms of real-estate on the board. Thus while status register bit positions and connector pins have been reserved for these functions, the final buffer design has a "fullness resolution" of only 1/8 of its capacity. What this means is that at the very end of a record or playback session, the last 0-4095 samples may be lost. Since this may represent as little as 50 milliseconds of sound and it is normal to pad the end of segments with silence, this is not seen as a major drawback.

Computer Interface

Of all of the subsystems, implementation of the computer interface had the widest variety of choice within the constraint that it be reasonably easy to interface to most computers. It is pretty clear that the interface should be parallel although somebody could probably make a case for 1M baud serial instead. Given that it is indeed parallel, it is also clear that it should be 8 bits wide for use with 8 bit systems even though data transfer efficiency is sacrificed when used with a true 16 bit host.

Some standard parallel interface structures were then examined to see if any were suitable for this rather unusual application. Centronics parallel was rejected because it is unidirectional (output only). An IEEE-488 interface would have worked, but for most computers an IEEE adapter board is quite expensive. Also many adapters are too slow to be used at the higher sample rates (up to 200K bytes per second needed) even though the IEEE-488 bus is rated for operation at up to 1M byte per second. The SCSI interface often used for remote rigid disk controllers was also a possibility but it seemed needlessly complex for the task at hand.

The approach finally taken was to imagine the entire conversion peripheral as an LSI integrated circuit chip and equip it with the sort of interface signals and philosophy that a chip designer might use. In many ways the interface resembles that presented by the NEC uPD765 floppy disk controller chip used in several computers including the MTU-130/140 and the IBM PC. These interface signals are summarized below (a leading - denotes inversion):

1. 8 data lines, bidirectional, positive true.

2. 1 address line, input, 0 selects the control/status register and 1 selects the data register.

3. -READ/WRITE, input, selects whether the strobe will read from the selected register or write into the selected register.

4. -STROBE, input, enables read data from the selected register onto the data lines or strobes write data into the selected register on its trailing edge.

5. -RESET, input, instantly stops operation, clears the buffer, and initializes the unit to an idle state.

6. -NEAR FULL, output, asserted when the buffer becomes 3/4 or more full.

7. -NEAR EMPTY, output, asserted when the buffer becomes less than 1/4 full.

8. -DATA REGISTER READY, output, negated following a strobe directed to the Data Register for the time required to dispose of written data or retrieve the next byte of read data. The not ready time averages approximately 1 uS.

9. -DEVICE SELECT, input, all inputs are ignored and the data lines float when negated.

A jumper is provided to force read data onto the data lines whenever -READ/WRITE selects read regardless of the state of -STROBE. The strobe is still necessary however to "tell" the device that data has been read. For simplified interfacing to some systems, jumpers are also provided to redefine -READ/WRITE and -STROBE as follows:

3. -READ ENABLE, enables read data from the selected register onto the data lines.

4. -WRITE ENABLE, strobes write data into the selected register on its trailing edge.

This interface can be easily driven by a dual 8 bit parallel port such as that provided by a 6820 PIA or 6522 VIA or 8255 PPI parallel interface chip. One of the 8 bit ports is connected to the data lines and must be easily switchable between output and input functions. The other port connects to the 8 control and status lines and should be capable of mixed (although constant) directionality with 5 outputs and 3 inputs. By suitable program manipulation of these two ports, all device functions can be controlled. It is also possible to utilize the various strobe outputs and edge sense inputs of these chips for some of the interface functions to simplify programming. Note that the data transfer rate through such a hookup may be limited by host CPU speed to a value less than the 200K bytes per second necessary to run stereo at a 50KHz sample rate.

Since the interface is "chip-like", it can in most cases also be connected to the host as a direct bus addressed peripheral. All that is needed for such a connection is an 8 bit transceiver (such as a 74LS245) between the host data bus and the converter data lines, a buffer between the host's least significant address line and the converter's address line, additional buffers for the -READ/WRITE and -STROBE signals (or -READ ENABLE and -WRITE ENABLE) and -RESET, and an address decoder. The remaining signals can be ignored. The interface can also be connected to a DMA

port for fast, automatic data transfer. The -NEAR EMPTY, -NEAR FULL, and -DATA REGISTER READY signals would be used to "throttle" the DMA rate and thus prevent overruns or underruns from occurring.

Programming

Operation of the converter is through a number of 8 bit registers. Each register is selected by a unique combination of the -READ/WRITE interface signal, the interface address line, and bits 0 and 1 of the control register. Note that some registers are either read-only or write-only.

The converter is placed into various operating modes by writing different bit patterns into the Control Register. The Control Register is a write-only register and is selected when the interface address line is low and a write operation of performed. Reset forces the Control Register to all zeroes. The bit assignments of the control register are shown below:

CONTROL REGISTER

Bit 7 0 = Pause, stop data flow between the converter and the buffer but retain buffer contents. 1 = Run, start data flow between the buffer and the converter.

Bit 6 0 = clear the buffer to empty status and zero the DAC outputs. 1=permit normal buffer and ADC/DAC operation.

Bit 5 0 = monophonic mode (channel A), 1 = stereo mode (channels A and B alternate with channel A first).

Bit 4 0 = DAC mode, samples are withdrawn from the buffer and converted to audio output. 1 = ADC mode, the audio input is digitized and the samples are written into the buffer.

Bits 3,2 00 = Reset the format converter.
01 = Select 8 bit companded format.
10 = Select 12 bit companded format.
11 = Select 16 bit linear format.

Bits 1,0 00 = The Data Register is connected to the buffer.
01 = The Data Register is connected to the Scale Factor Register.
10 = The Data Register is connected to Sample Rate Register 1.
11 = The Data Register is connected to Sample Rate Register 2.

The Status Register indicates the status of the buffer and the two external user digital inputs. The Status Register is read-only and is selected when the interface address line is low and a read operation is performed. The Status Register may be read at any time with no effect on operation. Bit assignments of the Status Register are shown below:

STATUS REGISTER

Bit 7 Not used, always zero. Reserved to indicate that a buffer overrun or underrun error has occurred.

Bit 6 Not used, always zero. Reserved to indicate that the buffer is exactly empty.

Bit 5 Reflects the state of external user digital input B.

Bit 4 Reflects the state of external user digital input A.

Bits 0-3 Indicate the degree of buffer fullness as follows:
0000 = less than 1/16 full
0001 = less than 1/8 full but at least 1 sample
0010 = less than 3/16 full but at least 1/16 full
....
1110 = less than 15/16 full but at least 13/16 full
1111 = at least 7/8 full

Note that although the resolution of the buffer fullness measure is 1/16, there is an added uncertainty of 1/16 which gives a useful resolution of 1/8. Addition of the exactly empty and exactly full detection logic would also eliminate this uncertainty. Depending on the interface method and programming preferences, either that Status Register or the 1/4 and 3/4 interface signals may be used to monitor the buffer status.

According to bits 0 and 1 of the Control Register, the Data Register may actually be connected to one of four internal registers: the FIFO buffer, the Scale Factor Register, and one of two Sample Rate Registers.

Data in the buffer always consists of 16 bit twos-complement values. Data is exchanged between the buffer and the Data Register through a format translator. Depending on the data format selected, 1, 1.5, or 2 bytes may be required to represent a sample. The format translator is "incremented" after each read or write to the buffer through the Data Register. The format translator may be reset to its initial state at any time to insure synchronization with the byte oriented data stream to or from the host. It must be reset first when changing to a different format. The buffer is read-only when in ADC mode and is write-only when in DAC mode.

The Scale Factor Register is used with the companded data formats to scale the reduced resolution sample values to take advantage of the 16 bit dynamic range of the A-to-D and D-to-A converters. During output to the D-to-A converter the lower 4 bits of this register determine the number of sign-extended right shifts that samples from the host should undergo when transferred into the buffer. During input from the A-to-D converter, they determine the number of left shifts that samples from the buffer should undergo when transferred to the host. In stereo mode the same scale factor applies to both channels. The high 4 bits are used to control 4 externally available user digital outputs.

Two registers are used to specify the sampling frequency. Sample Rate Register 1 holds the lower 8 bits of a 12 bit divisor while Register 2 holds the upper 4 divisor bits in its lower half. The actual divisor used is the twos complement of the value written into these 12 bits. The remaining 4 bits of Sample Rate Register 2 are as follows:

Bit 7 Not used.

Bit 6 ADC speed control, 0 = 17uS (full specs), 1 = 10uS (reduced specs)

Bit 5 Sample clock source, 0 = internal clock, 1 = external clock.

Bit 4 Basis frequency for internal clock, 0 = 14.400MHz, 1 = 14.31818MHz

The host is responsible for setting the ADC speed to a value compatible with the sample rate and mono/stereo selection. These are write-only registers.

Programming Examples

Programming details are highly dependent on the host computer, actual interface technique, and the type of mass storage device used. Figures 3 and 4 however show general flowcharts for recording and playback respectively. The blocks with dotted outlines in figure 4 are needed only if the 8 or 12 bit mode is used and the sound file format supports the block gain control feature for increased dynamic range.

It should go without saying that these core data transfer routines must be written in the assembly language of the chosen host. However it is entirely possible that these would be the only assembly language routines in an overall synthesis or analysis software system.

Filter Modules

Description of the filter modules has been deferred because they are independent of the main unit and in many cases may need to be optimized for the specific application. Each plug-in module consists of a front panel which measures 1" by 3" and a printed circuit board that measures 2" by 3". Connection to the user's audio equipment is made through a jack mounted to the module's panel. The modules are easily changed although power must be turned off first.

The standard A-to-D filter is a 7th order elliptical design that uses active negative impedance converters to replace inductors (5, 6). The general circuit topology for this type of filter is shown in figure 5. For the component values shown, the cutoff frequency is 9.76KHz, the pass-band ripple is .28dB, and the minimum stop-band attenuation is 79.7dB. The attenuation reaches maximum at 1.556 times the cutoff frequency. For A-to-D use in analysis, this point should be put at 1/2 the sample rate, therefore the cutoff frequency should be 32% of the sample rate. For A-to-D use in record-playback applications, the Nyquist frequency can be put midway in the transition band instead thus the cutoff can be raised to 39% of the sample rate.

The standard D-to-A filter is also a 7th order elliptical design with the same characteristics as the D-to-A filter. However, since the very high frequency content of synthesized material can be controlled and it does not even exist when playing back material recorded through the A-to-D filter, the cutoff frequency can be set at 39% of the sample rate. To be assured of less than -80dB alias distortion with this setting, there should be no attempt to synthesize a frequency higher than this. The D-to-A filter also has an additional 2-pole Chebyshev section to correct for the normal SinX/X rolloff of the deglitching sample-and-hold circuit.

Both filters are built using type TL072 dual low noise, low distortion, FET op-amps. Precision (1%) capacitors and resistors are used to avoid the need for tuning. Using negative impedance converters in the design allows all of the capacitors in the filter to be the same value although the resistor values then become all scrambled up. This is preferable however since 1% resistors are much easier to purchase that 1% capacitors are. Modifying the filter design for a different cutoff frequency then is a simple matter of changing all of the capacitors by the ratio of the old frequency to the new frequency.

Conclusion

While the A-to-D and D-to-A peripheral is really a minor part of a computer music system when software is considered, it is the part that causes the most trouble for a small organization or individual programmer/artist. It is believed that the conversion peripheral design outlined here will prove to have sufficient capability to satisfy virtually any digital audio conversion task at a cost that, while not trivial, is a significant improvement over earlier minicomputer oriented designs or in-house engineering.

References

1. H. Chamberlin, "A Sampling of Techniques for Computer Performance of Music", BYTE, Sept. 1977.

2. H. Chamberlin, "Delayed Playback Music Synthesis Using Small Computers", p. 27 Proceedings of the IEEE Symposium on Small Computers in the Arts, Nov. 20-22, 1981.

3. F. H. Covitz and A. C. Ashcraft, "Analysis and Generation of Complex Sounds Using Small Computers", p. 33, Proceedings of the IEEE Symposium on Small Computers in the Arts, Nov. 20-22, 1981.

4. A. C. Ashcraft et. al., "Noise in Real Time Digital Sound Generation" p. 5, Proceedings 2nd Symposium on Small Computers in the Arts, Oct. 15-17, 1981.

5. A. D. Delagrange, "Design Active Elliptic Filters with a 4-function Calculator", p. 135, EDN, March 3, 1982.

6. Anatol I. Zverev, Handbook of Filter Synthesis, p. 284, John Wiley and Sons, New York, 1967.

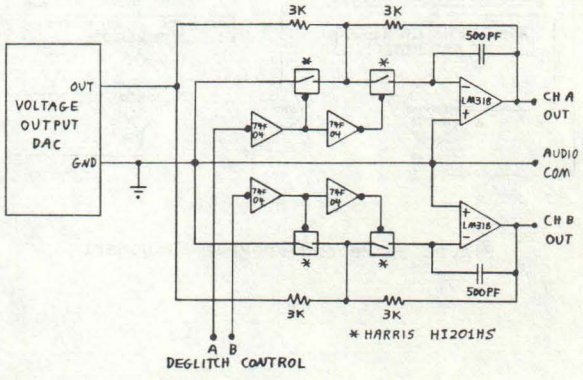


Figure 1. Stereo Deglitcher

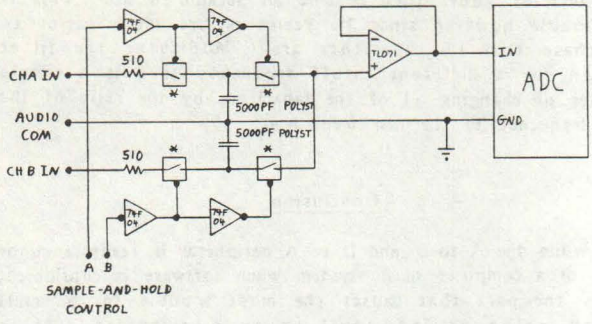


Figure 2. Stereo Sample-and-Hold

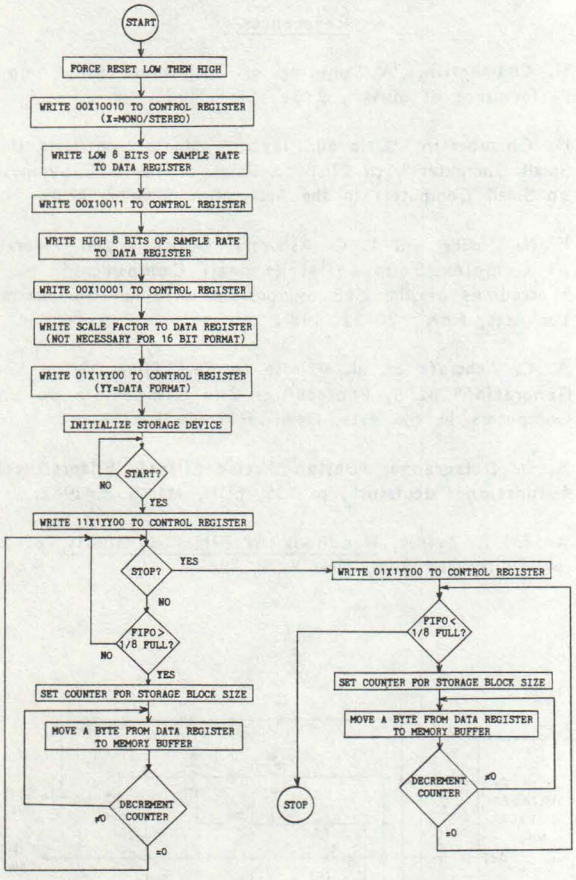


Figure 3. Record Program Flowchart

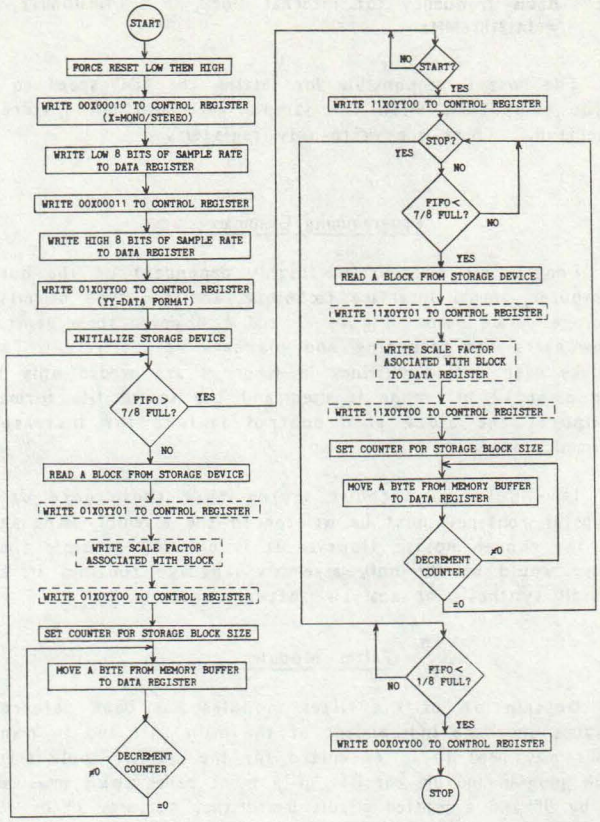


Figure 4. Playback Program Flowchart

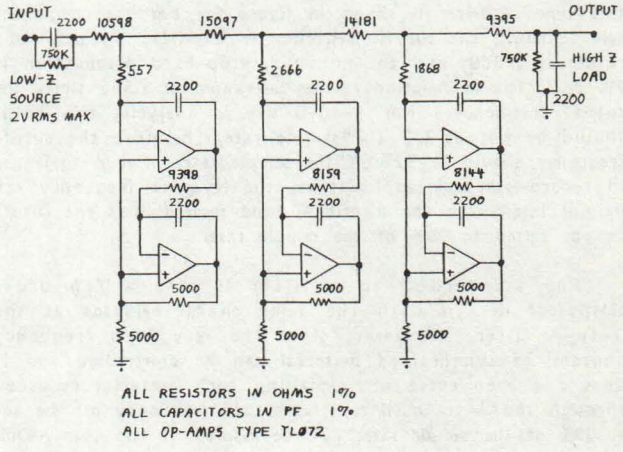


Figure 5. Anti-alias Low-pass Filter

A Computer Controlled Installation in a Gallery Space

George K. Shortess

Department of Psychology CU #17
Lehigh University
Bethlehem, Pa. 18015

Abstract

This paper describes a recent installation using a microcomputer as an interactive device for changing sound and video images as a function of the viewer's position in the gallery space.

Introduction

My involvement with microcomputers as an artistic medium has developed along with my attempts to express scientific ideas about the functioning nervous system, in artistic form.[1] By using microcomputers, I have been able to create works which change interactively with the viewer, whose participation can contribute to the work of art at any moment in time. The state of the art piece is a function of the interaction between the environment (including the viewers) and the internal structure of the piece itself (including the computer program and the hardware configuration). In many ways the art piece is like a human nervous system, changing over time, but constrained by both its environment and its internal structure.

In this paper I will describe a recent installation in which the art pieces responded to the positions of viewers in a gallery space by changing the sounds they emitted as well as the graphics displayed on video monitors. I will discuss the ways in which the microcomputer was used as part of the work, as well as some general ideas about aesthetics and contemporary cognitive psychology.

The Installation

Figure 1 shows a floor plan of the installation at the Kemerer Museum, Bethlehem, Pa. The larger squares and rectangles (labelled SC, M and B) were cloth covered wooden bases. On the

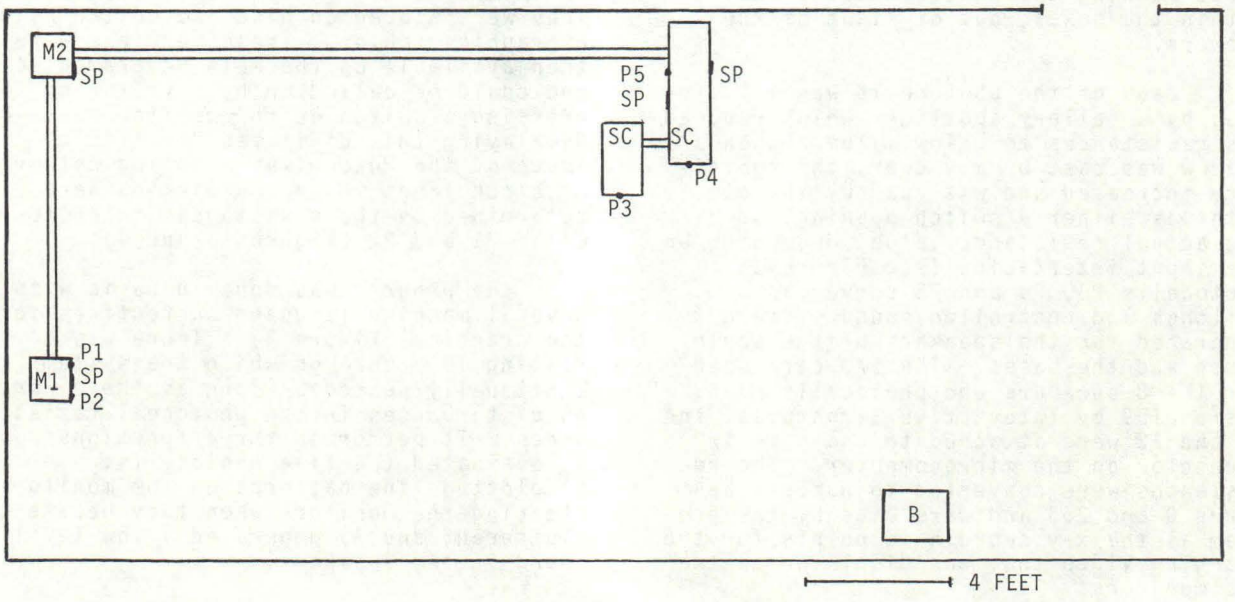


Figure 1. Floor plan of the gallery space

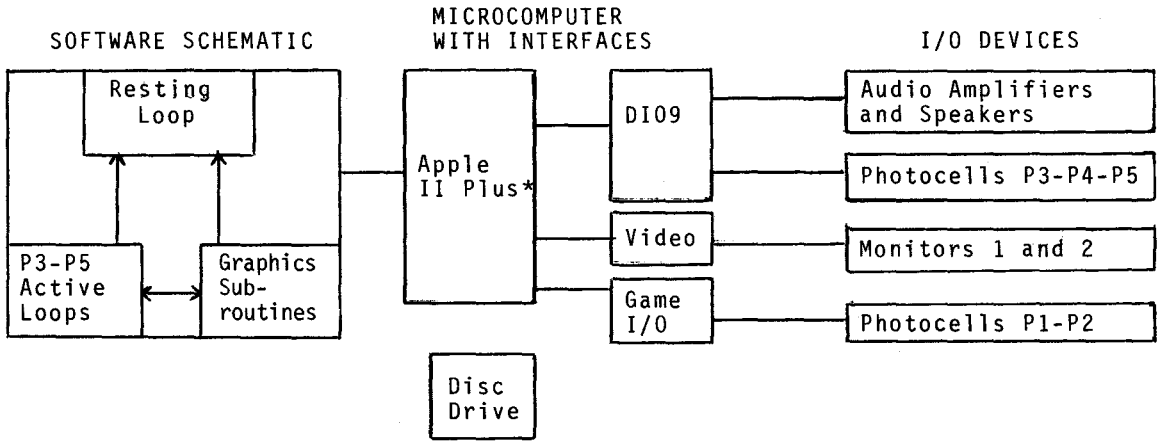


Figure 2. Schematic of software and hardware

M bases were video monitors (one each). On the SC bases was a sculptural piece made from aluminum tubing and containing four audio speakers. The M and SC bases were connected, by way of 1-1/2" square aluminum tubing (represented by double lines in Figure 1) to an Apple II Plus* microcomputer, which was located out of sight of the gallery. Within the bases were five photocells (indicated by P1-5 on Figure 1) and four speakers (indicated by SP), in addition to the four speakers on the sculptural piece. All speakers and photocells in the bases were under the cloth covering and thus not visible. The four speakers in the sculpture were enclosed in small aluminum boxes with cloth covers. All of these were attached to the microcomputer by wires running through the tubing and within the boxes, out of sight of the viewers.

Each of the photocells was illuminated by a gallery spotlight which reduced its resistances to a low value. When a shadow was cast by a viewer, the resistance increased and was read by the computer as either a switch opening, or as the actual resistance value, depending on the input interfacing (see Figure 2). Photocells P3, P4 and P5 served as switches and controlled sound patterns generated for the speakers of the sculptures and the bases. The I/O card used for the 8 speakers and photocells P3-5, was a DIO9 by Interactive Structures, Inc. P1 and P2 were attached to the game I/O connector on the microcomputer. The resistances were converted to numbers between 0 and 255 and were used by the program as the x-y coordinate points for the changing video that was displayed on the two monitors.

*Trademark of Apple Computer, Inc.

The output to the speakers consisted of brief clicks which simulate one kind of electrical activity of the human nervous system.[1] Low level activity was characterized by an overall low frequency of click occurrences. Higher levels of activity corresponded to a high overall frequency of click occurrences. In both cases the frequencies were not regular, but varied over a considerable range, with bursts and beat patterns being generated.

The video output was one of six digitized images taken from the Kemerer Museum. They were made from slide photographs by projecting them on a rear projection screen which was scanned by a video camera connected through an appropriate digitizing card (Digisector by The Micro Works) to the microcomputer. They were stored on disc and edited using a graphics tablet. These versions were then available on the main program disc and could be called up by a viewer by pressing a switch on the monitor base M1. Overlaying this digitized "realistic" image of the Museum was a moving network of black lines whose coordinates were determined by the resistances in photocells P1 and P2 (Figures 3 and 4).

The program was done in Basic with several machine language subroutines for the graphics (Figure 2). There was a resting loop through which the system continually passed as long as there were no disturbances in the photocell resistances. It performed three functions: 1) evaluated the five photocells; 2) plotted line patterns on the monitors, clearing the monitors when they became too cluttered; and 3) generated a low level of sound in the speakers.

As soon as a change in a photocell was detected the program switched out of this loop. When one of the photocells P3, P4 or P5 was activated, the program switched to one of three loops which generated new higher levels of sound in the speakers closest to the particular photocell activated. When P1 or P2 changed, it switched to a subroutine which changed the line pattern on the video display, using the new coordinates determined by the resistances of these photocells. This subroutine also generated a brief sound burst in the speakers in the monitor bases. As long as there was activity at one of the photocells P3, P4 or P5, the program stayed in that respective loop. If other photocells were activated, it would then switch back and forth between the loops involved. When P1 or P2 was activated, it would redraw the lines and then switch back to the loop from which it came.

Two different pictures were stored in the computer at one time, one at addresses $2000-$3999 and the other at $6000-$7999 ("Page 3"). High resolution Page 2 was always in view and loaded from Page 1 or "Page 3" with a machine language subroutine whenever it became cluttered with black lines. The image also changed when the resistance of P1 got large enough, so that rapid alternation between two views of the Kemerer could be achieved by appropriate movements of the viewer.

In addition to the computer driven displays just described, there was a separate unit called Bullrushes (labelled B in Figure 1). This unit consisted of a cloth covered base unit with 3 finished pieces of aluminum tubing extending perpendicularly from the base (total height about 6 feet). In the base was a small hard-wired unit that emitted deep clicking sounds when a photocell switch was activated. To the imaginative viewer the sounds could be from bullfrogs and the aluminum tubing could be the rushes.

To complete the exhibition 20 watercolors of landscape scenes and 3 silkscreen prints of the digitized Kemerer Museum, were arranged on the gallery walls.

Evaluation

The overall attempt was to create an interesting space which had a kind of quiet humor. When a viewer entered the gallery, Bullrushes served to set the expectation that something different from a standard pictures-on-the-wall show was in place. Further movement into the gallery space past P4 or P3 further reinforced these ideas. In addition, the digitized images were designed to suggest a different way of looking at the Museum itself, which is a Victorian mansion filled with the decorative art collection of Anne Kemerer. The Victorian images seemed just right to present in digitized form.

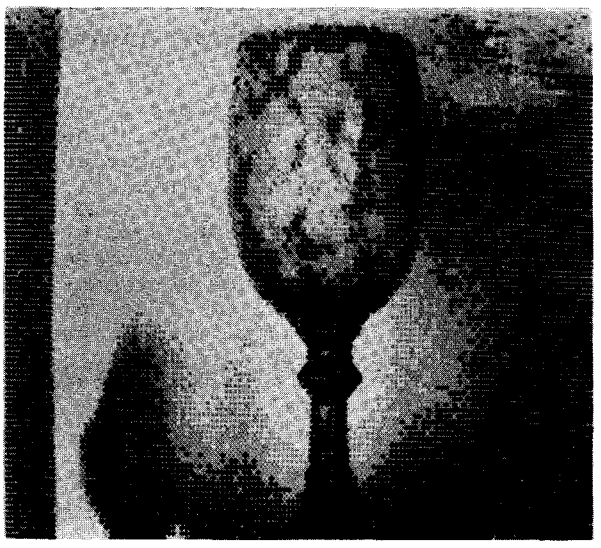
The reaction to the show was mixed. Some did not know what to do; others became more active. One problem arose when three or four people tried to interact with the computerized work at the same time. The feedback, either in sound or video, was noticeably slow and the viewers were no longer sure that they were controlling anything. But then, if one of us tries to interact meaningfully with four people at once, we become a little confused too. This problem could, of course, be corrected with several computers or more reliance on machine language programming.

Some Relationships to Cognitive Psychology

While this work has been based on ideas about neural function at the level of nerve cells and nerve nets, there is implicit in it, certain models of cognitive functioning that are behaviorally based. Cognitive psychologists, such as Neisser[2] or Wickelgren[3] for example, have developed models of the way perceptual and cognitive processes may take place. Without going into the specific details, it can be recognized that these models are essentially networks with activities or functions taking place at various nodes, with the transmission of information between nodes. In this way the formal features of these cognitive models are also reflected in my network art forms. In addition, all major contemporary theories of perception and cognition recognize and emphasize the active, interactive nature of perception. It is not a process of passive reception, but involves active seeking and interaction. I also emphasize this characteristic in my work. Taken together then, my work can be interpreted as an artistic statement about cognitive function as well as neural function. And at some level, the cognitive-perceptive and the neural functions may be the same.

References

1. Shortess, G. K. An Application of a Microcomputer as an Interactive Art Medium. Proc. Symposium on Small Computers in the Arts. Phila., Pa. Oct. 15-17, 1982. pp 75-77.
2. Neisser, U. Cognition and Reality. San Francisco: W. H. Freeman, 1976.
3. Wickelgren, W.A. Cognitive Psychology. Englewood Cliffs, NJ: Prentice-Hall, Inc. 1979.



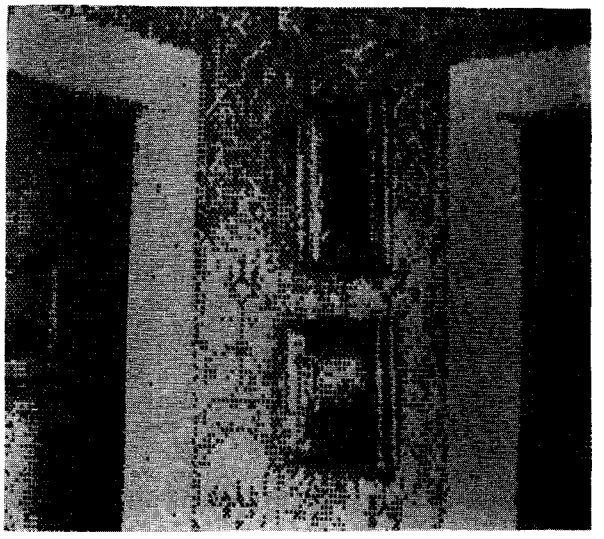
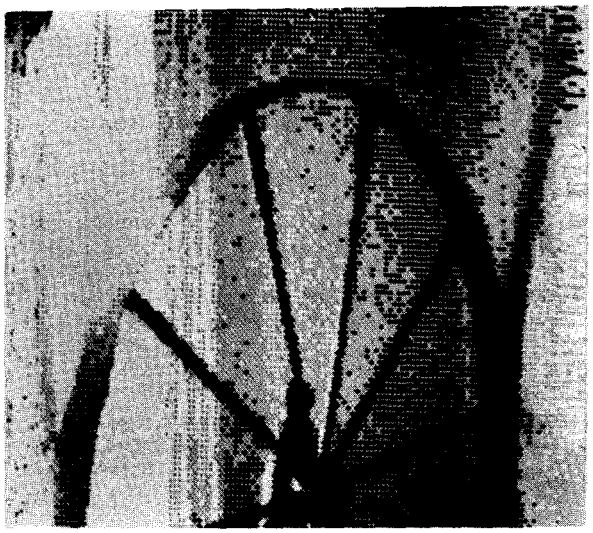


Figure 3 - Video images of the Kemerer Museum



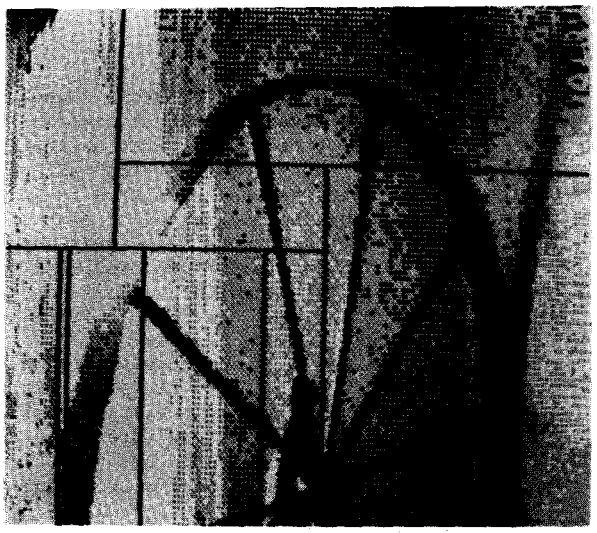


Figure 4 - Video images with and without grid lines

COMPUTERS, MUSIC, AND THE ARTS: a liberal arts college course

William Matthews

Music Department, Bates College
Lewiston, Maine 04240

ABSTRACT

For the past two years, a small computer system for music and graphics synthesis has been in use at Bates College, a private liberal arts campus in Lewiston, Maine. A course for students with no previous computer experience is offered; it has been extremely popular. The course introduces students to principles of computing, introduces several programs for music synthesis and graphics, introduces the PASCAL programming language, and allows students to create works of music and art, all in a single 14-week semester. The paper describes the pedagogy employed, discusses some of the artistic issues involved, and ends with examples of some of the student projects.

INTRODUCTION

It has been nearly twenty years since Susan Sontag identified an emergent response to what had been called the crisis of the "two cultures", the gulf separating humanists and scientists. In Against Interpretation (1966) Sontag described a new breed of artists whose work often employs advanced technologies in untraditional and sometimes esoteric aesthetic contexts. Central to her formulation was the idea that these artists are stimulated rather than intimidated by the complexity of contemporary society and science, and that their work was rapidly making irrelevant previous distinctions between the "high" culture of unique masterpiece artifacts and the "low" culture of popular mass-productions.

Sontag referred to such artists as pioneers of a "one culture" that might bridge any separation between humanists and scientists. In the two intervening decades, this culture has been represented not by any aesthetic or stylistic uniformity, but rather an artistic Brownian motion, in which a multitude of individual artists vibrate within small spheres of influence. This situation is enjoyably democratic, but has made critical or theoretical overview of the art itself difficult.

It is not difficult at all, however, to point to a single new technological tool which has emerged since 1966, more powerful and influential than any other and rapidly becoming more common than any other. In the context of this paper, neither apology nor explication need proceed a discussion of the role of microcomputers in the arts. With devices and programs for graphics, sound synthesis, and text processing now available in every community of a size in the country, one accepts their instrumental role at the foundation of a burgeoning "one culture".

Art using microcomputers demonstrates several attributes: it is often participatory in some way; it is often ephemeral, leaving no artifacts; it usually involves small audiences, often single viewers or listeners. The machines themselves are flexible tools, but make stringent demands on users who would employ them flexibly. Users must "think like computers", unlike the thinking traditionally done by artists, and the art most "organic" to the new technology extends sensation in abstract and often dynamic ways unlike previous art. These attributes require new educational strategies to introduce the techniques of computer-assisted art to potential members of a "one culture".

THE COURSE

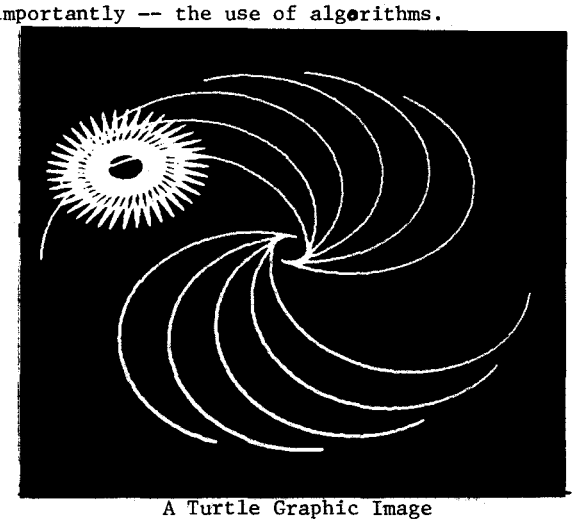
At Bates College in Maine, a course has been instituted to help students explore some possible artistic applications of microcomputers. Offered under the aegis of the Music Department, and taught by a composer with previous computer music experience, Computer, Music, and the Arts allows students to work with various operating systems, programs, programming languages, and hardware devices to generate sounds, images, and tests. The equipment for the course consists of a Z-80 based general purpose microcomputer with two double-side, double-density floppy disk drives, a Micro-Angele refreshed raster scan graphics system, and a Casheab 32-voice digital sound synthesizer. The system is called the Bates MusicBox.

CH1930-7/83/0000/0086$01.00

The course begins with a general overview and history of computers and their artistic uses, with audio and visual illustrations. Following this introduction students begin immediately with personal experience of computer art at the local video game parlor. Besides being fun, experience with commercial games is pedagogically useful. Observations can be made about the dazzling images and sounds now possible, the meaning of "user friendliness", how users can interact with programs and vice-versa, how information is transmitted from users to programs and vice-versa, the meaning of real-time, how randomness can be useful, how animation is achieved, the usefulness and drawbacks of "black boxes", and how programs maintain or lose their interest for the user. In addition, social, psychological, and aesthetic issues are revealed, setting discussion firmly with a humanistic as well as technical contest. In fact, a structure for the rest of the course evolves naturally from the discussion which follows esperience with games.

Because no previous computing (or for that matter, artistic) experience is prerequisite for the course, care is taken to introduce concepts and procedures gradually. The system is treated as a series of black boxes each of which is opened only when necessary to permit the user additional flexibility and control. Assigned tasks initially produce immediate gratification, and only gradually require deferral of gratification. Following bouts with video games, students are next asked to play a single pre-programmed piece of music on the College's system; this assignment requires turning on the computer, leading CP/M from a disk, calling up David Rayna's Casheab softwear package, and specifying a single instruction and file name. These five discrete tasks take less than a minute. The next assignment requires use of a different operating system (the UCSD PASCAL p-system) and a few more tasks, including some data entry; it results in a snappy graphic image that takes two or three minutes to generate, but which can be changed if new data is entered on restarting the program. Such a simple sequence of assignments is used to demonstrate levels of user-friendliness, increasing flexibility of possible outputs, and the advantages of deferred gratification. It also allows the introduction of relevant concepts such as operating systems, programs, data, devices, input, output, and so on; word by word, the vast vocabulary that sprouts up around computers can thus be introduced.

In the second week, students are asked to become proficient as users of an interactive program for turtle graphics, modeled after that described by Seymour Papert in his book Mindstorms: Children, Computers, and Powerful Ideas (1980). The Bates version of this program was written in FORTH by Andrew Wilcox, and is based on a kernel of instructions such as FORWARD, BACK, LEFT and RIGHT that, when combined with constants or user-defined variables for angles and distances, cause a little stylized turtle to move about the video screen, leaving graphic traces in its wake. Although based on a small handful of instructions, the program is powerful in that users can define their own new procedures using any sequences of already-defined instructions. Before they know it, students are thus programming, grappling with quantification, hierarchical procedures and -- importantly -- the use of algorithms.



A Turtle Graphic Image

Because turtle graphics are completely interactive in real-time (FORTH is an interpretive language), no compilation, editing, or file transactions are necessary. The latter two are next introduced as students return to the Rayna software for music synthesis. One component of this software allows users to address the digital oscillators directly, and requires one to specify frequencies, amplitudes, and waveforms; the temporal domain must also now be controlled. At first, the output from the synthesizer is heard through loudspeakers and seen on an oscilloscope; when students become more adept, lissajous figures are produced by phase-shifting the monophonic synthesizer output to provide different x and y axis inputs for the oscilloscope. Anuone with experience of lissajous figures knows how dynamic and unstable they can be, and what complex and beautiful images can be made rather easily.

The second half of the semester-long course takes students into the less user-friendly domain of general purpose programming languages; PASCAL is chosen for its highly-structured character and its resulting educational usefulness. Although the frustrations of learning and using a complicated operating system, a full-feature editor, a compiler and a language with fairly complex syntax are many, students understand by this point why the resulting power and flexibility are worth the effort. Programming thus comes to seem a natural extension of the artist's work towards gaining more control over a new tool already experienced as useful, enjoyable, and potentially powerful. From PASCAL, external machine-language procedures can be called, allowing students to control the graphics and synthesizer as they work out individual projects in the time remaining for the course. The final black box of machine architecture remains closed, but a surprising number of others have been opened, proving to initiates

that the seemingly arcane world of computing is not so esoteric after all.

The final class projects have included musical compositions, random poetry, the game of hangman, a spelling game that shows a picture of an object when correctly spelled by the user, an extended piece for sounds producing lissajous figures, a program to produce images like Kandinsky drawings, a program to produce designs for Easter eggs, and a piece for sounds, lissajous figures and narrative text presented on the video screen as captions for the other events. This final piece is called, affectionately, the Bug Opera. Several students continued work after the course ended, in formal and informal independent studies including the production of speech-like sounds with the synthesizer, musical compositions, and an elaborate work involving the construction of a device for visual display of computer-generated audio using reflective vibrating membranes and focused light sources.

CONCLUSION

Computers, Music, and the Arts attracted three times as many students as could be accommodated the first time it was offered. The students who did complete the course were surprised and pleased at how much they learned about computing in general as well as about specific artistic applications; they were also quite proud of their artistic products. Because concepts and techniques are explained in as generalized a way as possible, participants are prepared, after only a few weeks of work, to continue making art with microcomputers in other settings and with other devices. Having explored the mysteries one by one, future contributers to a "one culture" are well-equipped to start working.

CH1930-7/83/0000/0089$01.00 © 1983 IEEE

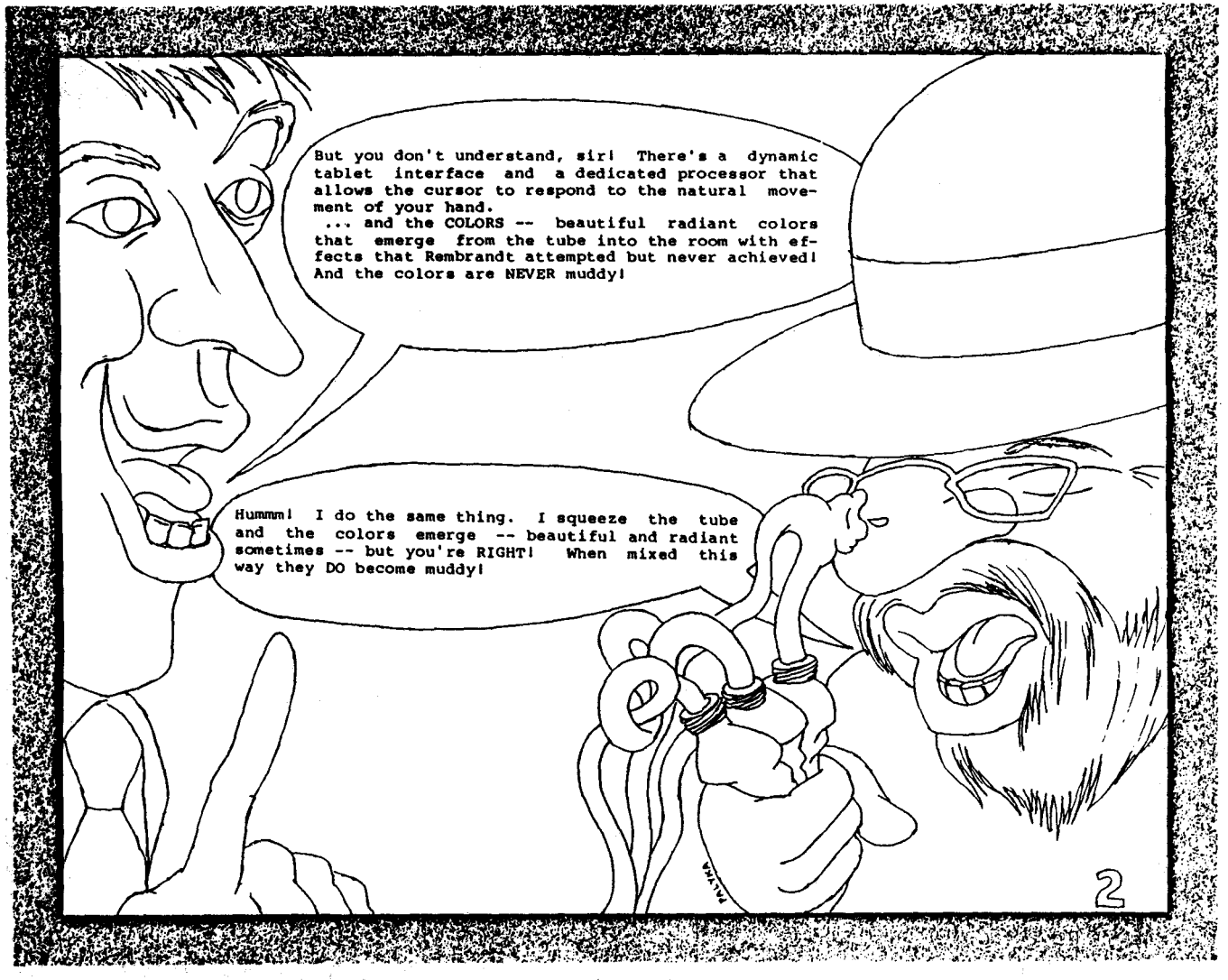
PATTER-PUSHIN'-PALYKA PRESENTS TOO-LOOSE-A-TREK THRU THE SELLING OF A PAINT SYSTEM

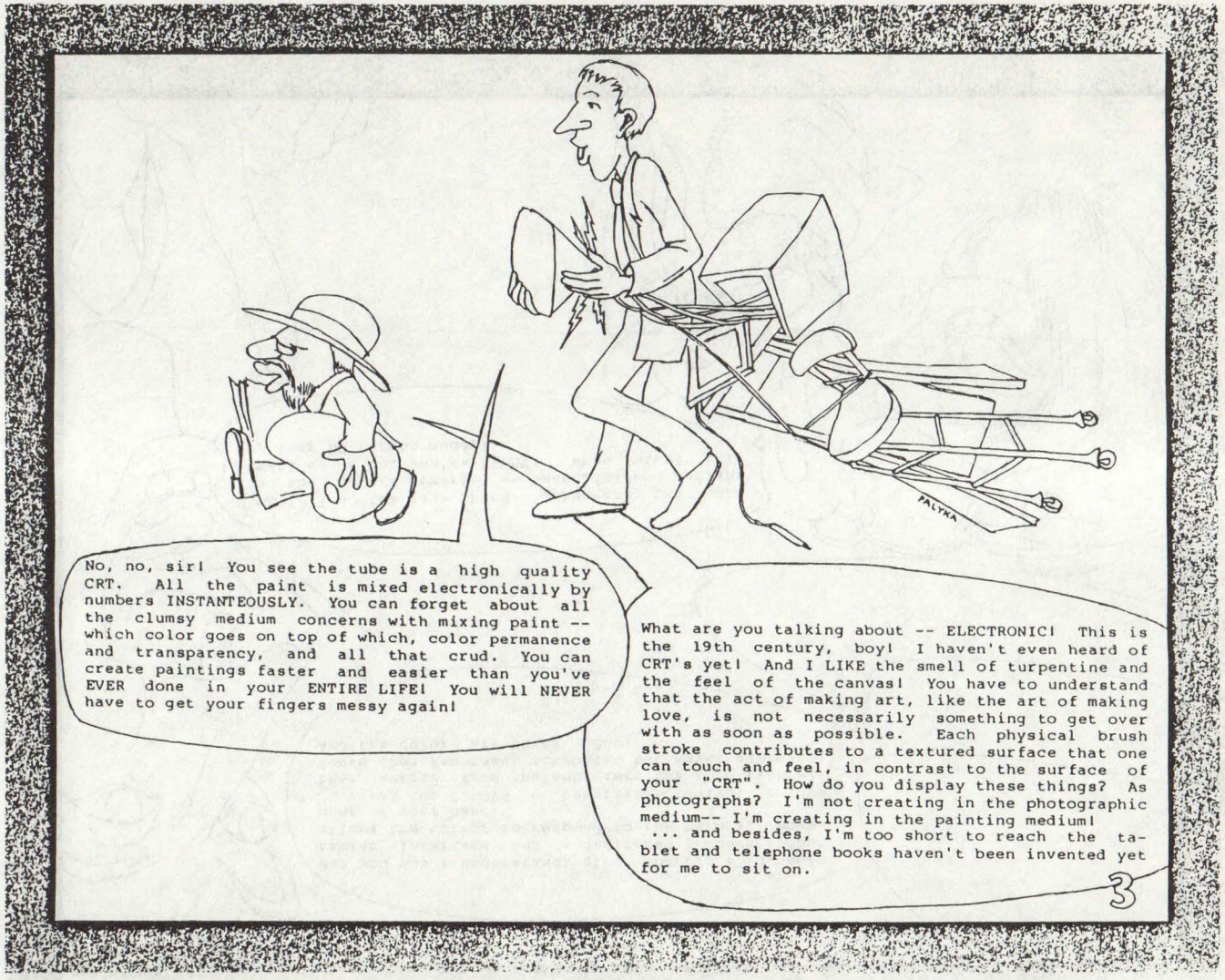
Duane M. Palyka

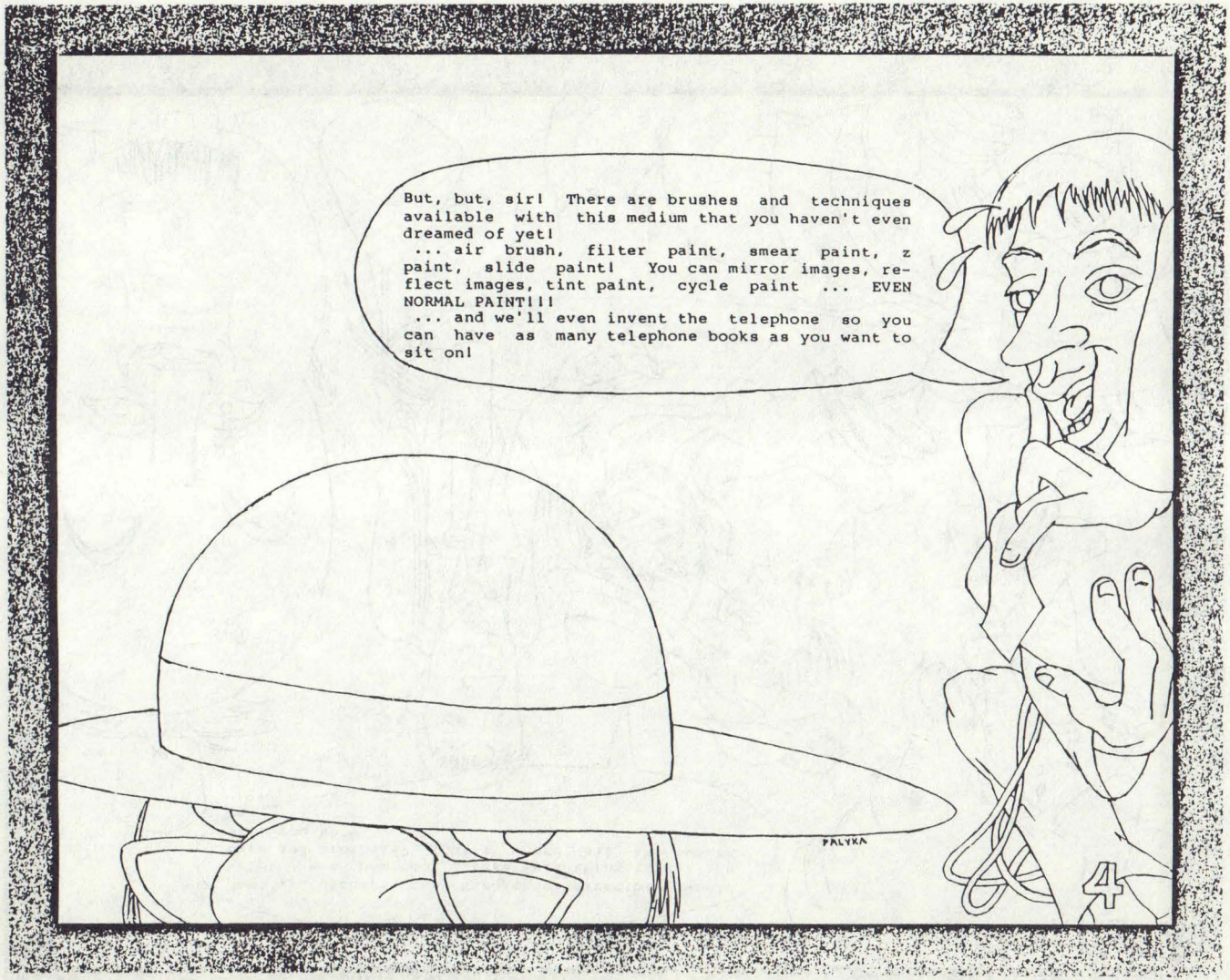
Computer Graphics Laboratory
New York Institute of Technology

May 1983

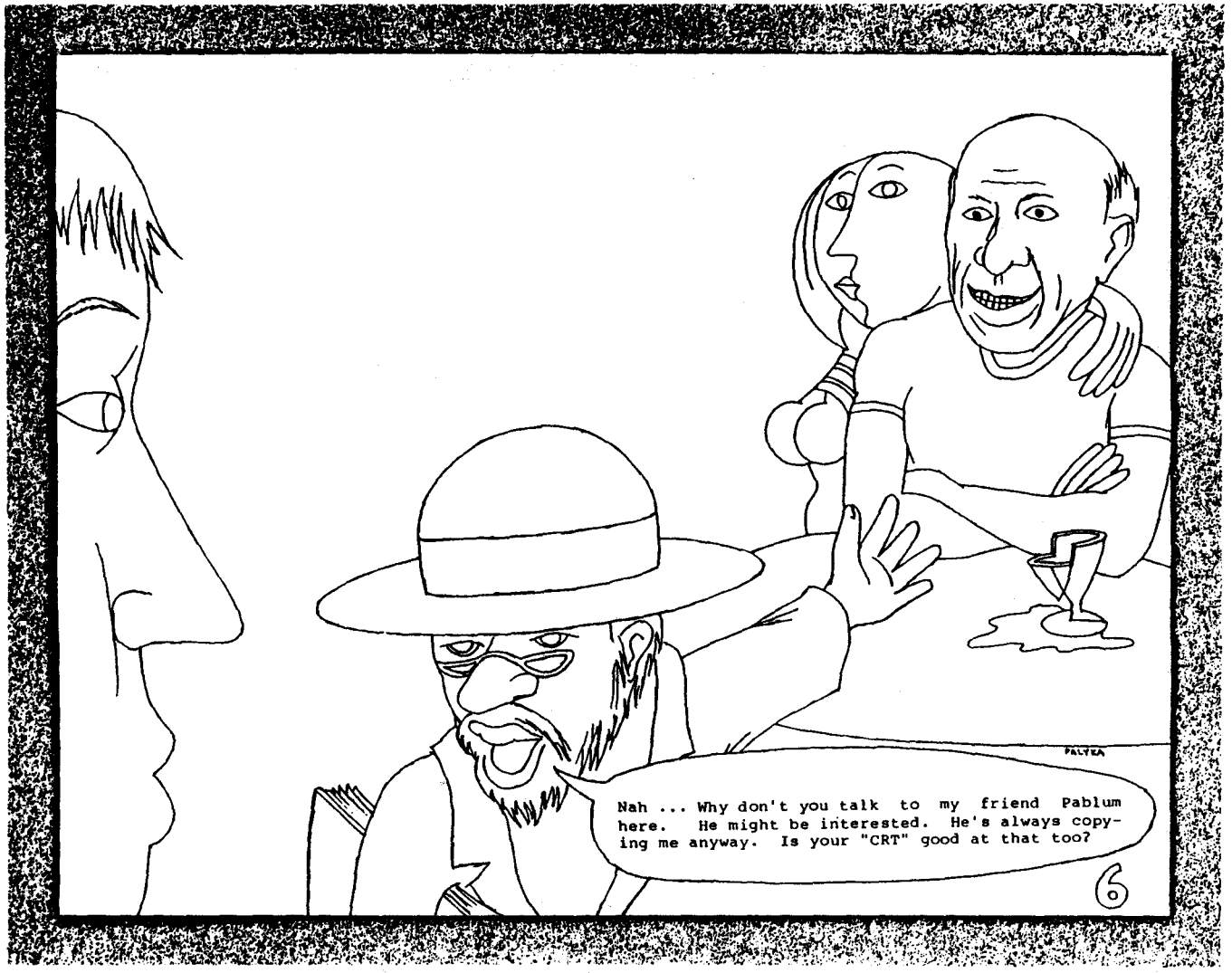

Excuse me, sir, but I couldn't help but notice that you look like a person who likes to compose two-dimensional images of your very own creation in FULL-COLOR!
... who perhaps has a longing need to display 256 distinct colors AT THE SAME TIME in a 512 x 512 raster configuration! How's THAT for resolution?! Could I perhaps interest you in a new electronic paint system?
Go away, boy, you bother me. Can't you see I'm busy looking for the seamy side of life to render in beautiful spontaneous line and fantastic color in all its wretched glory? You just don't look seamy enough!
COPYRIGHT © 1983 PIXAR

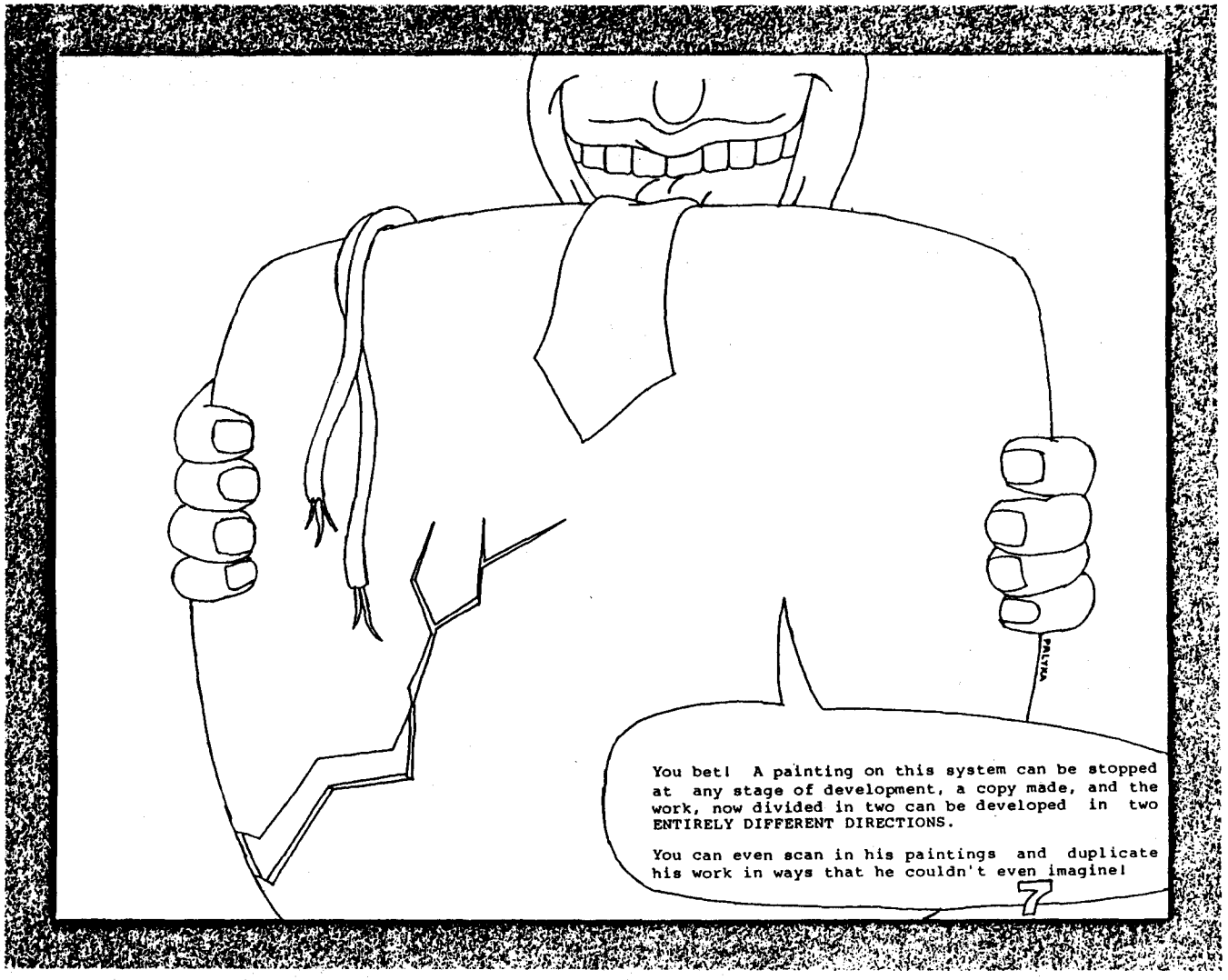

But you don't understand, sir! There's a dynamic tablet interface and a dedicated processor that allows the cursor to respond to the natural movement of your hand.
... and the COLORS -- beautiful radiant colors that emerge from the tube into the room with effects that Rembrandt attempted but never achieved! And the colors are NEVER muddy!
Hummm! I do the same thing. I squeeze the tube and the colors emerge -- beautiful and radiant sometimes -- but you're RIGHT! When mixed this way they DO become muddy!
2

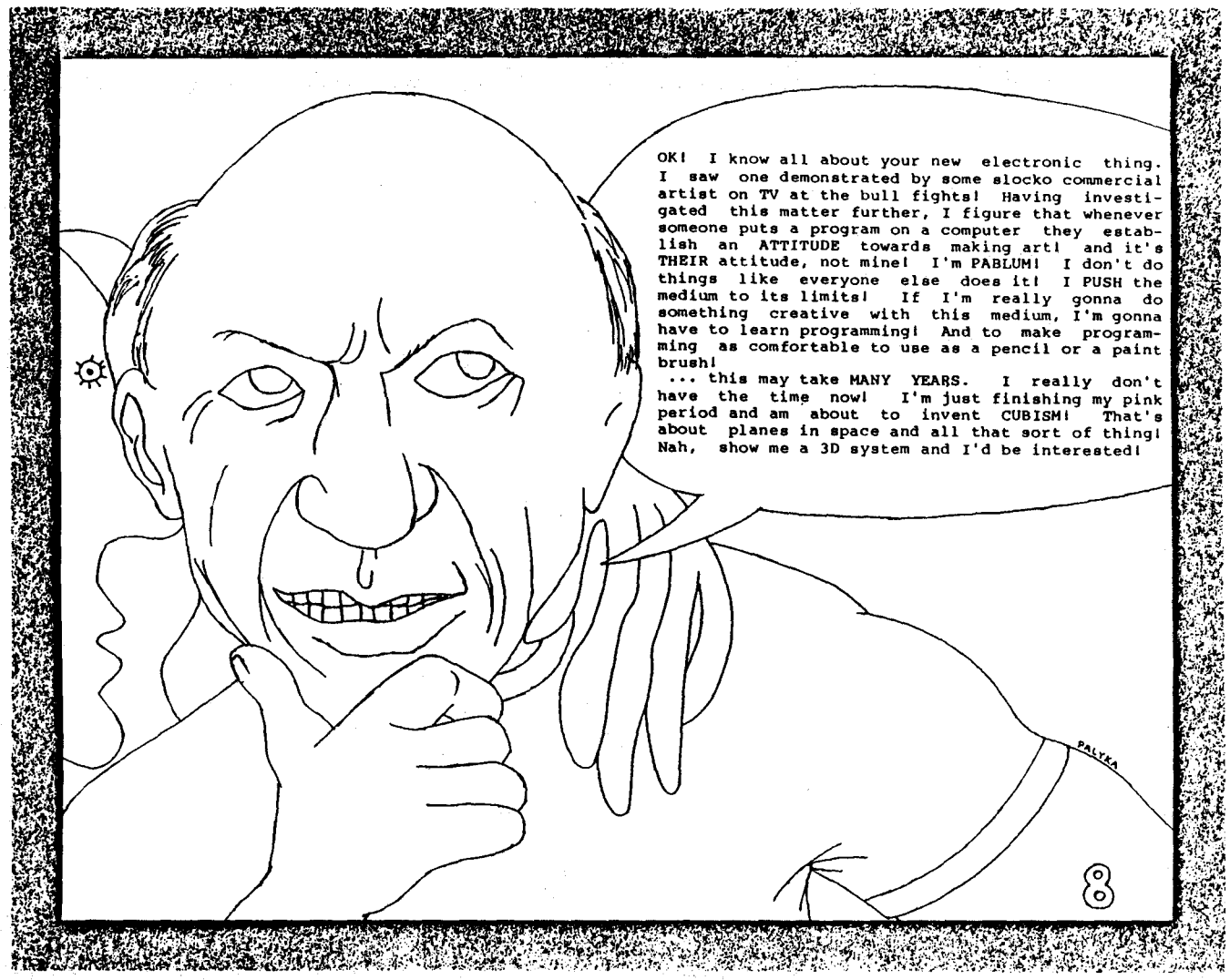

No, no, sir! You see the tube is a high quality CRT. All the paint is mixed electronically by numbers INSTANTEOUSLY. You can forget about all the clumsy medium concerns with mixing paint -- which color goes on top of which, color permanence and transparency, and all that crud. You can create paintings faster and easier than you've EVER done in your ENTIRE LIFE! You will NEVER have to get your fingers messy again!
What are you talking about -- ELECTRONIC! This is the 19th century, boy! I haven't even heard of CRT's yet! And I LIKE the smell of turpentine and the feel of the canvas! You have to understand that the act of making art, like the art of making love, is not necessarily something to get over with as soon as possible. Each physical brush stroke contributes to a textured surface that one can touch and feel, in contrast to the surface of your "CRT". How do you display these things? As photographs? I'm not creating in the photographic medium-- I'm creating in the painting medium!
... and besides, I'm too short to reach the tablet and telephone books haven't been invented yet for me to sit on.
3
PALYKA

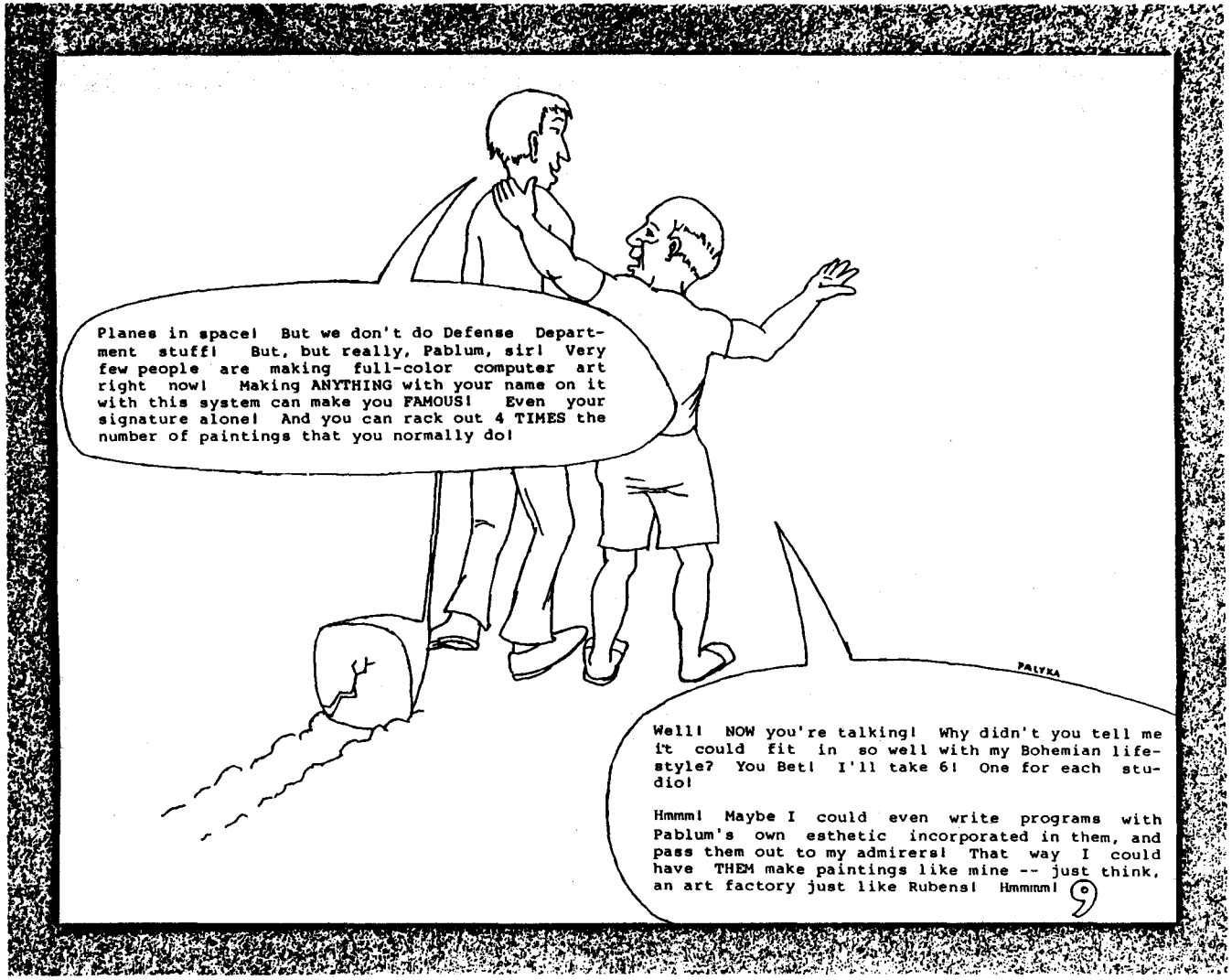

But, but, sir! There are brushes and techniques available with this medium that you haven't even dreamed of yet!
... air brush, filter paint, smear paint, z paint, slide paint! You can mirror images, reflect images, tint paint, cycle paint ... EVEN NORMAL PAINT!!!
... and we'll even invent the telephone so you can have as many telephone books as you want to sit on!
PALYKA
4


Hmmm ... perhaps the Manhattan directory would help! But how do I take this thing to all the bars and brothels? That's where all the women are, you know.
PALYKA
5


Nah ... Why don't you talk to my friend Pablum here. He might be interested. He's always copying me anyway. Is your "CRT" good at that too?
6


You bet! A painting on this system can be stopped at any stage of development, a copy made, and the work, now divided in two can be developed in two ENTIRELY DIFFERENT DIRECTIONS.
You can even scan in his paintings and duplicate his work in ways that he couldn't even imagine!
7


OK! I know all about your new electronic thing. I saw one demonstrated by some slocko commercial artist on TV at the bull fights! Having investigated this matter further, I figure that whenever someone puts a program on a computer they establish an ATTITUDE towards making art! and it's THEIR attitude, not mine! I'm PABLUM! I don't do things like everyone else does it! I PUSH the medium to its limits! If I'm really gonna do something creative with this medium, I'm gonna have to learn programming! And to make programming as comfortable to use as a pencil or a paint brush!
... this may take MANY YEARS. I really don't have the time now! I'm just finishing my pink period and am about to invent CUBISM! That's about planes in space and all that sort of thing! Nah, show me a 3D system and I'd be interested!
PALYKA
8


Planes in space! But we don't do Defense Department stuff! But, but really, Pablum, sir! Very few people are making full-color computer art right now! Making ANYTHING with your name on it with this system can make you FAMOUS! Even your signature alone! And you can rack out 4 TIMES the number of paintings that you normally do!
PALYKA
Well! NOW you're talking! Why didn't you tell me it could fit in so well with my Bohemian lifestyle? You Bet! I'll take 6! One for each studio!
Hmmm! Maybe I could even write programs with Pablum's own esthetic incorporated in them, and pass them out to my admirers! That way I could have THEM make paintings like mine -- just think, an art factory just like Rubens! Hmmmm!
9

Microcomputer Typography

Michael Keith
D46 Abbington Drive
Hightstown, NJ 08520

This paper will describe and illustrate a program called EROFF (Enhanced Run-OFF), a typesetting program for the Apple II computer. This program, when used with an inexpensive printer such as the Epson MX-80 and a commercially-available word-processing program, can format and print very high quality text and graphics. Features of EROFF include the ability to use different fonts of arbitrary sizes, proportional and fixed spacing, text justification, table construction and simple line drawing, algorithmically generated italics and boldface, and mixing of graphics with text. As an illustration of its use, this entire paper (including figures) was printed using EROFF.

Overview

One of the basic features of EROFF is its *independence*. That is, all its features are isolated from the editor, word processor, and printer hardware that may be used with it. To use a different printer, for example, only a relatively simple driver program needs to be written. All other aspects of program use remain the same. I am currently using an Epson MX-80 printer and the Screenwriter II word processor.

The way this independence is achieved is depicted in Figure 1, which shows a simple hardware/software block diagram of the operation of EROFF. The two key characteristics of EROFF which allow this flexibility are:

1. EROFF works with standard Apple text files for input. This text file may either be one created with an editor or one which has been run through a text-formatting program to be justified, formatted, etc. All EROFF commands are sequences of one or more control characters embedded in the text file. These characters are ignored by the formatting program and passed through to EROFF.

2. The interface to the printer, as shown in the figure, is simply a screen dump of Apple hi-res pages 1, 2, and 3, concatenated horizontally. All printing is done to the Apple screen, then the screen images are dumped to the printer. This has the additional advantage of making many of the features of EROFF easier to implement (it is easier to plot to the Apple screen than directly to a printer).

The control character commands are assigned in a simple mnemonic way. For example, ^P turns proportional spacing on or off, ^I does italics, ^E does emboldening, ^F does font changes, etc. Not all of the 26 letters are assigned to commands, leaving room for future expansion.

The screen-dump interface for the Epson (which produced this paper) uses the highest resolution dot-graphics mode of the printer for an effective resolution of about 100 dots per inch.

I will now briefly describe some of the features of EROFF in more detail.

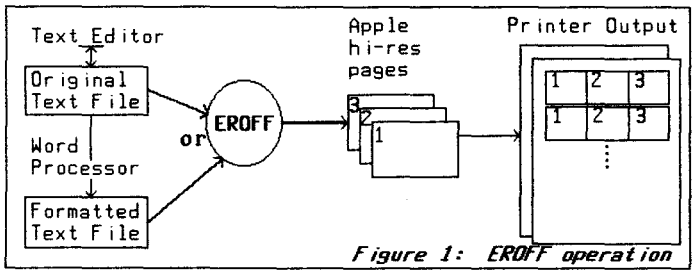


Figure 1: EROFF operation

Features

One of the major features of any typesetting program is the ability to use different typefaces (or **fonts**). A unique feature of EROFF is the fact that fonts can be completely arbitrary in size. Many font packages for the Apple are restricted to 7 x 8 fonts (the standard Apple size font) or multiples thereof such as 14 x 16. In EROFF special character-plotting routines are used which circumvent these restrictions by, for example, allowing a character to be drawn at the extreme right edge of hires page 1 which is automatically split correctly between pages 1 and 2 (see figure 2).

Here are some sample EROFF fonts of various sizes:

Old English Font 28x32

Narrow Font (13x32)

SHADOW FONT 20X24

Very small font (7x12), which will be even smaller after this paper is reduced!

Another feature of EROFF is ability to embolden and/or italicize any available font. This is done by slightly offset overstriking (for bold) or changing the character slope (for italics). This is much more efficient than storing separate fonts for italics and bold, and also allows interesting effects such as italicized Old English print! For example, here are the four fonts obtained by combinations of emboldening and italics applied to a sans-serif font:

Here is the standard font.
Here is the bold version of the font.
Here is the italics version.
And here is bold AND italics.

Another desirable feature of any typesetting system is the ability to do proportional spacing; that is, to allow the spacing between letters to depend on the width of the letters themselves. For example, here is a sample sentence printed in both fixed and proportional spacing.

Fixed:
Lillian locates lilacs on Illampu.
Proportional:
Lillian locates lilacs on Illampu.

This paper w ill describe and illustrate a progr im called EROFF (Enhanced Run-OFF), a typesetting program for the Apple II computer. This program, when us ed with an inexpensive printer such as t ie Epson MX-80 and a commercially-availat ble word-processing program, can form at and print

Figure 2:

Snapshot of the images in hi-res pages 1 & 2 during run-off of this paper, showing how letters actually can span page boundaies. This allows easy use of arbitrary character sizes.

Because it is handled by EROFF, proportional spacing is available regardless of whether the printer being used supports this feature.

Of course, there is no reason that a font has to contain the familiar ASCII character set. A font can contain up to 96 text or graphic symbols of any sort. For example, Figure 3 uses a chess font to illustrate a chess position. There are 24 primary characters in this font (each of the 6 different chess pieces x 2 colors x 2 colors of background) plus some other miscellaneous characters (such as blank squares). These graphic fonts can be combined with regular fonts to allow some simple mixing of text and graphics.

Another feature of EROFF is its line-drawing capability. At present, only horizontal and vertical line-drawing is available, but this is quite sufficient for drawing boxes around text, making tables, etc. This works as follows: the three control characters ^V (Vertical), ^L (horizontal Line), and ^B (Box) are used to mark endpoints of lines. For example, the following input

```
^B                ^B
  Words in a box
^B                ^B
```

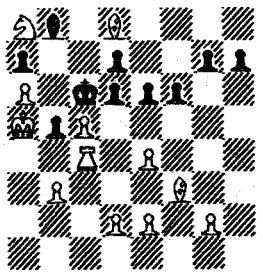
will produce the output

```
Words in a box
```

A more complicated example (a multi-column table) is shown in Figure 4.

EROFF also has a "preview mode" which runs off the document to the monitor screen rather than to the printer. This allows checking for formatting or other errors without having to wait for an actual hardcopy printout.

An interactive mode of EROFF is also available. In this mode, the user types on the Apple keyboard and the text appears on hi-res page 1. Most of the features of EROFF are available in this mode such as different fonts, italics, emboldening, and even proportional spacing, providing the unique experience of a proportional spacing typewriter! In conjunction with a graphics editor or paint program, this mode can be used to produce labelled drawings or figures. Figures 1 and 2 in this paper were produced in this way.



Chess Problem #1:

White to move:

1. What was Black's last move? Prove it.

2. White to mate *on the move*.

Figure 3:
Illustration of the use of a graphic font. Each square of the chessboard is one character in a special chess font.

Group Ib Elements			
Symbol	**Name**	**No.**	**Weight**
Cu	Copper	29	63.54
Ag	Silver	47	107.87
Au	Gold	79	196.97

Figure 4: Sample table created with EROFF.

Limitations

The major limitation of EROFF at this time is speed. This is not due to slow execution speed of the program but rather to slow printing speed of the Epson MX-80 printer. A faster printer would significantly reduce the time it takes to run off a document. Even so, it is not unacceptably slow; presumably, if you want high-quality print you will be willing to wait longer for it to be produced. The advantages of owning a personal computer (e.g., the ability to print out a final draft while you're eating dinner or doing some other activity during which the computer is usually idle) help make this delay tolerable.

Vital Statistics

The EROFF program is quite small, being composed of about 6K of Applesoft BASIC and about 2K of machine code. It *has* to be small since 24K is required for the three hi-res pages and a 5K area is reserved for font storage. It runs on an Apple II with no additional hardware or software (other than the printer), and has proven to be a useful addition to available Apple software tools.

KEYFRAME ANIMATION FOR THE MICROCOMPUTER

by William J. Kolomyjec, Ph.D., M.F.A.

Department of Engineering Graphics
The Ohio State University, Columbus, Ohio

Abstract

Keyframe animation is the process where one given figure or scene is changed into another. Important aspects of keyframe animation are: translation, iteration interval and transformation. A microcomputer can be used to generate transitional or in-between frames. An animated story can be produced with a series of keyframe sequences. This paper will present a cursory discussion of important aspects of keyframe animation. Programs will be given for the Apple (TM) microcomputer that will perform image transformation and animation. Image transformation will be accomplished by two kinds of interpolation, linear and quadratic. These algorithms will be given in programs that will allow prospective animation sequences to be previewed. Animation programs utilizing graphics page flipping will also be given which will provide a means of displaying image transformation for either observation or single frame recording.

Introduction

To begin, this article will present a cursory discussion of several aspects of keyframe animation. Next, two demonstrations of image transformation using linear and quadratic interpolation will be given. Linear interpolation requires two digitized images and the transformation between figures takes place along a straight line. Quadratic interpolation involves three digitized images and the transformation takes place along a curve of the second order through all three images. These two demonstrations not only illustrate interpolation transformations but they also provide a method of previewing potential animation sequences.

The remainder of this article will present application software that animates both types of interpolation transformations utilizing the page flipping capabilities of the Apple (TM) microcomputer. This technique enables complete frames to appear on the screen at once. Thus, the viewer does not have to watch each frame as it is being generated. Moreover, this technique should provide a decrease in overall animation time since, while an image is being recorded on film, the next frame is being processed.

Aspects of Keyframe Animation

Keyframe animation is an animation procedure where a number of critical or key frames in a sequence are designed. The concept is analogous to drawing the frames of a comic strip or storyboard. A computer with graphics capabilities is then used to generate a specified number of in-between or transitional frames. Keyframe imagery can be whole frames or parts of frames. In other words, complete scenes may be transformed, or parts of a scene may be allowed to change while the remainder of the scene remains unchanged.

Important aspects to be considered in keyframe animation are: 1) image location, 2) iteration interval, and 3) transformation. Control over these aspects will be of primary concern in the software to follow.

Translation involves moving images to desired screen locations in preparation for an animation sequence. Translation is also used in sizing a subject image. Translation is actually a three step process: 1) Initial translation. To center the image on its coordinate system origin. 2) Scale. To multiply the centered image coordinates by a scale factor to enlarge or reduce the figure. And, 3) Counter translation. To move the image to the desired screen location and adding in X and Y offset, known as the counter translation coordinates.

Iteration interval is the distance any one transitional image moves between frames. Interval size determines image speed in the animation. When planning for an animated film the film speed conventions for projection equipment must be considered. For 8 and 16 mm these are 18 frames per second for a silent film, and 24 frames per second for a film with a sound track. At constant film speed uniform intervals between images will produce uniform movement. Varying interval size will cause nonuniform movement. Clearly, at a constant film speed, image acceleration or deceleration cannot be achieved by employing uniform interval. However, there is one trick worth mentioning: One way of producing nonuniform movement with uniform intervals is to record more than one frame of an iterative image.

Image transformation is a very dynamic visual event. Transformation usually involves one figure changing completely into another, i.e.,

CH1930-7/83/0000/0103$01.00 © 1983 IEEE

metamorphosis. Figures may have the same or different counter translation coordinates. Movement alone may be created by changing an image into a copy of itself somewhere else on the screen. The illusion of depth may be created by changing an image into a scaled version of itself at the same or different screen location. Movement between images can also be in a linear or nonlinear fashion which is determined by the type of transformation algorithm and the number of digitized images employed.

There are several techniques available for metamorphosizing imagery but this discussion will focus on two variations of interpolation. The first type is linear interpolation. Linear interpolation allows the coordinate pairs of two digitized images having the same number of points to map into each other in a linear fashion. Figure 1 graphically illustrates the concept. If the line segments joining coordinate pairs of two figures are bisected and joined in order then an intermediate or interpolated figure is formed. Since the lines were bisected (divided into two equal parts) the interpolation fraction (later to be associated with a variable named PCT) is said to be 0.5. Further bisection of line segments between the interpolated figure and the figures to its left and right will yield interpolated figures corresponding to the interpolation fractions 0.25 and 0.75 respectively.

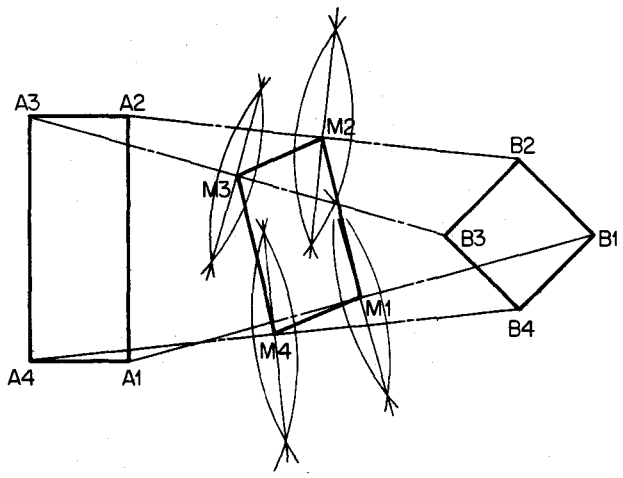


Figure 1. Graphical Interpolation of Two Figures

With the interpolation algorithm image transformation will not appear if both figures are specified to be the same, i.e., one figure is either an exact or scaled copy of the first. This may seem dubious, however, it is efficient from a programming point of view. It is important to realize that in the programs that follow, the user has the power to change subject images by substituting array names in the assignment statements where the manipulation arrays are defined. These locations will be clearly indicated in the code in the MANIPULATED DATA subroutine (starting at line 6000 in all programs to follow).

A second type of interpolation transformation technique is quadratic interpolation. Quadratic interpolation involves three figures each with the same number of coordinate pairs. Imagine a corresponding point of each figure to be somewhere on the screen. Next, imagine a second order curve, for example a parabola, passing through these three corresponding points in consecutive order, i.e., starting with the point from figure one, passing gracefully through the point from figure two and terminating at the point from figure three. This will be accomplished by way of a special subroutine.

Geometric Figure Database

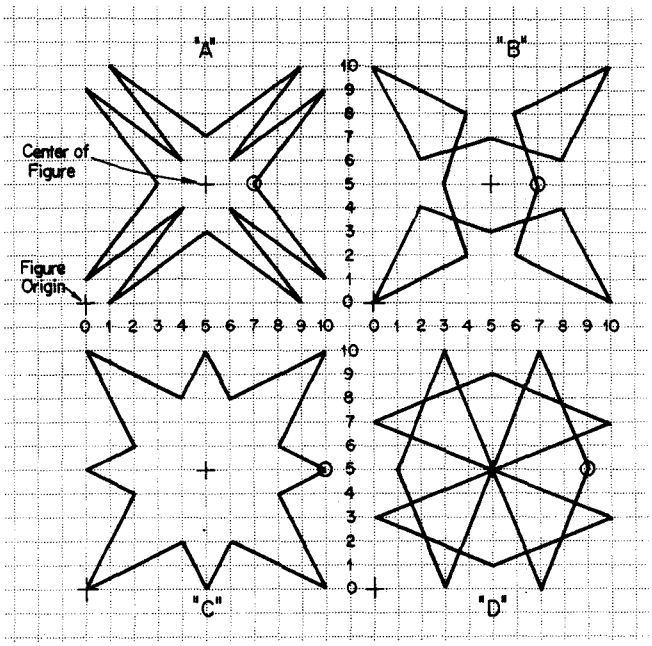


Figure 2. Geometric Figures

Four geometric designs, Figure 2, have been constructed and digitized. Each figure consists of sixteen line segments and they have been digitized in the same general direction, i.e., counter clockwise, from a similar starting point on a horizontal line to the right of the figure's center. This is a good rule of thumb to follow when digitizing imagery for interpolation. If direction and starting location criteria are not consistent, imagery will cross over itself (forming a knotted blob) during the transformation. In this event, transformation sequences will be of poor visual quality.

In the illustration, Figure 2, as well as in the program code of the subroutine, the figures are labeled A, B, C, and D corresponding to the

two dimension arrays that will hold their respective data. It will be the purpose of a separate subroutine entitled FOUR GEOMETRIC FIGURE DATA SUBROUTINE (starting at line 5000) to read in these data into their respective arrays. This subroutine will be a necessary part of each of the programs that follow.

Program Linear Interpolation

Program linear interpolation, Listing 1, will allow variable placement of two geometric figures anywhere within the screen boundaries. The scale of each figure is variable and the number of frames is arbitrary. In the example output from this program, Figure 3, figure "B" has been located in the upper left of the screen and figure "D" in the lower right. Each figure has been scaled with a different scale factor. Figure "B" has been scaled by a factor of three, figure "D" has been scaled by a factor of four. The transformation has been specified to take place in twelve iterations by assigning the value 12 to the variable NFRAMES (line 220). NFRAMES will control the total number of images (original images plus transitional images) in all programs given in this article. Moreover, temporal duration of a transformation in any animation sequence is controlled by the value of NFRAMES.

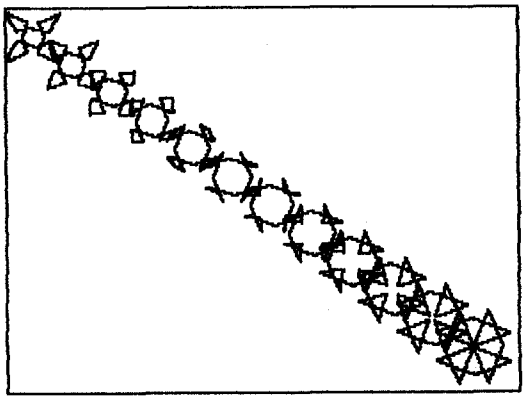


Figure 3. Linear Interpolation, Uniform Intervals

Notice two things in Figure 3: One, the completed drawing provides a preview of the animation sequence. This program can be used to create and develop prospective animation segments. Two, a border has been included for camera positioning. The border effectively outlines the workable screen area. A brief discussion of this program follows.

The main program (lines 100 - 999 in Listing 1) controls the operation of several subroutines as do most good graphics programs. First, the figure data are input into a series of four two dimensional arrays by a call to the GEOMETRIC FIGURE DATA SUBROUTINE. Note that in Applesoft (TM) arrays begin with a zero element. Thus, array declaration may seem to be "off by one." Next, manipulation arrays are dimensioned, and the data manipulation subroutine is called. Manipulation arrays are a convenient way to provide versatility to a program. In the MANIPULATE DATA subroutine, array assignment can be quickly changed. Notice that this subroutine takes care of translation, scaling and counter translation, achieving the first aspect of keyframe animation. The initial and terminal images (original figures) can be adjusted in terms of location and size within this routine.

The remainder of the main program does the interpolation and drawing. Once the desired number of transitional images are specified, the graphics mode is entered and the border routine is called. A loop is opened to generate the individual images. A one line sizing algorithm is used to calculate the interpolation fraction (PCT) based on the loop index (K) and the total number of desired iterations (NFRAMES). When K equals 1 then PCT equals 0, when K equals NFRAMES, then PCT equals 1. Thus, the interpolation fraction (given as a percentage) ranges between 0 and 1. The interpolation fraction provides for the second aspect of keyframe animation, i.e., iteration interval. It is used by the interpolation equations to generate the transitional image. The X and Y coordinate components of each corresponding image are calculated using this fraction.

These data, along with the appropriate "beam" control, are sent to a plotting subroutine (PLOTSUBB starting at line 1000). A point is either moved-to or drawn-to under the direction of this routine. The plotting subroutine also adjusts the screen origin to the lower left corner (as it should be) and corrects the aspect ratio. This technique increases the effective drawing area to 279 x 217 pixels.

When the transformation sequence is complete the last aspect of keyframe animation is achieved. The image is held on the screen by a dummy input statement. The user may go back and redefine variables and work a scene until it meets his/her creative requirements. A whole story can be put together with keyframe animation sequences. The procedure is to simply have the last frame of one sequence correspond to the first frame of the next sequence.

Linear Interpolation with Nonuniform Intervals

By adding one line of code and changing another, the iteration interval between transitional imagery can be altered. After making two changes to Listing 1, Figure 4 was produced. These changes are as follows:

```
ADD:
     225 HALFPI = 3.14159/2

CHANGE:
     280 IPCT = (K-1)/(NFRAMES-1):
     PCT = SIN (IPCT * HALFPI)
```

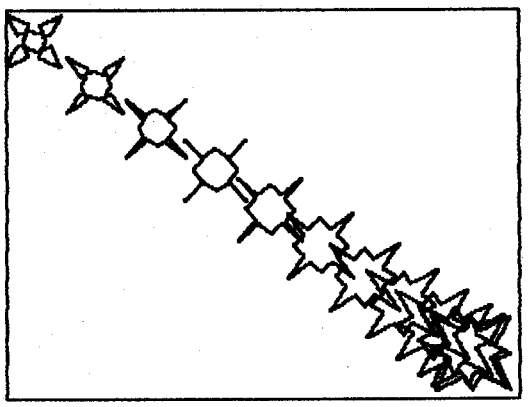


Figure 4. Linear Interpolation, Sinusoidal Intervals

This little algorithm uniformly samples the interval between 0 and PI/2 radians and takes the sine of that value. Since the sine function ranges between 0 and 1 in this interval, the value of PCT lies in the correct range, however, the interval spacing is now sinusoidal rather than uniform.

Program Quadratic Interpolation

The main differences between quadratic interpolation and linear interpolation are the number of figures involved and the path of the interpolation. Program Quadratic Interpolation is presented as Listing 2. Careful comparison will reveal similarities and differences. First of all, the plotting subroutine and the geometric data input subroutine contain no changes. The MANIPULATE DATA subroutine has been modified to translate, scale and counter translate three geometric figures and store the results in one three dimensional manipulation array, called M. M is dimensioned in the main program prior to calling to the subroutine. A three dimensional array best facilitates the programming of multiple figures.

In light of these similarities and relatively minor differences, both programs operate in identical fashion prior to the interpolating and drawing loop. The first major difference comes with the calculation of PCT. Since three figures are involved PCT must be defined to be in the range -1 to +1. The second major difference explains why. Notice the addition of a subroutine called FUNCTION QNTRP (starting at line 3000). Note that this subroutine is really a function subprogram. The function requires four parameters: an interpolation fraction and three corresponding coordinate elements. It returns the quadratic interpolation quantity relative to these values. FUNCTION QNTRP must be called for both the X and Y coordinate component. The curious may examine the code to gain an understanding of the mathematics involved in the algorithm.

The interpolation and drawing loop contains both major changes. When PCT equals -1 then the first figure is drawn. When PCT equals 0 then the second figure is drawn. And when PCT equals +1 then the third figure is drawn. Examination of Figure 5 reveals the path of the transformation is indeed nonlinear.

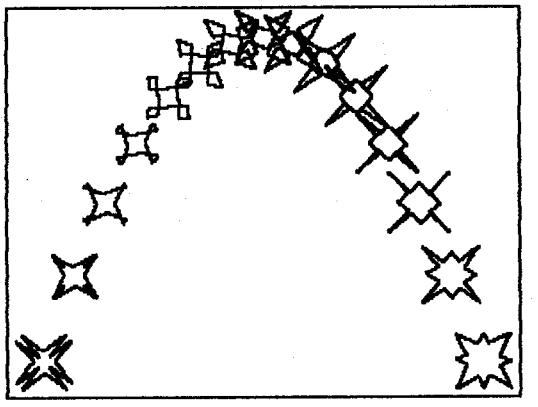


Figure 5. Quadratic Interpolation, Uniform Intervals

Animating Linear and Quadratic Interpolation

It is unfortunate that this software will only work on an Apple (TM). However, before a discussion of the specific program is given, the logic in general will be outlined. Perhaps readers with other computational equipment can adapt this concept to their particular systems.

Assuming that there are two graphics pages available and it is possible to display them on command, the following sequence will produce a slow but smooth animation.

Step 1: CLEAR graphics PAGE 1 and PAGE 2

Step 2: DISPLAY PAGE 1 (blank)

Step 3: START LOOP

Step 4: IF first time here THEN skip to next step ELSE, UNDRAW PAGE 2

Step 5: DRAW and SAVE PAGE 2

Step 6: DISPLAY PAGE 2

Step 7: IF first time here THEN skip to next step ELSE, UNDRAW PAGE 1

Step 8: DRAW and SAVE PAGE 1

Step 9: DISPLAY PAGE 1

Step 10: END LOOP

Both versions of Program Keyframe Animation (Listings 3 and 4) use the animation algorithm outlined above. Both listings are liberally

commented and the narrative that follows will address only the programming specifically related to the operation of the animation algorithm. Interpolation procedures are identical to those used in the previous programs.

First, two critical addresses must be identified to the program. SWITCH (decimal 230) controls the graphics page indexing value. The subscripted two value array PAGE(0) = 32 and PAGE(1) = 64 contains the required index values. If drawing is to take place on graphics page 1 then POKE SWITCH, PAGE(0) is necessary. If drawing is to take place on graphics page 2 then POKE SWITCH, PAGE(1) is necessary. The variable DISPLAY (decimal 49236) contains the address of an Apple (TM) "soft switch." POKEing DISPLAY with 0 displays graphics page 1. POKEing DISPLAY + 1 (another soft switch address) with 0 displays graphics page 2. Thus, the graphics pages can be drawn on and displayed with POKEs to SWITCH and DISPLAY.

The trick, as it were, is to keep track of what to do when. For this purpose a Boolean (logical) variable called FLIP is used. Boolean variables are seldom used as such in BASIC, however, the mechanism for their use is inherent to the language. For example, FLIP = NOT FLIP will change FLIP to a 1 if it is a 0, and to a 0 if it is a 1. FLIP serves a number of purposes: It allows toggling between the graphics pages. It represents the graphics page index when used as a subscript in the manner: POKE SWITCH, PAGE(FLIP). And it is used as a subscript to keep track of the page index in the drawn image storage array (i.e., T(coordinate, number, FLIP)) to undraw the appropriate image on the appropriate page at the appropriate time.

After the critical addresses and variables are defined and initialized the data is read in and manipulated. (Again, these subroutines, starting at lines 5000 and 6000 respectively, are identical to those used in Listings 1 and 2.) Next NFRAMES is defined. Remember every multiple of twenty-four frames represents one second of animation. Then, the animation algorithm begins as outlined above.

The graphics pages are cleared by executing HGR2 and HGR in this order. By executing HGR second, the graphics page index is set to page 1. Displaying Page 1 after it is cleared results in a blank screen. Before the interpolation and drawing loop begins, the graphics page index is set to page 2. Notice the loop index ranges from 1 to NFRAMES in steps or increments of two. This is because one pass through the loop draws and displays two frames.

The first thing that is done inside the loop is to POKE two more graphics soft switches which close the text window at the bottom of graphics page 1. If this is not done then page 1 will not be full page graphics. Apparently each time page 1 is displayed, it opens this text window, therefore, it must be continuously re-closed. A first-time-through-the-loop test is performed. Since it is true, no undrawing needs to be done. Control is transferred to the calculation of the interpolation fraction. PCT is defined based on the type of interpolation being used. If program keyframe animation is doing linear interpolation, then PCT will be defined in the range 0 to 1. If the program is doing quadratic interpolation, then PCT is defined in the range -1 to +1. In either case the DRAW/SAVE subroutine is called.

The DRAW/SAVE subroutine takes the value of the interpolation fraction and draws the appropriate transitional image on the graphics page currently in background, designated by the graphics page index. While this image is being generated, its coordinates are being saved in a temporary storage array called T. T is a three dimensional array which holds the X, Y and P values (first subscript) for each coordinate group (second subscript) of the figure, as well as the page it is drawn upon (third subscript). This last parameter is the current value of FLIP. Return then transfers control back to the main program.

Back in the main program the completed image (on page 2) is displayed. Note that this is accomplished with a POKE to the appropriate address: DISPLAY + 1. FLIP is toggled from 1 to 0 and the graphics page index is changed to page 1. Again, since this is still the first time through the interpolation and drawing loop, there is nothing to undraw on page 1. PCT is calculated for the second frame: note the slight variation in the algorithm. The DRAW/SAVE subroutine is called for a second time but this time FLIP has changed. The DRAW/SAVE subroutine now draws the next (in this case, second) transitional image in background simultaneously storing these coordinates in the array T. Remember the third subscript is determined by the value of FLIP. Control is then returned to the main program.

Back in the main program the completed image (on page 1) is displayed. Note: this is accomplished with a POKE to the appropriate address: DISPLAY. However, the bottom of the loop has been reached and control is transferred to the top of the loop. The text window at the bottom of page one is re-closed. This time the first-time-through-the-loop test is false and the instructions that were passed over before are executed. These instructions are: FLIP is changed back to 1, and the graphics page index is reset to page 2. HCOLOR is set to 4 or black. Thus, undrawing is accomplished by drawing over the previous image using the values stored in the array T with a black line, effectively erasing the image. Once undrawn, a new interpolation fraction is calculated and the next transitional image is generated on the just cleared page 2 by calling the DRAW/SAVE subroutine. Notice that HCOLOR is set back to 3, or white, early on in this routine.

Needless to say, this process continues until the interpolation sequence is complete. The end of the sequence is signalled by a beep. The last image is held on the screen until the return key is depressed.

Conclusion

All aspects of keyframe animation have been included in the animation software presented above. This software will perform animation if the data is correctly formulated. It is up to the animator to establish the subject of the animation and design the imagery. It is strongly suggested that any sequence be previewed using one of the first two programs prior to filming. Ideally, the animation program(s) should have a built in mechanism to trigger the animation camera. A subroutine that uses the annunciator outputs, which themselves are controlled by soft switches, is the clue. (See the Apple II Reference Manual, Apple Product No. A2L0001A, p. 23.)

Keyframe animation is a simple way to begin using the power of the microcomputer to generate animated films. Animation provides a working knowledge of the time dimension. This software is not intended to generate real-time moving imagery. It may take hours of filming to produce seconds or minutes of film. However, the effort is its own reward when the imagery comes to life on the silver screen.

Listings

Listing 1

```
100  REM  <<<<< LINEAR INTERPOLATION >>>>>
110  REM  <<< TRANSLATE AND TRANSFORM BETWE
EN TWO FIGURES >>>
120  REM
130  REM  COPYRIGHT 1983 W.J.KOLOMYJEC
140  REM
150  REM  GET GEOMETRIC FIGURES DATA
160  GOSUB 5000
170  REM  DEFINE MANIPULATION ARRAYS
180  DIM Q(1,N),Z(1,N)
190  REM  ASSIGN DATA TO MANIPULATION ARRAY
S
200  REM  TRANSLATE, SCALE & COUNTER TRANSL
ATE IN SUBROUTINE
210  GOSUB 6000
220 NFRAMES = 12
230  HGR2 : HCOLOR= 3: REM  INITIALIZE GRAP
HICS
240  GOSUB 1250: REM  OUTLINE SCREEN
250  FOR K = 1 TO NFRAMES
260  REM  UNIFORM INTERVALS
270  REM  PCT RANGE: 0 TO 1
280 PCT = (K - 1) / (NFRAMES - 1)
290  FOR J = 0 TO N
300  REM  INTERPOLATION EQUATIONS
310 X = (Z(0,J) - Q(0,J)) * PCT + Q(0,J)
320 Y = (Z(1,J) - Q(1,J)) * PCT + Q(1,J)
330  IF J = 0 THEN 350
340 P = 1: GOTO 360
350 P = 2
360  GOSUB 1000
370  NEXT J
380  NEXT K
390  INPUT A$: TEXT : REM  TERMINATE
999  END
1000  REM  <<<<< PLOTSUBB >>>>>
1010  REM  PARAMETERS: X,Y AND P
1020  REM  P VALUE IS BEAM CONTROL: 1=DRAW,
 2=MOVE
1030  REM  FLIP Y COORD. AND CORRECT ASPECT
 RATIO (0.881)
1040  REM  PLOT AREA: 0<=X<=279,0<=Y<=217
1050 Y9 = 192 - (Y * 0.881 + 0.5)
1060  IF P = 1 THEN  GOTO 1100
1070  IF P <  > 2 THEN  PRINT "PEN ERROR":
STOP
1080  HPLOT X,Y9
1090  RETURN
1100  HPLOT  TO X,Y9
1110  RETURN
1250  REM  <<<<< OUTLINE SCREEN >>>>>
1260  HPLOT 0,0 TO 279,0 TO 279,191 TO 0,19
1 TO 0,0
1270  RETURN
5000  REM  <<<<< FOUR GEOMETRIC FIGURE DATA
 SUBROUTINE >>>>>
5010  HOME : VTAB 10: PRINT "FILLING DATA A
RRAYS..."
5020  REM
5030 NUM = 17
5040 N = NUM - 1
5050  DIM A(1,N),B(1,N),C(1,N),D(1,N)
5060  REM  READ DATA INTO ARRAYS
5070  FOR J = 0 TO N
5080  READ A(0,J),A(1,J),B(0,J),B(1,J),C(0,
J),C(1,J),D(0,J),D(1,J)
5090  NEXT J
5100  REM  GEOMETRIC FIGURE DATABASE
5110  DATA  7,5,7,5,10,5,9,5
5120  DATA  10,9,6,8,8,6,7,10
5130  DATA  6,6,10,10,10,10,5,5
5140  DATA  9,10,8,6,6,8,10,7
5150  DATA  5,7,5,7,5,10,5,9
5160  DATA  1,10,2,6,4,8,0,7
5170  DATA  4,6,0,10,0,10,5,5
5180  DATA  0,9,4,8,2,6,3,10
5190  DATA  3,5,3,5,0,5,1,5
5200  DATA  0,1,4,2,2,4,3,0
5210  DATA  4,4,0,0,0,0,5,5
5220  DATA  1,0,2,4,4,2,0,3
5230  DATA  5,3,5,3,5,0,5,1
5240  DATA  9,0,8,4,6,2,10,3
5250  DATA  6,4,10,0,10,0,5,5
5260  DATA  10,1,6,2,8,4,7,0
5270  DATA  7,5,7,5,10,5,9,5
5280  RETURN
6000  REM  <<<<< MANIPULATE DATA >>>>>
6010  HOME : VTAB 10: PRINT "MANIPULATING D
ATA..."
6020  REM  FIGURES TO BE TRANSLATED ABOUT O
RIGIN AND SCALED
6030  REM  THEN COUNTER TRANSLATED TO SCREE
N LOCATIONS
6040  REM  OFFSETS TO CENTER IMAGES
6050 XOFF =  - 5:YOFF =  - 5
6060  REM  DEFINE IMAGE SCALE FACTORS (VARI
ABLE)
6070 S1 = 3:S2 = 4
6080  REM  DEFINE COUNTER TRANSLATION COORD
INATES
6090 X1 = 15:Y1 = 200
6100 X2 = 254:Y2 = 25
6110  REM  FIGURE ASSIGNMENTS CAN BE ALTERE
D HERE
```

```
6120  FOR J = 0 TO N
6130 Q(0,J) = (B(0,J) + XOFF) * S1 + X1
6140 Q(1,J) = (B(1,J) + YOFF) * S1 + Y1
6150 Z(0,J) = (D(0,J) + XOFF) * S2 + X2
6160 Z(1,J) = (D(1,J) + YOFF) * S2 + Y2
6170  NEXT J
6180  RETURN
```

Listing 2

```
100  REM  <<<<< QUADRATIC INTERPOLATION >>>
>>
110  REM  <<< TRANSLATE AND TRANSFORM BETWE
EN THREE FIGURES >>>
120  REM
130  REM  COPYRIGHT 1983 W.J.KOLOMYJEC
140  REM
150  REM  GET GEOMETRIC FIGURES DATA
160  GOSUB 5000
170  REM  DEFINE MANIPULATION ARRAYS
180  REM  M(FIGURE,COORD,NUMPTS)
190  DIM M(2,1,N)
200  REM  ASSIGN DATA TO MANIPULATION ARRAY
S
210  REM  TRANSLATE, SCALE & COUNTER TRANSL
ATE IN SUBROUTINE
220  GOSUB 6000
230  REM  TO GET ALL THREE FIGURES NFRAMES
MUST BE ODD
240 NFRAMES = 15
250  HGR2 : HCOLOR= 3: REM  INITIALIZE GRAP
HICS
260  GOSUB 1250: REM  OUTLINE SCREEN
270  FOR K = 1 TO NFRAMES
280  REM  UNIFORM INTERVALS
290  REM  PCT RANGE: -1 TO +1
300 PCT = (K - 1) * 2 / (NFRAMES - 1) - 1
310  FOR J = 0 TO N
320 F1 = M(0,0,J):F2 = M(1,0,J):F3 = M(2,0,
J)
330  GOSUB 3000:X = QNTRP
340 F1 = M(0,1,J):F2 = M(1,1,J):F3 = M(2,1,
J)
350  GOSUB 3000:Y = QNTRP
360  IF J = 0 THEN 380
370 P = 1: GOTO 390
380 P = 2
390  GOSUB 1000
400  NEXT J
410  NEXT K
420  INPUT A$: TEXT : REM  TERMINATE
999  END
1000  REM  <<<<< PLOTSUBB >>>>>
1010  REM
1020  REM  ***** SAME AS LISTING 1 *****
1030  REM
1110  RETURN
1250  REM  <<<<< OUTLINE SCREEN >>>>>
1260  HPLOT 0,0 TO 279,0 TO 279,191 TO 0,19
1 TO 0,0
1270  RETURN
3000  REM  <<<<< FUNCTION QNTRP >>>>>
3010  REM  SIMULATES FUNCTION QNTRP (PCT,F1
,F2,F3)
3020  REM  QUADRATIC INTERPOLATION
3030  REM  QNTRP IS A FUNCTION WHICH DETERM
INES THE PLOT COORDINATE
3040  REM  FOR ANY PCT VALUE (-1 TO +1) FOR
 A QUADRATIC FUNCTION, THUS
3050  REM  QNTRP(-1)=FIGURE 1, QNTRP(0)=FIG
URE 2, QNTRP(+1)=FIGURE 3
3060 A = (F1 + F3 - 2 * F2) / 2
3070 B = (F3 - F1) / 2
3080 C = F2
3090 QNTRP = A * PCT * PCT + B * PCT + C
3100  RETURN
5000  REM  <<<<< FOUR GEOMETRIC FIGURE DATA
 SUBROUTINE >>>>>
5010  REM
5120  REM  ***** SAME AS LISTING 1 *****
5130  REM
5280  RETURN
6000  REM  <<<<< MANIPULATE DATA >>>>>
6010  HOME : VTAB 10: PRINT "MANIPULATING D
ATA..."
6020  REM  FIGURES TO BE TRANSLATED ABOUT O
RIGIN AND SCALED
6030  REM  OFFSETS TO CENTER IMAGES
6040 XOFF =  - 5:YOFF =  - 5
6050  REM  IMAGE SCALE FACTORS
6060 SFTR = 3
6070  REM  DEFINE IMAGE TRANSLATION COORDIN
ATES (SCREEN LOCATIONS)
6080 X1 = 20:Y1 = 20
6090 X2 = 140:Y2 = 200
6100 X3 = 259:Y3 = 20
6110  REM  FIGURE ASSIGNMENTS CAN BE ALTERE
D HERE
6120  FOR J = 0 TO N
6130 M(0,0,J) = (A(0,J) + XOFF) * SFTR + X1

6140 M(0,1,J) = (A(1,J) + YOFF) * SFTR + Y1

6150 M(1,0,J) = (B(0,J) + XOFF) * SFTR + X2

6160 M(1,1,J) = (B(1,J) + YOFF) * SFTR + Y2

6170 M(2,0,J) = (C(0,J) + XOFF) * SFTR + X3

6180 M(2,1,J) = (C(1,J) + YOFF) * SFTR + Y3

6190  NEXT J
6200  RETURN
```

Listing 3

```
100  REM  <<<<< KEY FRAME ANIMATION >>>>>
110  REM  <<< BASED ON PAGE FLIPPING  >>>
115  REM
120  REM  <<< LINEAR INTERPOLATION SEQUENCE
 >>>
125  REM
130  REM  COPYRIGHT 1983 W. J. KOLOMYJEC
140  REM
150  REM  DEFINE CRITICAL ADDRESSES AND VAL
UES
160 SWITCH = 230:DISPLAY = 49236
170 PAGE(0) = 32:PAGE(1) = 64
```

```
180  REM  INITIALIZE BOOLEAN VARIABLE FLIP
190 FLIP = 1
200  REM  GET GEOMETRIC FIGURES DATA
210  GOSUB 5000
220  REM  DEFINE MANIPULATION ARRAYS
230  DIM Q(1,N),Z(1,N),T(2,N,1)
240  REM  ASSIGN DATA TO MANIPULATION ARRAY
S
250  GOSUB 6000
260  REM  DEFINE DURATION OF IMAGE TRANSFOR
MATION
270  REM  24 FRAMES EQUALS 1 SECOND OF ANIM
ATION
280 NFRAMES = 24
290  REM  ***** BEGIN ANIMATION CODE *****
300  REM  CLEAR BOTH GRAPHICS PAGES
310  HGR2 : HGR
320  REM  DISPLAY PAGE 1
330  POKE DISPLAY,0
340  REM  SWITCH PAGE INDEX TO PAGE 1
350  POKE SWITCH,PAGE(FLIP)
360  FOR K = 1 TO NFRAMES STEP 2
370  REM  CLOSE TEXT WINDOW FOR FULL SCREEN
 GRAPHICS
380  POKE 49232,0: POKE 49234,0
390  REM  UNDRAW  (SKIP FIRST TIME)
400  IF K = 1 THEN 510
410 FLIP =  NOT FLIP: REM  0 --> 1
420  REM  SWITCH PAGE INDEX TO PAGE 2
430  POKE SWITCH,PAGE(FLIP)
440  HCOLOR= 4
450  REM  UNDRAW PAGE 2
460  FOR L = 0 TO N
470 X = T(0,L,FLIP):Y = T(1,L,FLIP):P = T(2
,L,FLIP)
480  GOSUB 1000
490  NEXT L
500  REM  PCT IS THE INTERPOLATION FRACTION

510 PCT = (K - 1) / (NFRAMES - 1)
520  GOSUB 2000: REM  DRAW/SAVE
530  REM  DISPLAY PAGE 2
540  POKE DISPLAY + FLIP,0
550 FLIP =  NOT FLIP: REM   1 --> 0
560  REM  SWITCH PAGE INDEX TO PAGE 1
570  POKE SWITCH,PAGE(FLIP)
580  REM  UNDRAW (SKIP FIRST TIME)
590  IF K = 1 THEN 660
600  HCOLOR= 4
610  REM  UNDRAW PAGE 1
620  FOR L = 0 TO N
630 X = T(0,L,FLIP):Y = T(1,L,FLIP):P = T(2
,L,FLIP)
640  GOSUB 1000
650  NEXT L
660 PCT = K / (NFRAMES - 1)
670  GOSUB 2000: REM  DRAW/SAVE
680  REM  DISPLAY PAGE 1
690  POKE DISPLAY,0
700  NEXT K
710  REM  ***** END ANIMATION CODE *****
720  REM  BEEP WHEN DONE
730  PRINT  CHR$ (7)
740  REM  HOLD IMAGE ON SCREEN UNTIL RETURN
 IS PRESSED
750  INPUT A$: TEXT : REM  TERMINATE GRAPHI
CS
999  END
```

```
1000  REM  <<<<< PLOTSUBB >>>>>
1010  REM
1020  REM  ***** SAME AS LISTING 1 *****
1030  REM
1110  RETURN
2000  REM  <<<<< DRAW/SAVE ONE IMAGE >>>>>
2010  HCOLOR= 3
2020  FOR J = 0 TO N
2030  REM  INTERPOLATION EQUATIONS
2040 X = (Z(0,J) - Q(0,J)) * PCT + Q(0,J)
2050 Y = (Z(1,J) - Q(1,J)) * PCT + Q(1,J)
2060  IF J = 0 THEN 2080
2070 P = 1: GOTO 2090
2080 P = 2
2090  GOSUB 1000
2100  REM  STORE IMAGE COORDINATES FOR UNDR
AWING LATER
2110  REM  USE FLIP TO KEEP TRACK
2120 T(0,J,FLIP) = X:T(1,J,FLIP) = Y:T(2,J,
FLIP) = P
2130  NEXT J
2140  RETURN
5000  REM  <<<<< FOUR GEOMETRIC FIGURE DATA
 SUBROUTINE >>>>>
5010  REM
5120  REM  ***** SAME AS LISTING 1 *****
5130  REM
5280  RETURN
6000  REM  <<<<< MANIPULATE DATA >>>>>
6010  REM
6020  REM  ***** SAME AS LISTING 1 *****
6030  REM
6180  RETURN
```

Listing 4

```
100  REM  <<<<< KEY FRAME ANIMATION >>>>>
110  REM  <<< BASED ON PAGE FLIPPING  >>>
115  REM
120  REM  <<< QUADRATIC INTERPOLATION SEQUE
NCE >>>
125  REM
130  REM  COPYRIGHT 1983 W. J. KOLOMYJEC
140  REM
150  REM  DEFINE CRITICAL ADDRESSES AND VAL
UES
160 SWITCH = 230:DISPLAY = 49236
170 PAGE(0) = 32:PAGE(1) = 64
180  REM  INITIALIZE BOOLEAN VARIABLE FLIP
190 FLIP = 1
200  REM  GET GEOMETRIC FIGURES DATA
210  GOSUB 5000
220  REM  DEFINE MANIPULATION ARRAYS
225  REM  NOTE: M(FIGURE,COORD,NUM), T(COOR
D,NUM,PAGE)
230  DIM M(2,1,N),T(2,N,1)
240  REM  ASSIGN DATA TO MANIPULATION ARRAY
S
250  GOSUB 6000
260  REM  DEFINE DURATION OF IMAGE TRANSFOR
MATION
270  REM  24 FRAMES EQUALS 1 SECOND OF ANIM
ATION
280 NFRAMES = 24
```

```
290  REM  ***** BEGIN ANIMATION CODE *****
300  REM  CLEAR BOTH GRAPHICS PAGES
310  HGR2 : HGR
320  REM  DISPLAY PAGE 1
330  POKE DISPLAY,0
340  REM  SWITCH PAGE INDEX TO PAGE 1
350  POKE SWITCH,PAGE(FLIP)
360  FOR K = 1 TO NFRAMES STEP 2
370  REM  CLOSE TEXT WINDOW FOR FULL SCREEN
 GRAPHICS
380  POKE 49232,0: POKE 49234,0
390  REM  UNDRAW  (SKIP FIRST TIME)
400  IF K = 1 THEN 510
410 FLIP =  NOT FLIP: REM  0 --> 1
420  REM  SWITCH PAGE INDEX TO PAGE 2
430  POKE SWITCH,PAGE(FLIP)
440  HCOLOR= 4
450  REM  UNDRAW PAGE 2
460  FOR L = 0 TO N
470 X = T(0,L,FLIP):Y = T(1,L,FLIP):P = T(2
,L,FLIP)
480  GOSUB 1000
490  NEXT L
500  REM  PCT IS THE INTERPOLATION FRACTION

510 PCT = (K - 1) * 2 / (NFRAMES - 1) - 1
520  GOSUB 2000: REM  DRAW/SAVE
530  REM  DISPLAY PAGE 2
540  POKE DISPLAY + FLIP,0
550 FLIP =  NOT FLIP: REM  1 --> 0
560  REM  SWITCH PAGE INDEX TO PAGE 1
570  POKE SWITCH,PAGE(FLIP)
580  REM  UNDRAW (SKIP FIRST TIME)
590  IF K = 1 THEN 660
600  HCOLOR= 4
610  REM  UNDRAW PAGE 1
620  FOR L = 0 TO N
630 X = T(0,L,FLIP):Y = T(1,L,FLIP):P = T(2
,L,FLIP)
640  GOSUB 1000
650  NEXT L
660 PCT = K * 2 / (NFRAMES - 1) - 1
670  GOSUB 2000: REM  DRAW/SAVE
680  REM  DISPLAY PAGE 1
690  POKE DISPLAY,0
700  NEXT K
710  REM  ***** END ANIMATION CODE *****
720  REM  BEEP WHEN DONE
730  PRINT  CHR$ (7)
740  REM  HOLD IMAGE ON SCREEN UNTIL RETURN
 IS PRESSED
750  INPUT A$: TEXT : REM  TERMINATE GRAPHI
CS
999  END

1000  REM  <<<<< PLOTSUBB >>>>>
1010  REM
1020  REM  ***** SAME AS LISTING 1 *****
1030  REM
1110  RETURN
2000  REM  <<<<< DRAW/SAVE ONE IMAGE >>>>>
2010  HCOLOR= 3
2020  FOR J = 0 TO N
2030  REM  USE QNTRP FUNCTION
2035 F1 = M(0,0,J):F2 = M(1,0,J):F3 = M(2,0
,J)
2040  GOSUB 3000:X = QNTRP
2045 F1 = M(0,1,J):F2 = M(1,1,J):F3 = M(2,1
,J)
2050  GOSUB 3000:Y = QNTRP
2060  IF J = 0 THEN 2080
2070 P = 1: GOTO 2090
2080 P = 2
2090  GOSUB 1000
2100  REM  STORE IMAGE COORDINATES FOR UNDR
AWING LATER
2110  REM  USE FLIP TO KEEP TRACK
2120 T(0,J,FLIP) = X:T(1,J,FLIP) = Y:T(2,J,
FLIP) = P
2130  NEXT J
2140  RETURN
3000  REM  <<<<< FUNCTION QNTRP >>>>>
3010  REM
3020  REM  ***** SAME AS LISTING 2 *****
3030  REM
3100  RETURN
5000  REM  <<<<< FOUR GEOMETRIC FIGURE DATA
 SUBROUTINE >>>>>
5010  REM
5120  REM  ***** SAME AS LISTING 1 *****
5130  REM
5280  RETURN
6000  REM  <<<<< MANIPULATE DATA >>>>>
6010  REM
6020  REM  ***** SAME AS LISTING 2 *****
6030  REM
6180  RETURN
```

PIPEDREAMS, A COMPLETE CAD CAM SYSTEM FOR TUBULAR SCULPTURES

Frank M. Smullin

Art Department, Duke University, Durham, N. C.

ABSTRACT

The rapidly increasing power of microcomputers has made it possible for artists working with small budgets, outside of high tech installations to have the benefit of computer-aided design and manufacturing technology. This paper presents the application of CAD CAM solid modeling to sculpture made of cylinders. The database and the computer-aided operations are based on a simple vector model of cylinders. The methodology is sufficiently general to suggest many other applications.

BACKGROUND

For the past eight years I have been developing a computer-aided design and fabrication system for sculptures made of mitered cylinders. I call this echnique Analytic Constructivism. Its latest mainfestation,named PIPEDREAMS, is run on a Tektronix 4052 desktop computer (16 bit, 64K). The evolution of my techniques parallels the ongoing revolution in microelectronics and the resulting accesibility of computers.

Sculptors working with modular materials have been constrained in their plastic manipulations by the difficulties of accomodating one elemment to another. Joining them is difficult and modifying the structure, once made, is harder. Ideas once materialized are hard to change. My particular experience in this area is creating structures with cylinders, all of the same diameter, mitered into complex intersections. The problems are similar for prismatic, planar or polyhedral modules; they are all hard to prepare for joining by purely visual inspection and manual practice. However, the uniformity of shape in the chosen element makes it possible to devise general, precise and quantitative means for manipulating, cutting, and joining the 3-d arrangements, using mathematical functions to determine or alter the arrangements. Many of the functions that one would use are independent of the particular form and have more to do with the nature of 3-d space itself. While the functions are relatively few, the sequence in which they are applied can become quite complex. Prior to the advent of electronic computing, the tedious computational chore ruled out a computational approach to complex designs, making many conceptual projects physically impossible. Other sculptors who are making new things possible include Ruth Leavitt[3], Robert Mallory and Ron Resch[2,5,6].

It was in order to construct DAEDALUS (fig. 1), a commissioned 27 foot high sculpture, in 1974, that I worked out and demonstrated the feasibility of a quantitative approach to analyzing and constructing tubular structures. The calculations for that first sculpture, done with a slide rule, a trig table and -occassionaly- a desktop calculator took 300 hours. Despite this discouraging chore, the success of the sculpture was sufficient to impell me to find a better way to do the quantitative analysis that had made it possible. Working with a student programmer, I discovered that the methods I had developed to organize reams of computations constituted at least half of the task of writing a computer program. When encoded as a BASIC program running on a PDP-11, the same analysis instead of taking 300 hours (with many errors!) was accomplished and printed in a neat format

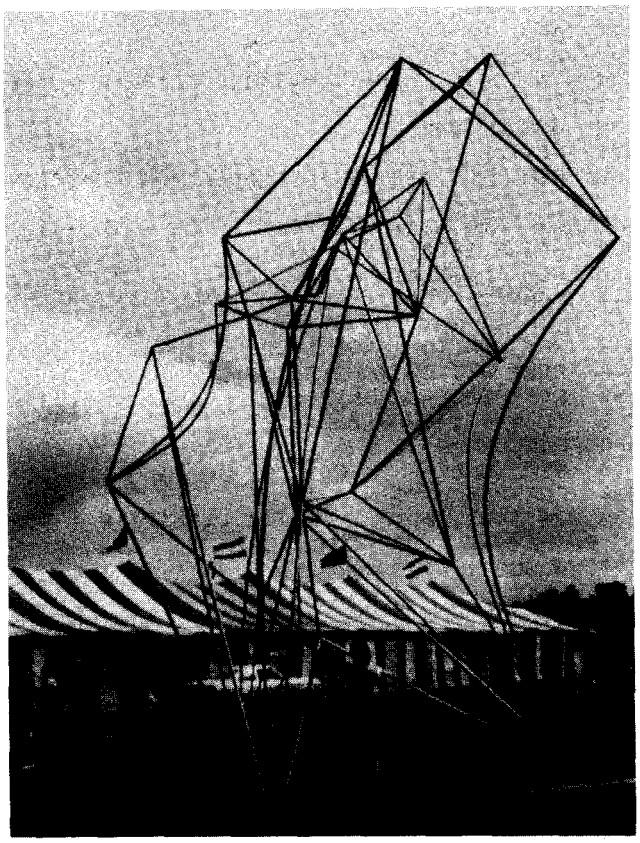


fig. 1: DAEDALUS, 1974, iron pipe, 27' high.

CH1930-7/83/0000/0112$01.00 © 1983 IEEE

in about 3 minutes. Such computational wonders are now entirely commonplace and the sculptures that I am now making are only one of a multitude of things that were quite impossible a few years ago.

From that first program, CONPIPES, my method has evolved as I began to use it, becoming more familiar with this quantitative approach to design and as I came to articulate more precisely the kind of information and the kinds of forms I wanted. Especially since beginning to write my own programs, in 1978, the complexity of my system and of the resulting plastic freedom in the design of sculptures has continued to grow.

Underlying Analytic Constructivism is the conceptual model of the component cylider. This geometric form is abstracted as what I call a fat vector, an axial vector with a perpendicular radius locating its surface. A network of cylinders, then can be described by listing the end-points or intersection points of the cylinders, and listing the interconnections among these points. Thus, the structure of cylinders is just a "wire-frame" model made with fat wires. Thus, the description does not explicitly involve any surface or edge informations. Rather, by using a solid modeling approach, these features are calculated as implications of the relationships of the solid component cylinders and, in the case of perspective drawing, as the relative implication of the viewpoint. This method with its efficient database lets the designer work with very complex structures on a small computer.

This abstraction came about quite naturally from my desire to enlarge the small welding rod model of DAEDALUS. The form of these space-frames was, itself, nearly a vector object, the diameter of the rods being so small as to be visually, and practically, inconsequential. The transparency of the object also facilitated the projection of the points onto graph paper in a manual digitizing technique that sculptors know as "pointing" or "plumbing."

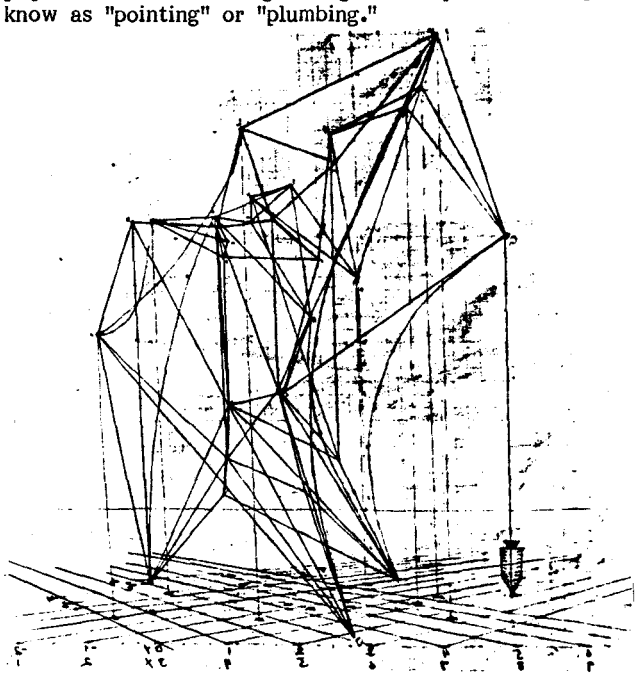
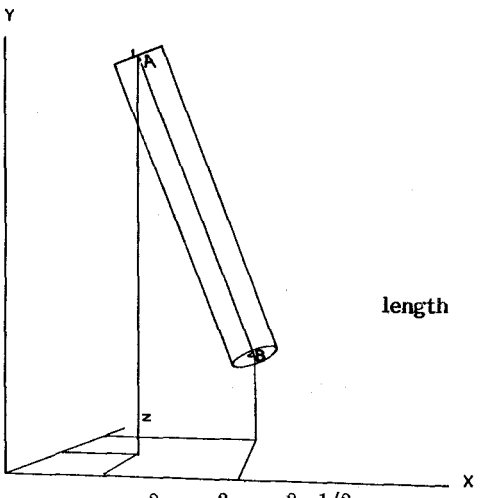
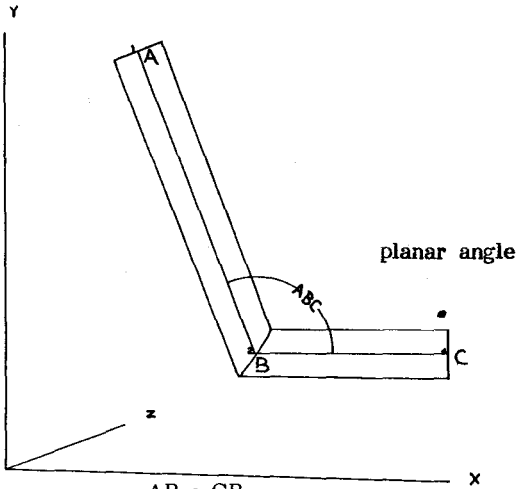


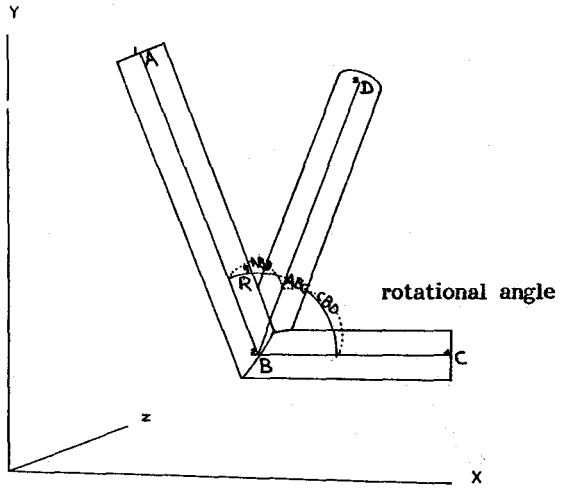
fig. 2: Illustration of plumb-line pointing technique for digitizing a wire model.



$$|AB| = (AB_x^2 + AB_y^2 + AB_z^2)^{1/2} \qquad \text{eq.(1)}$$



$$\cos(ABC) = \frac{AB \cdot CB}{|AB| * |CB|} \qquad \text{eq.(2)}$$



$$\cos(R) = \frac{\cos(CBD) - \cos(ABC) * \cos(ABD)}{\sin(ABC) * \sin(ABD)} \qquad \text{eq.(3)}$$

Fig. 3: The geometric relationships between connected components of a tubular network.

While a list of coordinates, connections and radius uniquely defines the object, it is not sufficient information from which to build the structure when enlarged to any significant diameter (greater than 1/4 inch). However, using analytic geometry and vector algebra, it is possible to obtain, from this simple description, all the information necessary to build the structure at any scale. Actually, the first sculpture for which I used any digitizing analysis was **ONE FORM**, 1968. To build that 15 ft. high pipe sculpture,from the wire model, I realized that I could "point" the intersections and use these coordinates to calculate the lengths using the Pythagorian Theorem,fig. 3, eq. (1). However, in the course of building this piece, I discovered that it was necessary to know more information about the geometry of the intersections to properly shape the pipes for their complex meetings. It was only when I returned to this problem in 1974 that I worked out the other two fundamental relationships as functions of the point coordinates. These are: the angle between a pair of pipes, eq. (2); and the rotational angle around one pipe between two others joined to it, eq. (3). These geometric relationship and equations are shown in fig. 3.

The spatial relations described by these functions manifest themselves in the elegant, complex traceries of the mitered intersections of pipes. The translation from abstract numbers to this concrete form is made possible by an efficient reiterative, recursive algorithm using the above functions to generate parameters for each pipe and by the use of this data in plotting procedures that create templates for cutting the pipes.

My early work, 1974-1978, utilized only this limited analysis, simply translating a digitized model into a tubular rendering. However,, since 1979, I have made more extensive use of analytic geometry to devise ways of precisely modifying and rendering views of the model.

THE METHOD

Following is an outline of the design and fabrication process followed in making an Analytic Constructivist sculpture using **PIPEDREAMS.** In its current form it is written in TEKTRONIX 4050 BASIC. Much of it was previously written in PL/I and in BASIC+ for other installations. I am in the process of rewriting it in PASCAL/Z to run on the Vector 4 using raster graphics.

I. DESIGN OF THE INITIAL MODEL

The sculpture can start as a wire-frame or toothpick model, a drawing, or a set of coordinates derived , for instance, from topological considerations, or rudimentary drawings. Recently, my models have been of the digital type as a result the improved interactive modes that I have developed for inputting and modifying the coordinate descriptions and as I have become more comfortable thinking my way, quantitatively, around 3-d space and because of the relative difficulty of actually building and digitizing a preliminary physical model of my designs.

II.TRANSLATION OF THE MODEL INTO DATA STRUCTURE

The model is translated into the coordinate-connection-radius description. PIPEDREAMS lets the sculptor enter the coordinates in a flexible manner. They can be specified numerically as cartesian coordinates, either absolute, or relative to some other specified points. They can also be input graphically from a tablet or plotter, making it easy to digitize directly off a freehand drawing. In my current installation, the third dimension must be specified numerically by the sculptor. Obviously, a 3-d digitizing device could be incorporated as a useful peripheral. The sculptor can intermix coordinate and connection specifications according to a flexible format that can be shaped with the user-defined function calls to subroutines.

The arrays holding coordinate and connection data are dynamically dimensioned, stretched or compressed according to actual need to maximize memory needed for large temporary arrays needed, particularly during the drawing routine. With a chosen radius, the description is complete. The data base used for all the segments of **PIPEDREAMS** consiststs of:

1. title
. number of points
3. greatest number of connections at any point
4. coordinates of all intersection or terminal points
5. connections between the points
6. coordinates of viewpoint, center of view and vertical orientation last used
7. the radius of the cylinders.

This data is stored in a sequential file and can be read or written from the program.

POINT	COORDINATES			CONNECTIONS
1	1.00	1.00	1.00	2 4 5
2	1.00	1.00	-1.00	1 3 4 5 6 7 9
3	1.00	-1.00	-1.00	2 4 7
4	1.00	-1.00	1.00	1 2 3 5 7 8 9
5	-1.00	1.00	1.00	1 2 4 6 7 8 9
6	-1.00	1.00	-1.00	2 5 7
7	-1.00	-1.00	-1.00	2 3 4 5 6 8 9
8	-1.00	-1.00	1.00	4 5 7
9	0.00	0.00	0.00	2 4 5 7

```
CENTER OF MASS : 0 0 0
VPT: 0.6 1 -4
DOWN: 0.6 0 -4
COV : 0 0 0
RADIUS: 0.05
```

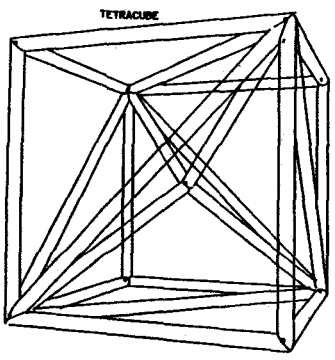
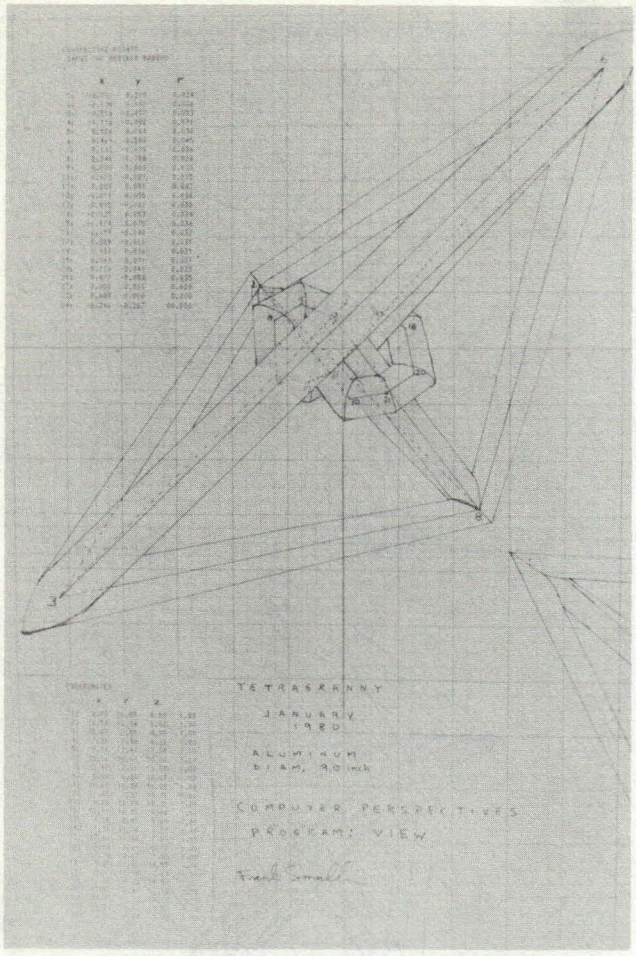
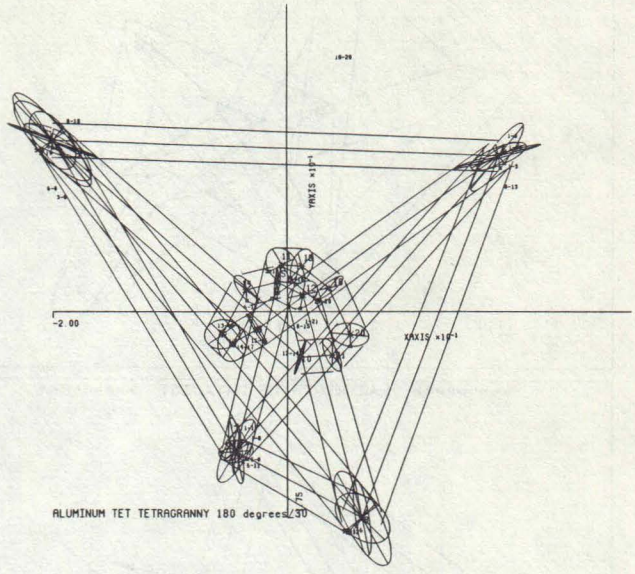


Fig. 4: Database description of and perspective rendering of TETRACUBE.





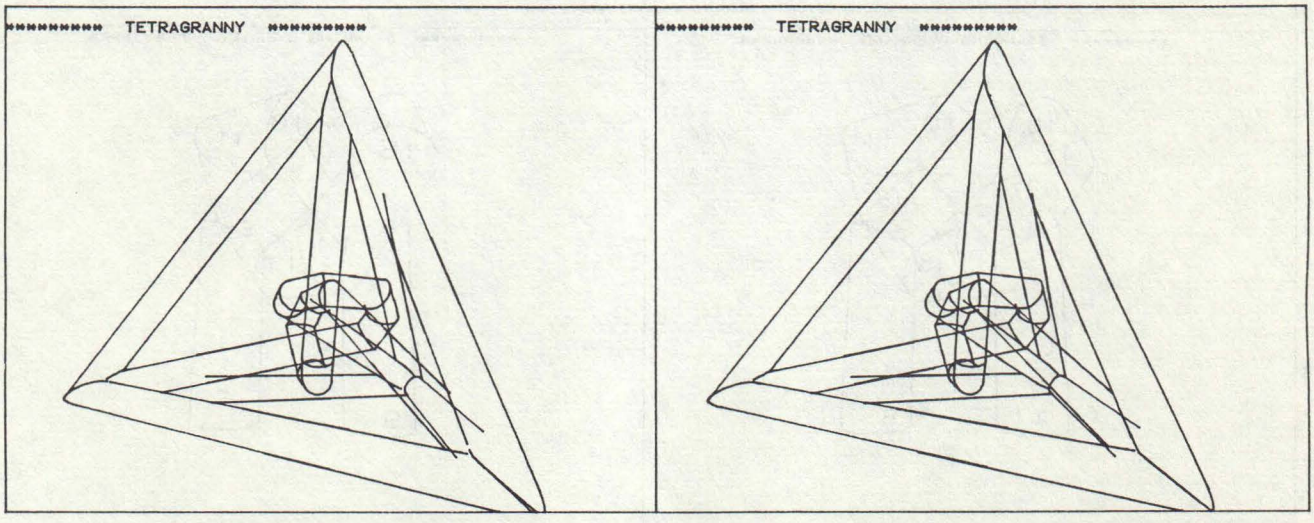
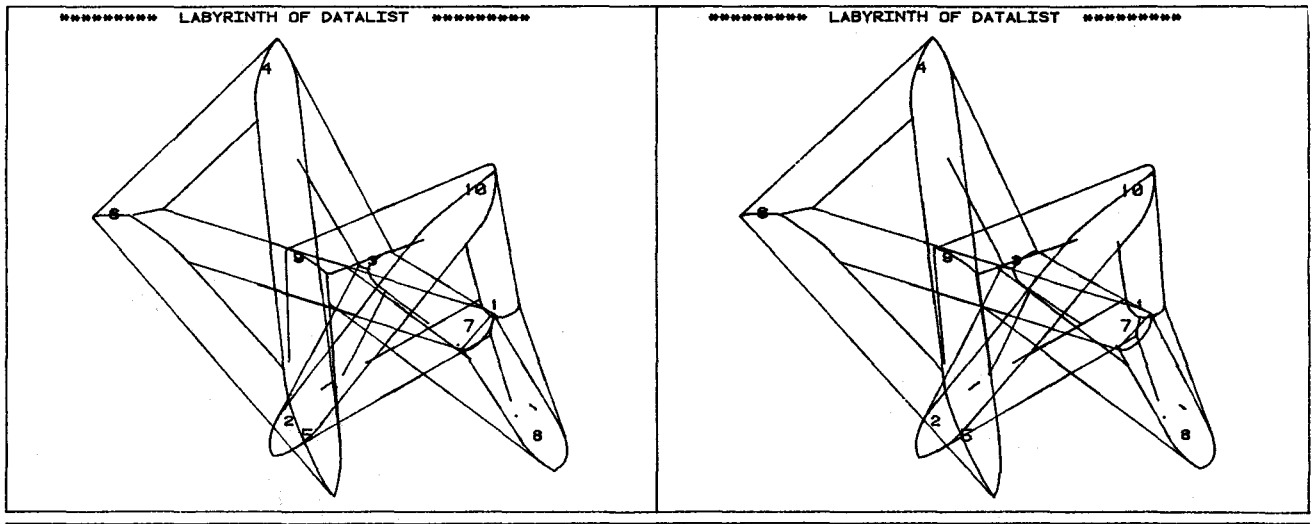
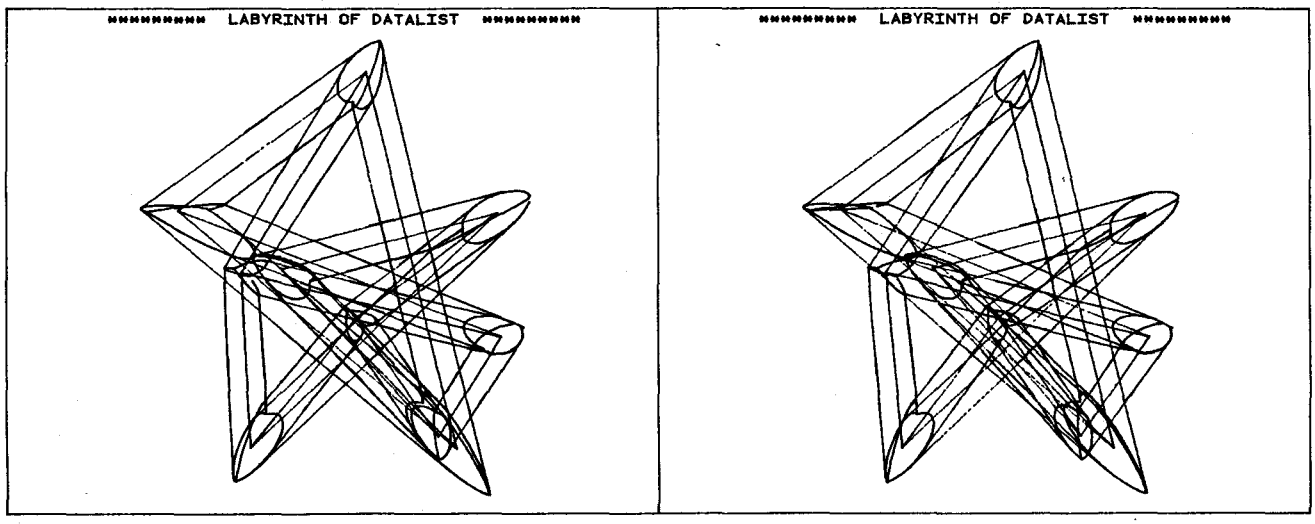
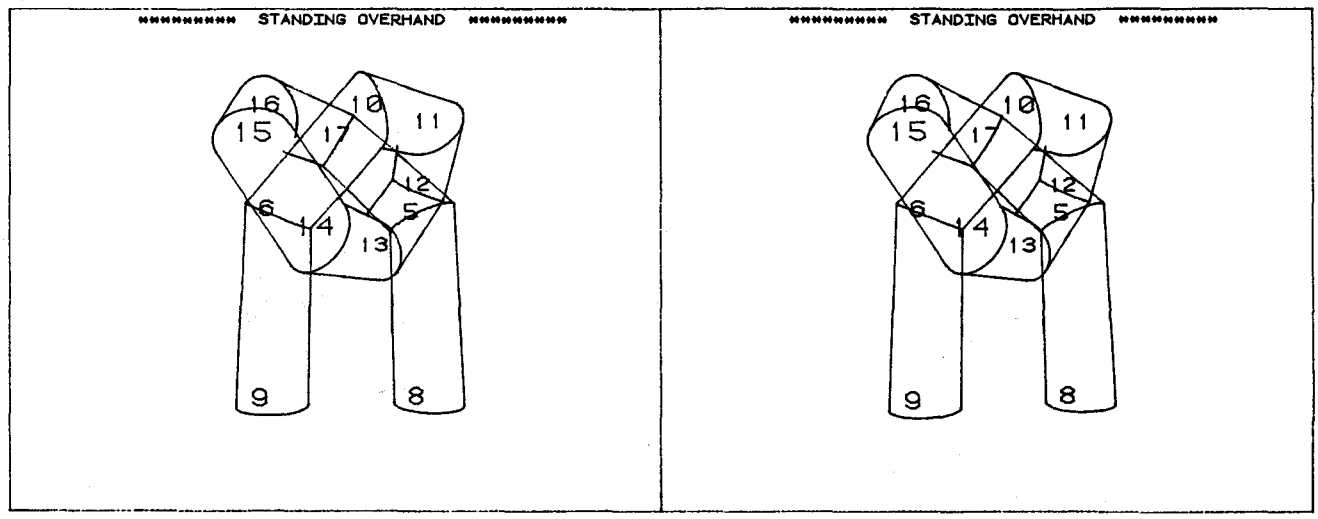


fig. 5: The evolution of PIPEDREAM graphics: a) Jan. 1980, hand-plotted perspectives from numeric output, free-hand ellipses at intersections; b) Sept. 1980, PLOT 10, Calcomp plot, ellipses calculated, each intersection plane drawn complete; c) Oct. 1982, Tektronix PLOT 50 stereo plot (cross-eyed), only the front surfaces of composite remnant of intersection ellipses drawn and edges are hidden where they enter the clusters.

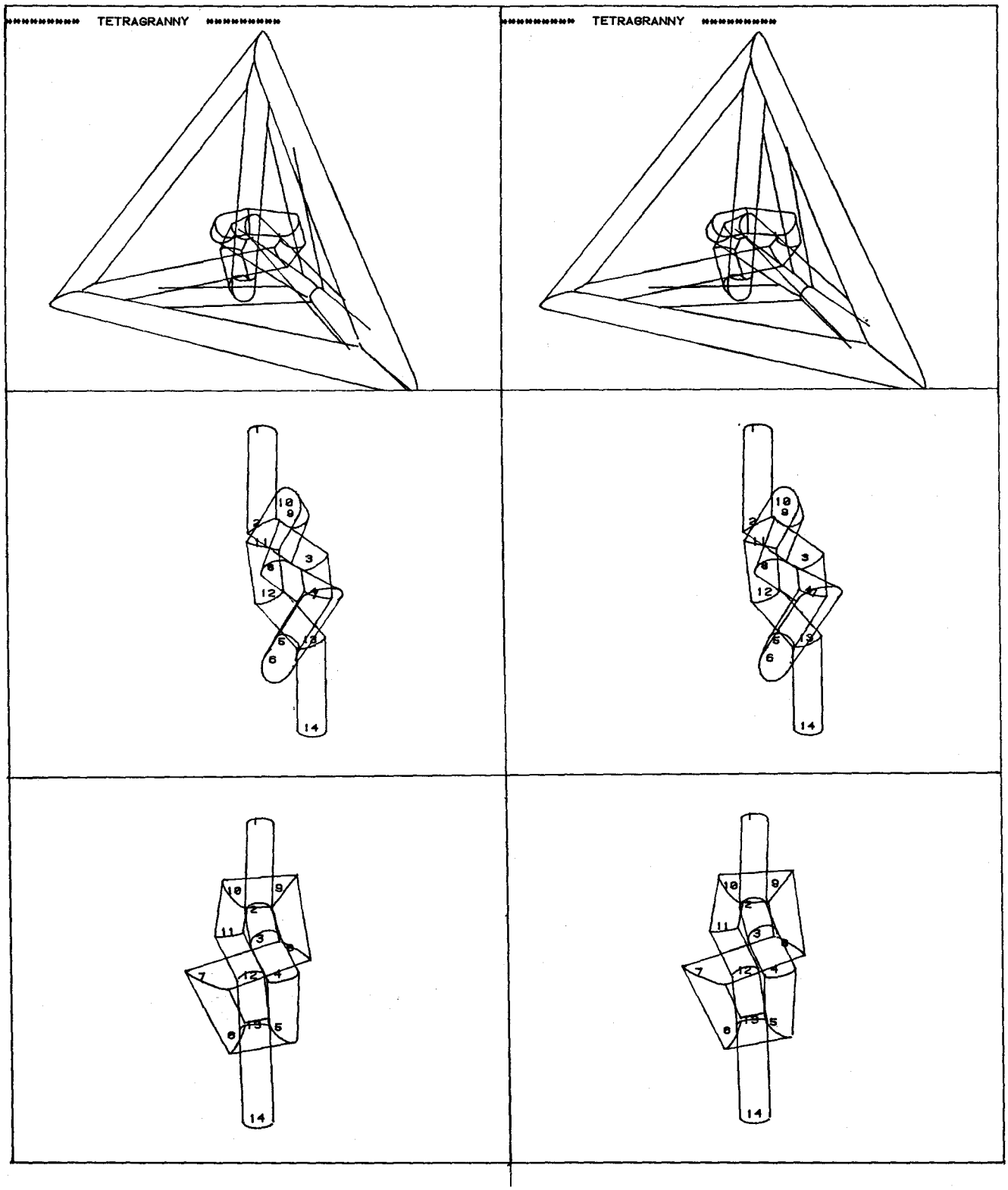






THE CROSS-EYED ART LOVERS
STEREO SCULPTURE GALLERY

THE CROSS-EYED ART LOVERS
STEREO SCULPTURE GALLERY



THE CROSS-EYED ART LOVERS
STEREO SCULPTURE GALLERY

THE CROSS-EYED ART LOVERS
STEREO SCULPTURE GALLERY

At this stage, the physical model, if any, can be put aside and all work done on the data model. If the sculptor is satisfied with this initial description, he skips to step V, ready to construct.

III. EXAMINATION OF THE DATA MODEL

The computer can be used to examine the data model in a variety of ways. Most obvious, it can project drawings of the physical model displayed on the graphic screen or plotter. By specifying a viewpoint and a center of view, a perspective image or a series of rotated views can be created, allowing the sculptor to experience the work as it would actually appear. These drawing routines are base on standard coordinate transformation techniques.

Since the database description of the cylinders includes only the endpoint coordinates and the radius it is a trivial problem to draw a backbone or axial rendering of the design, but much more complex to render the contours of the cylinders and their surface intersections. The algorithm for drawing the intersections is based on the fact that these junctions are elliptical conic sections of the cylinders. The minor axis is equal to the diameter and the major axis is inversely proportional to the angle of intersection. To draw the projection of one of these elliptical intersections, 2-d coordinates are calculated for the points on the perimeter and then these are projected into the 3-d viewing coordinate system and then into the 2-d screen coordinates. The point of tangency of the straight edges of the cylinder with ellipse is calculated as an angular function with respect to the circumference of the cylinder and the major axis of the ellipse. In the case of an intersection of more than two cylinders, the end of each cylinder is characterized by a cutting pattern that is a composite of all the elliptical patterns, typically of differing eccentricity and rotational orientation with respect to the pipe axis. The program, therefore, collects parameters for every intersection at the point in question then calculates and projects only the relevant points along the composite shape. This is discussed further in step V where preparing the cutting templates deals with essentially the same problem.

The current graphic program does not deal with hidden line removal where one pipe crosses another; this was judged to be too complex to include in the 32k or 64k memory and too slow for microcomputer interactive CAD. Hidden lines are removed within each set of intersecting pipes. Excellent depth effects can be obtained by plotting pairs of stereoscopic images, a method that I have found to be a very powerful design aid and which is illustrated for the cross-eyed reader in figure 15.

Computer art is usually thought of in terms of graphic art with the end product created directly on a digital image maker of some type. However, until 1981, I worked without any graphic I/O devices. For a sculptor this is more reasonable than for a graphic artist since the final art object is likely to be physically made, some steps distant from the digital operation. Having only a teletype terminal communicating with a mainframe, I was forced to develop more indirect ways of examining my data model. Analogous to the techniques of crystallographers and biochemists seeking the structure of crystals and molecules, I developed analytic routines that inspect the spatial relations between the parts of a sculpture, using vector algebra to substitute for visual examination.

ALLDIST, based on the algorith, NEAR, is a routine that looks at the distance between each cylinder and all others not connected to it and reports any pair that is less than a specified minimum separation. This way, the Sculptor can identify any pairs that have been placed such that they would collide (separation less than the diameter) and can take action to relieve that interference. Other parameters can also be checked, including the center of gravity, the angles between joined or unjoined cylinders, and the length of a cylinder.

```
THE INPUT POINTS
A1: 4          4.3          13.76        -22.36
A2: 13         11.61        9.245        -17.415

B1: 10         6.45         3.44         -13.76
B2: 11         10.32        12.9         -21.5

PROPOSED SEPARATION: 1.72

CLOSEST PAIR OF POINTS:
LINE-A:        9.2286       10.7159      -19.026
LINE-B:        9.2833       10.3658      -19.4263
THE DISTANCE BETWEEN THEM: 0.5348
 4-13 IS TO CLOSE TO 10-11 BY  1.1852


THE INPUT POINTS
A1: 6          0            1.72         -20.64
A2: 7          11.18        10.32        -24.08

B1: 8          12.47        6.88         -24.08
B2: 11         10.32        12.9         -21.5

PROPOSED SEPARATION: 1.72

CLOSEST PAIR OF POINTS:
LINE-A:        10.8587      10.0729      -23.9811
LINE-B:        11.4321      9.7862       -22.8345
THE DISTANCE BETWEEN THEM: 1.3137
 6-7 IS TO CLOSE TO 8-11 BY  0.4063


THE INPUT POINTS
A1: 6          0            1.72         -20.64
A2: 13         11.61        9.245        -17.415

B1: 10         6.45         3.44         -13.76
B2: 11         10.32        12.9         -21.5

PROPOSED SEPARATION: 1.72

CLOSEST PAIR OF POINTS:
LINE-A:        8.9554       7.5244       -18.1524
LINE-B:        8.3884       8.1783       -17.6368
THE DISTANCE BETWEEN THEM: 1.0074
 6-13 IS TO CLOSE TO 10-11 BY  0.7126
```

fig. 8: ALLDIST, a segment of the output analyzing the distance between all pipes in a network. See notes by Dave Tolle, Programmer, Appendix.

Even after developing the methods for visual examination, I find that such quantitative analysis provides important information about the configuration, giving a clearer sense of the nature of the structure and suggesting the most appropriate ways to modify it to achieve the desired work of art.

IV. MODIFICATION OF THE MODEL

Having inspected the model, the artist has a choice of a variety of means and sequences of altering it to achieve a more satisfactory configuration. PIPEDREAMS includes a library of subroutines that give

fig. 9: LABYRINTH OF DATALIST, 1979, corten, 10' h. The first sculpture designed with Vector modification techniques.

a wide choice of ways to push things around in the mathematical model. Much of my effort has been directed toward achieving maximal density in my sculptures by combining inspection and manipulation to move cylinders together until they are separated by a specified distance. The first sculpture done in this manner was **Labyrinth of Datalist.** My primary concern was to be able to shift pipes out of what I considered impossible situations, i.e., where they were found to be colliding with each other, but to keep them touching. The reiterative routine, **BOUNCE,** which bounces a specified pipe back and forth between two others until it is a specified distance from both was useful in achieving the internal weaving that characterizes this sculpture.

In reshaping this piece from its roughly defined wire model, I came to realize the importance of optimizing flexible control of a specified part of the structure while holding everything else constant. Since 1979, I have continued to add to my library of manipulative operations based on mathematical formulations by Richard A. Scoville. One such routine, **TWIST,** lets the sculptor move one pipe closer orfurther from another by sliding its endpoints along the axes of some other pipes to which it is joined. Another, **SLIP,** brings a pipe to a specified distance from another by holding one end steady and moving the other toward some designated point or along a designated vector. Several of these operations a diagramed in fig. 10.

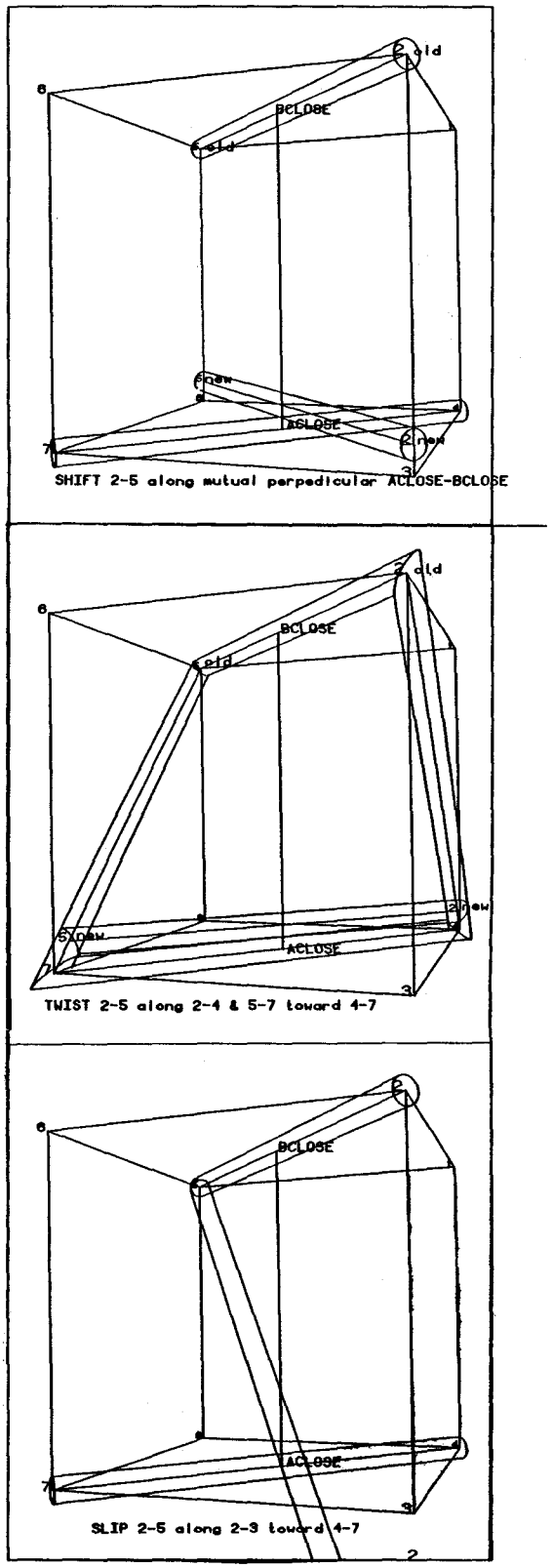


fig 10. several vector manipulations.

By increasing this type of specificity, I have been able to design and construct a series of sculptures in the form of tight knots . Also helpful in this pursuit is**TRUNCATE,** which truncates an acute intersection of two pipe by adding a third pipe in place of the intersection and "twisting" it up against another (see fig. 11).

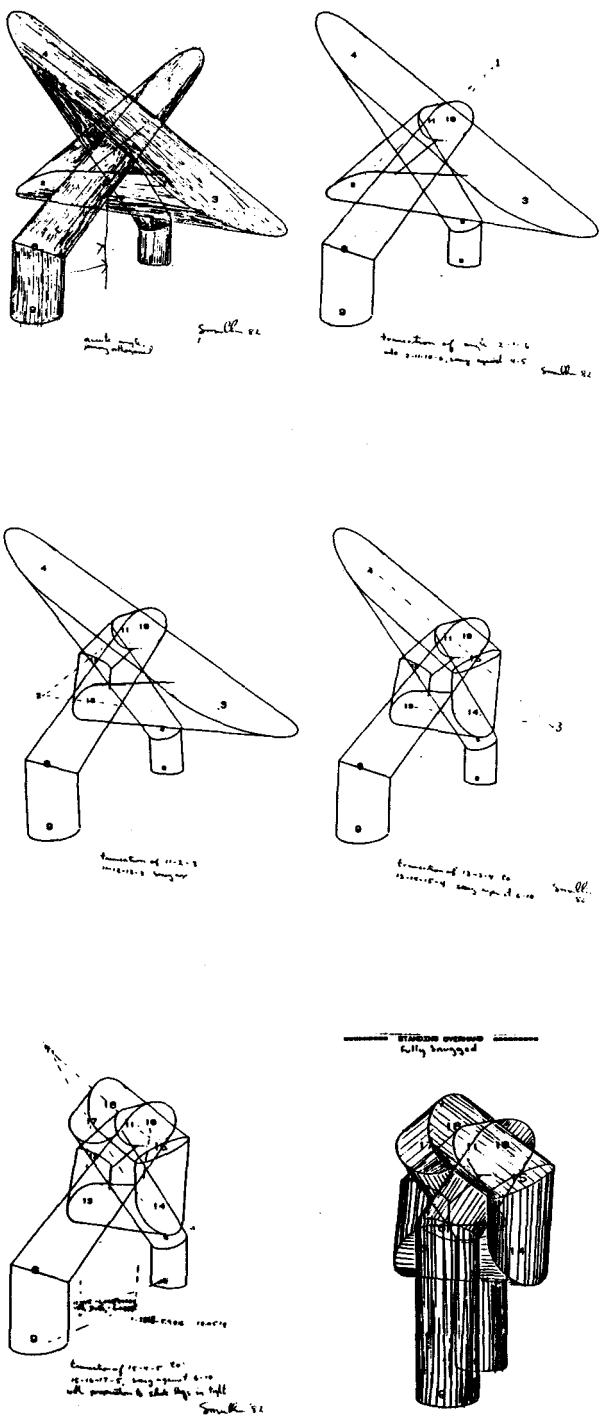


fig. 11: UTICA OVERHAND, 1980, and illustration of use of TRUNCATE to tighten the knot.

After having spent much time developing means for correcting impossible situations (where pipes would be colliding), I came to realize that these situations were, in fact, not impossible, just difficult, when I worked out a method to make the necessary interpenetration patterns descriptively. Richard A. Scoville then provided a function, eq. (4), to describe the situation quantitatively and I wrote it into a subroutine. This has allowed superdense structures such as in the series, **FIVE STOPS ON THE WAY TO A BLACK HOLE,**1983.

$$y = r * \frac{\sin(\Theta)*\cos(X) \pm \mathrm{sqr}(1-(\cos(\Theta)-D)}{\sin(X)}$$

eq.(4)

where

y = distance along the axis of pipe A from the point closest to pipe B, projected onto the surface;
r = the radius;
Θ = rotation around the axis from AxB;
X = the angle between A and B;

$$D = \frac{\text{distance between axes}}{\text{radius}}.$$

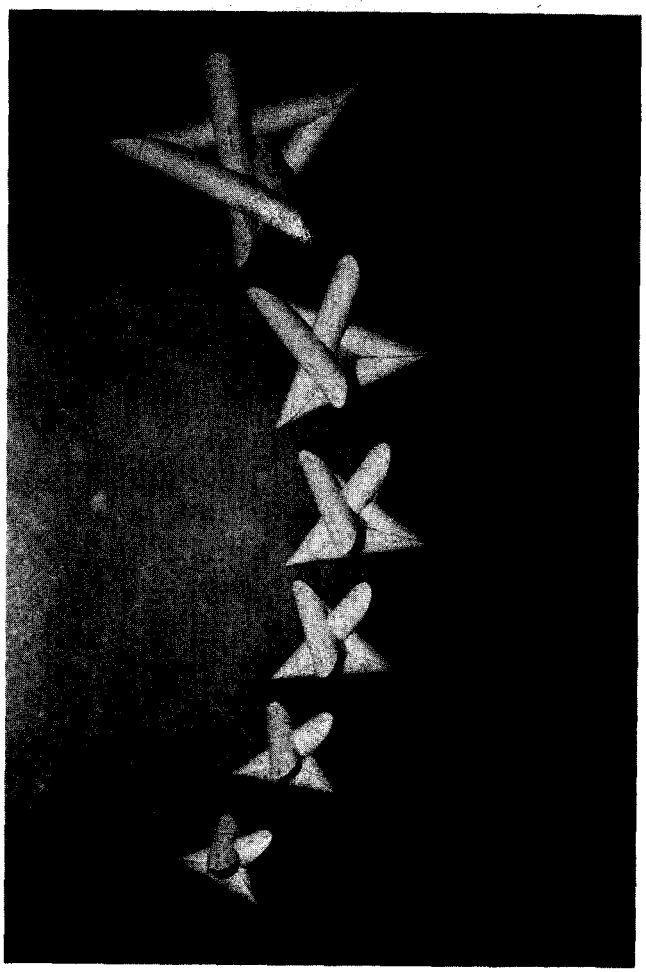


fig. 12: FIVE STOPS ON THE WAY TO A BLACK HOLE, 1983. Superdense interpenetrating networks.

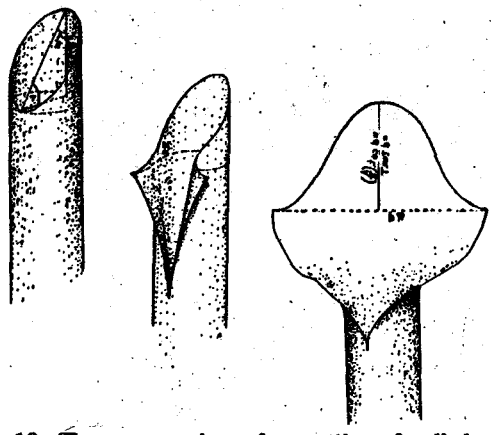


fig 13: The unwrapping of a section of cylinder to show the relationship between the elliptical cut plane and the sinusoidal template.

V. CALCULATION OF THE CONSTRUCTION DATA

Relatively few parameters need to be calculated to construct a tubular network. However, the number of instances for which the calculations are done in a complex structure are many and then that information must be used with great precision.

The application of these functions can be outlined as follows:

At each point
calculate
1. its length
2. its angle of intersection with every other pipe joining there
3. the rotational angle of that plane of intersection with respect to some reference plane at that point
4. the rotational angle of that reference plane with respect to the reference plane at the opposite end of the pipe.

This data is used to create the appropriate cuts in the pipes to allow them to join as contiuous surfaces. Actually, the angle and the rotational functions are only intermediate parameters that are used to calculate the sinusoidal function that describes the intersection of any pair of intersecting pipes. In fig. (12) one sees demonstration of the relationship between the elliptically shaped planar section of the pipe and the sinusoidal pattern that is wrapped around the pipe. Trigonometry shows that the amplitude of this curve is a function of the radius and the angle of intersection, or

$$amp = R/\tan(A/2). \qquad \text{eq. (5)}$$

Then, the complete expression for each pipe joining in a complex cluster of several pipes is:

$$y = R / \tan(A/2) * \sin(x+x_0+x_1) \qquad \text{eq. (6)}$$

where

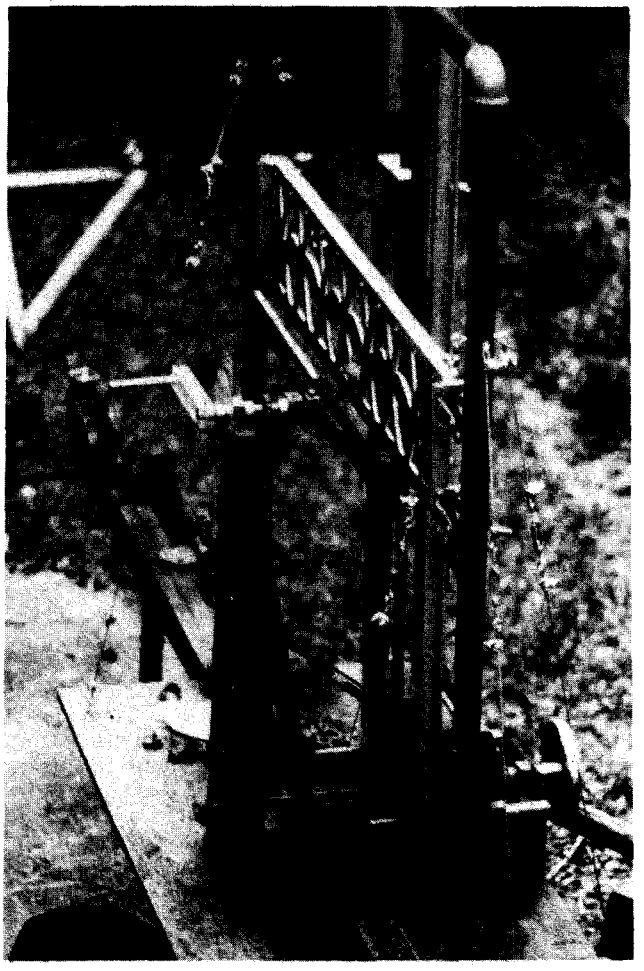
r = radius;
A = angle of intersection;
x= distance (or rotation) along circumferential axis;
x_0= rotation of the plane of intersection from a reference pair at the same point;
x_1= rotation of the plane of intersection from a reference plane at the other end of the pipe.

As in drawing the perspectives, in the case of an intersection of more than two pipes,the pattern consists of the minimum values of the superimposed sinusoids, i.e. what is left of the pipe after all cuts have been made. One can see that x_0 must take into account the rotational relationship with other pairs of pipes joining at the same point as well as with the plane of some pair at the opposite end of the pipe being prepared.

VI. PREPARATION OF THE MITER PATTERNS

With this data, the component cylinders for any structure can be prepared. Until this past year, the data was used to manually or mechanically draft templates and to cut the pipes into the proper lengths and mitering shapes. As demonstrated by my sculpture from 1974-1981, quite complex and large structures can be made done in this manner. Most of my sculptures from 1976 through 1981 were done with a mechanical analog pattern maker (fig. 14) which draws the pattern directly onto a pipe or onto a wrap-around template. Patterns for large diameter pipes (over 3.5 in. diameter) were prepared by variations on standard pipefitting projection techniques. However, in the current version of PIPEDREAMS, the construction data program has been written to drive a Tektronix 4662 flatbed digital plotter to make a single, complete pattern for each length of pipe. That pattern is made in the following sequence:

1. All intersections at one end of the pipe are calculated and the composite pattern plotted along a vertical circumference. Numerical and graphic alignment aids are plotted.
2. A global search is done for interference between this pipe and any others not connected to it, collecting parameters for all collisions found.
3. An interpentetration pattern is plotted for each collision at the proper distance and rotation along the pipe.
4. Any collisions beyond the limit of the length of the bed are saved for another plot for which the user is asked to mount a new paper and the plotting carries on.
5. If the total pattern length is greater than the plotter bed, one has the option of making a truncated pattern, or mounting a new paper to complete the full length.
6. For the opposite end of the pipe, step one is repeated, properly referenced to the first end in terms of distance and rotation.



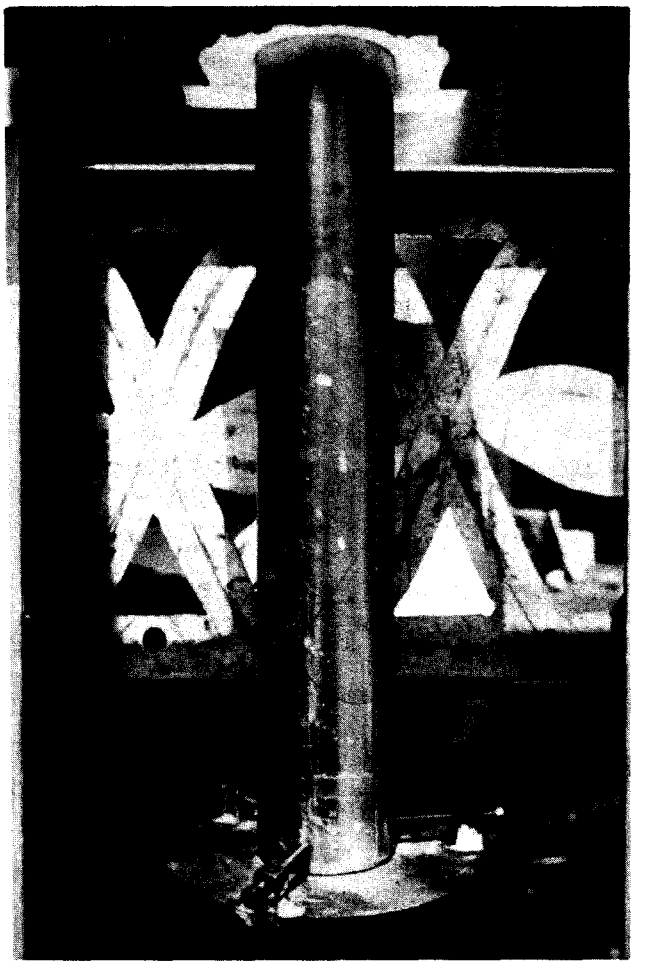


fig. 14a: Mechanical Pipe Mitering Device, 1976, draws patterns directly on pipe of diameter <4". Scotch yoke cam adjusts for sine amplitude, rotor for dihedral rotation. fig. 14b: A section of 2"diameter pipe being scribed .

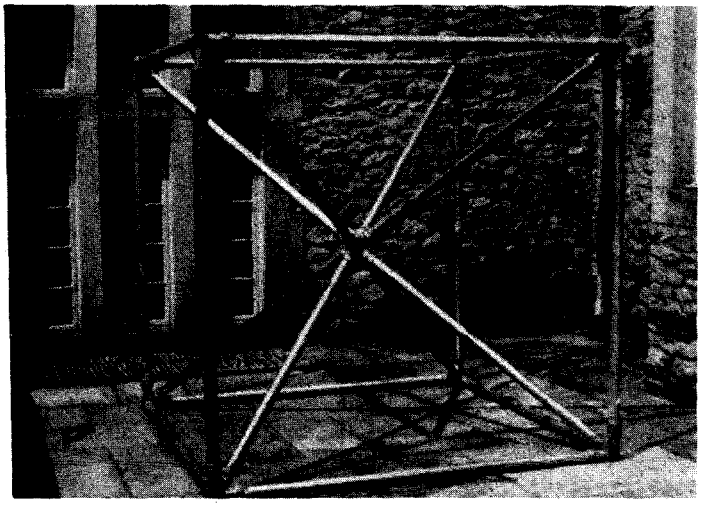


fig. 14 c: Vertex-centered Cube, 1977, electrical metallic tubing and compression couplers. An example of sculptures constructed with the device. Fig.14b shows one of the center pieces.

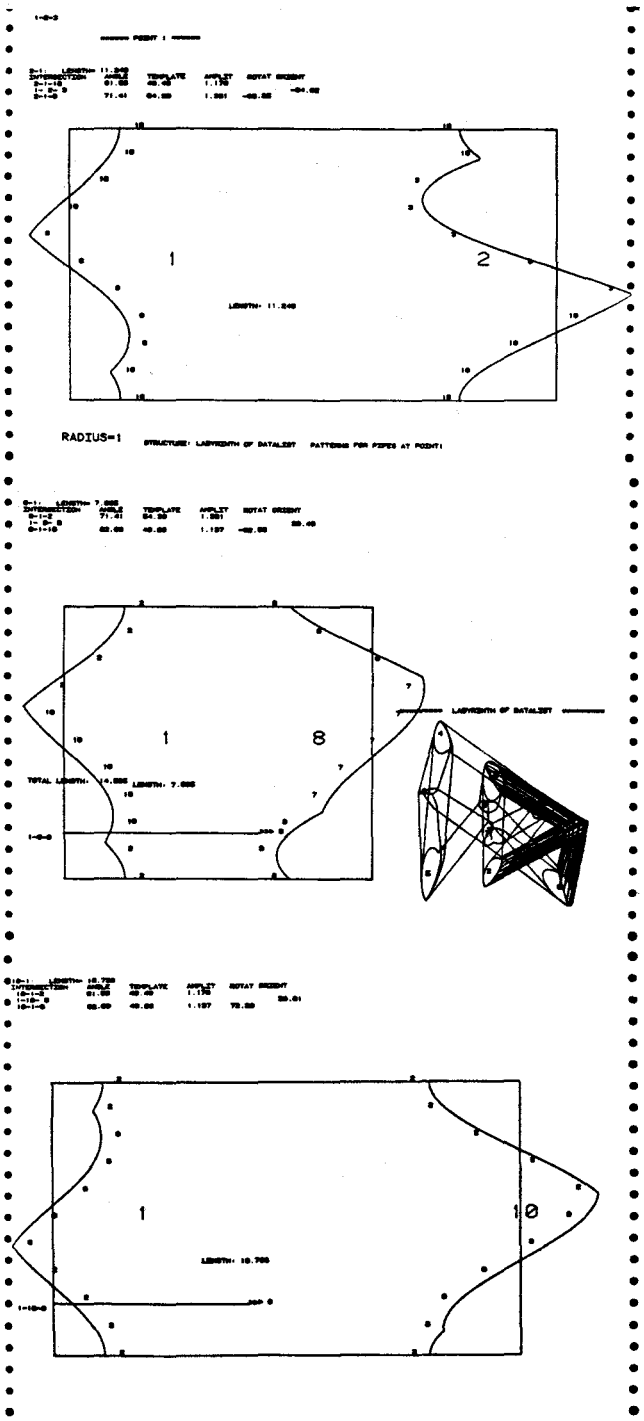
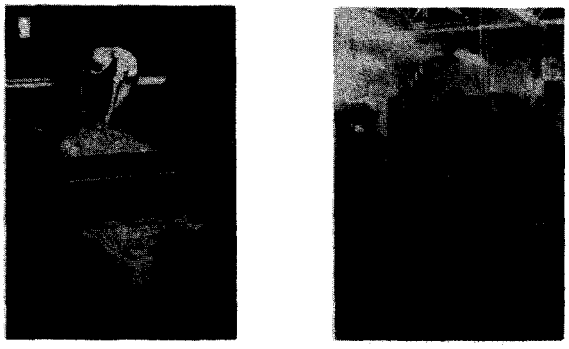
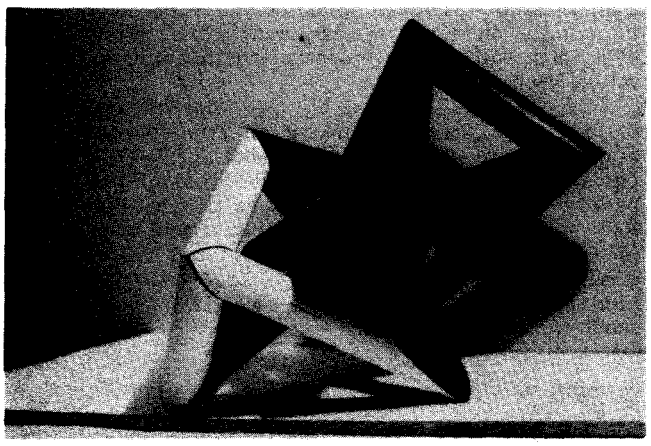


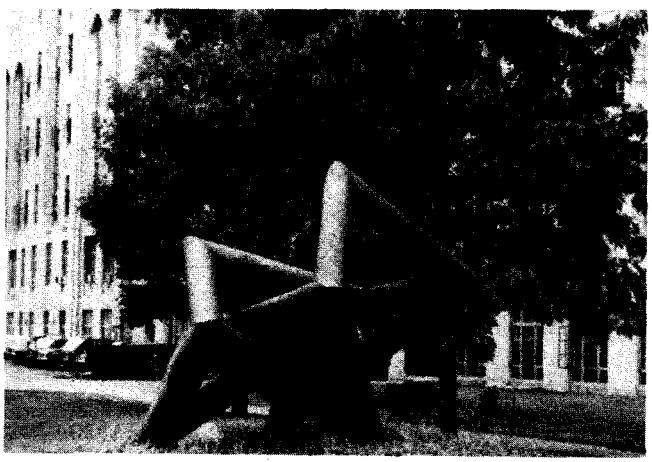
fig. 15: Pattern Making. a) Complete templates made by PIPEDREAMS on the 4662 plotter for the 3 pipes joining at point 1 of LABYRINTH OF DATALIST. Parameters are printed and patterns labled. Note each pattern end consists of two sine curves of different amplitude properly rotated from each other. Also, the two ends are rotated.



b) Steel model of LABYRINTH with patterns wrapped on tubes coming from point 1 (this model was actually created with patterns made on the mechanical analog pipe miter, see fig. 14).



c) Patterns for the full-scale Labyrinth, descriptively made by hand with data from CONPIPES, are traced onto corten plates, then band-sawed and rolled into pre-mitered cylinders.



d) The finished sculpture at Boston University.

VII. CUTTING AND JOINING THE CYLINDERS

Execution of this and the previous step with sufficient precision to make the quantitative analytic process meaningful has been as challenging as the programming and has resulted in as many innovative method. The sculptor doing CAD-CAM has major problems not encountered by the graphic artist. CAM strategies such as mine leave the artist with 2-d computer generated items which must be turned into 3-d objects and arranged in 3-d space. The final peripheral devices are manual, and the final object is hand-made. During the past decade, I have experimented with several different techniques for fabricating my sculptures. The appropriate method is a consideration of what devices and materials are available. Note (fig. 15) that the cylinders for LABYRINTH OF DATALIST were made from plates that were rolled into cylinders after the patterns were cut on the ends, whereas TETRAGRANNY was made from stock aluminum pipe (fig. 16).

Since completion of the current system which plots the complete pattern, many of the previously encountered problems have vanished as the integral pattern manifests all the critical spatial relationships when glued onto a pipe of the proper radius. It is then possible to cut and glue the the segments together in a linear or branching fashion with the certainty that everything will project properly and that networks will return upon themselves.

As one is limited by the dimensions of the plotter, I am currently working on two approaches to achieving comparable direct methods for large diameter pipes. I am developing a microprocessor-controlled scribe to draw directly on pipes of any dimension. More likely to be in use before this article is published is a 10:1 pantograph. These devices will make it possible to work on a very large scale with the same kind of certainty that now characterizes my work with small cardboard tubes.

CONCLUSION

The current version of PIPEDREAMS is written to run on the TEKTRONIX 4052. While it is possible to operate within a 32K RAM, I have increased the memory to 64K which allows handling of more complex structures, and having more of the program options in memory at one time. Memory allocation is the major problem in such a micro workstation. PIPEDREAMS is written in four major programs: GENERAL(14K), with all the common subroutines; MAIN(12K), which handles the inputting of descriptions, simple modifications and tape I/O; DRAW(24k), which does all the viewing drawing; and MOD(23K), which does the complex vector modifications. Since I can not have all these in memory simultaneously, GENERAL allows me to have any two plus GENERAL at once. Typically, this means having DRAW available while doing the original input or while doing MODifications. The vector drawing of the Tektronix CRT display is a perfect match for the conceptual model I have chosen to use. Also, its fine resolution permits undistorted ellipse drawing. In the near future I will write a version to run on a raster display in order to utilize surface modeling and color.

In the course of the past nine years, this technique has undergone much modification and elaboration. As I have become more aware of what I want to do with the system, I have been able to make it perform more efficiently in those manners. Aside from the considerable assistance I have received from several people, this project has been mostly a one person operation. I have been artist, programmer, machine designer and fabricator, sequentially and simultaneously. While this singlehandedness has often been a handicap, I have found that my intimate knowledge of all phases of the process has been very important in relating the problems and potentials of each phase of the operation to the final goal of making sculptures that otherwise would be impossible. I am finding satisfaction in seeing my pipedreams become reality.

fig. 16a: The pieces for TETRAGRANNY are cut from 9" diam. aluminum tube with a Stihl cut-off saw. The planar blade facilitates a straight cut.

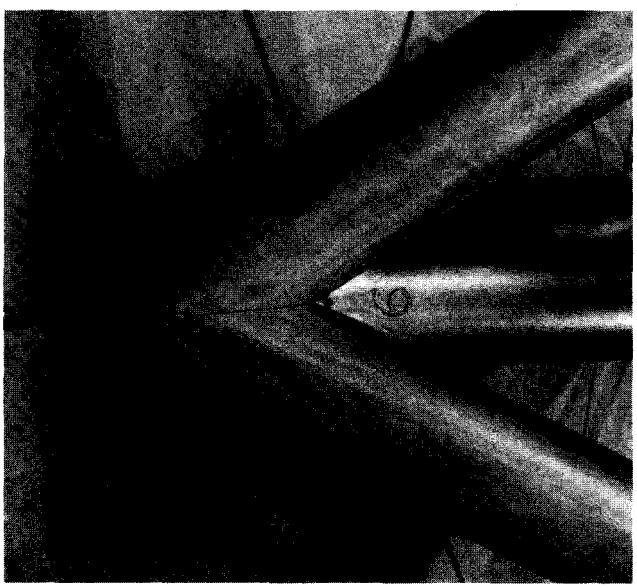


fig. 16b: A detail of TETRAGRANNY showing the accuracy of the mitering, allowing complete on-site assembly in three hours using iron wire to tie it together.

ACKNOWLEDGEMENT

Throughout this extended endeavor I have had the assistance of knowledgeable, creative and generous people. From the outset, whenever my math has failed me, I have been able to turn to Richard A. Scoville of the Department of Mathematics, Duke University to derive a function. Eric Bam, now with IBM, while an undergraduate at Duke wrote the first programmed version of my system. I learned to program by reading his program and much of his style has remained in the basic structure of PIPEDREAMS. Dave Tolle, while a graduate student at University of North Carolina wrote the first graphic I/O version of my system in 1976. He also wrote the routine NEAR that is the heart of all my quantitative inspection operations. Gary Grady and Shiang Ti Tuan of the Duke university Computation Center were very important in carrying me to more complex levels of PL/I. Joseph Francis, a Duke Undergraduate, working as my assistant, helped me rewrite the system into BASIC and contributed many useful modifications, making it more user-friendly.

Important financial support for this project has come from Duke University, The Mary Duke Biddle Foundation, The Alcoa Foundation and Boston University.

Finally, I acknowledge the special support I have received from my wife, Ruth Ann, and my children, Sylvia and Rebecca, who have helped me keep some human perspective as my work has got more amd more tied to a machine that is most unforgiving.

BIBLIOGRAPHY

1. Chasen, Sylvan Geometric Principles and Procedures for Computer Graphics Applications, Prentice-Hall, 1978.
2. Del Zoppo, Annette, The Vegreville Pysanka, pub. Ron D. Resch,Salt Lake City, 1976.
3. Leavitt, Ruth Hartman "Computer-aided Sculpture," Proceedings of NCGA'83,National Computer Graphics Association, Chicago, 1983.
4. Newman, W.M. and Sproull, R.F. Principles of Interactive Computer Graphics. McGraw-Hill Book Co., N.Y., 1973.
5. Resch, Ron D. "The Topological Design of Sculptural and Architectural Systems," Proceedings of the National Computer Conference,1973, pp.643-650.
6. _____ "Computer Works", Proceedings of the 7th International Sculpture Conference, University of Kansas, 1972, pp. 121-129.
7. Smullin, Frank M. "Description of 3-D Structures with Analytic Geometry,".SCLPTR,Southern Association of Sculptors, spring-summer 1977,pp.2-6.
8. _____"The Computer as a Sculptor's Tool," Sculptors International, International Sculpture Center, vol.1, no.3.

9.Thomas, George B., Jr. Calculus and Analytic Geometry. Addison-Wesley, 1960.

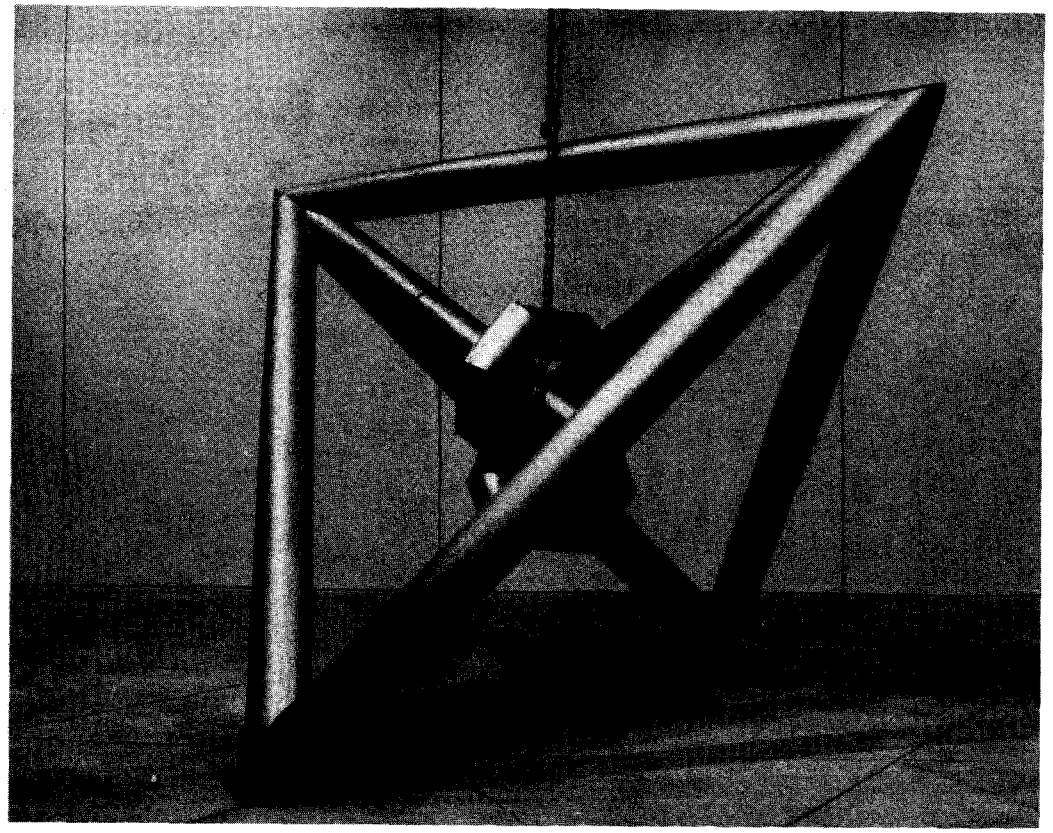
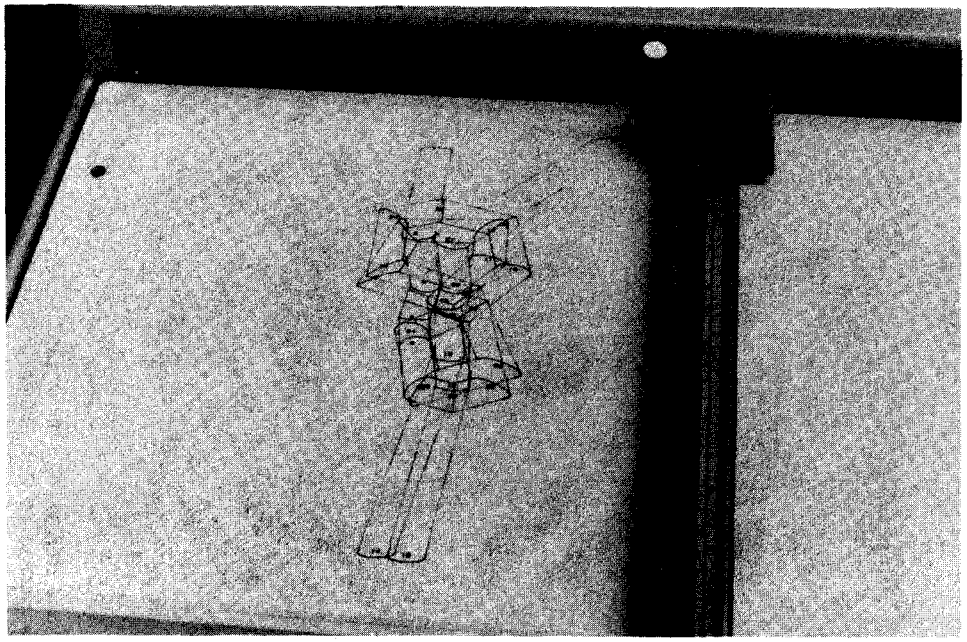
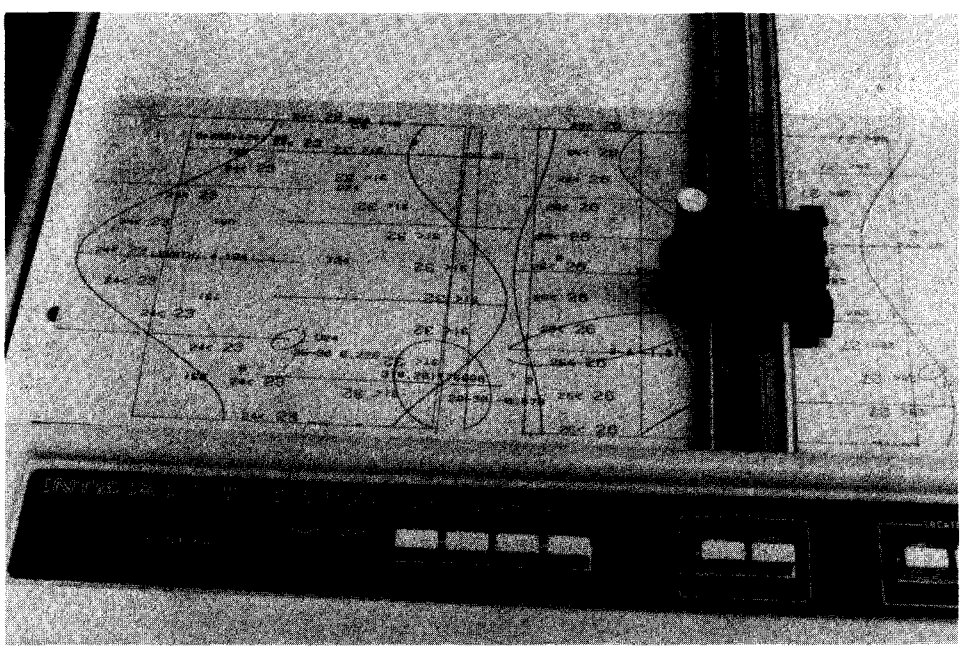


fig. 17: TETRAGRANNY, 1981, aluminum and iron wire, 140" x 140" x 99". Installation: Southeastern Center for Contemporary Art, January 1981.





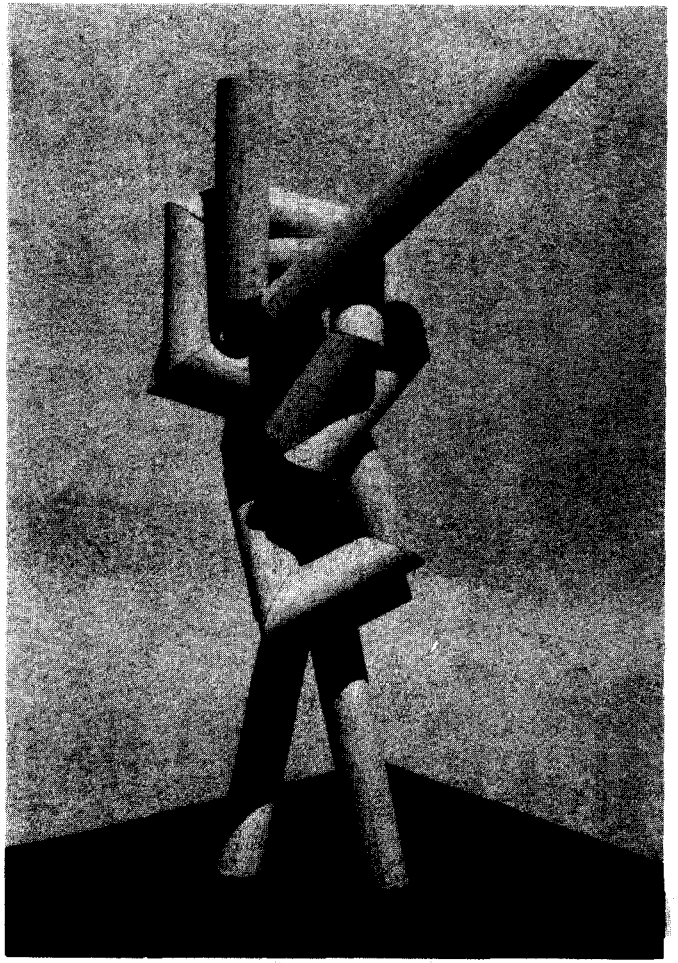


fig. 18: PIPEDREAM GRAPHICS. Using PLOT 50 on the Tektronix 4052 computer and 4662 plotter (upper left) a perspective view of a developing design, optionally, points are labled and intersections opaque; (lower left) complete patterns for sculpture are plotted with intersection and interpenetrating curves; (right) the finished sculpture, FIGURE EIGHT FIGURE, 1983, cardboard tubes with paper patterns, 14"x8"x28", note the free ends have been cut to patterns making them horizontally planar.

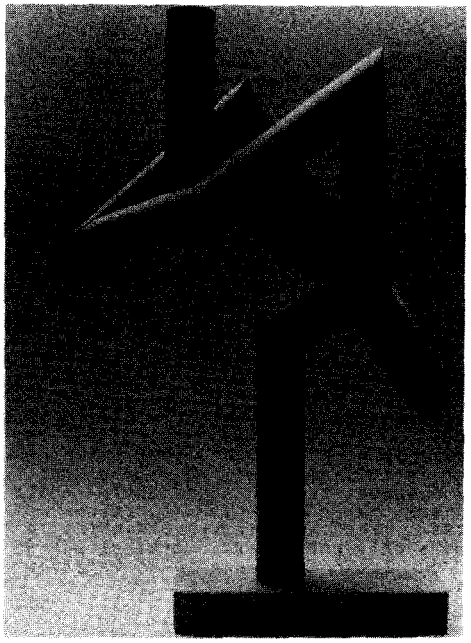


fig. 19a: FIGURE EIGHT-1, 1983, cardboard tubes with computer plotted paper templates,15"x8"x20".

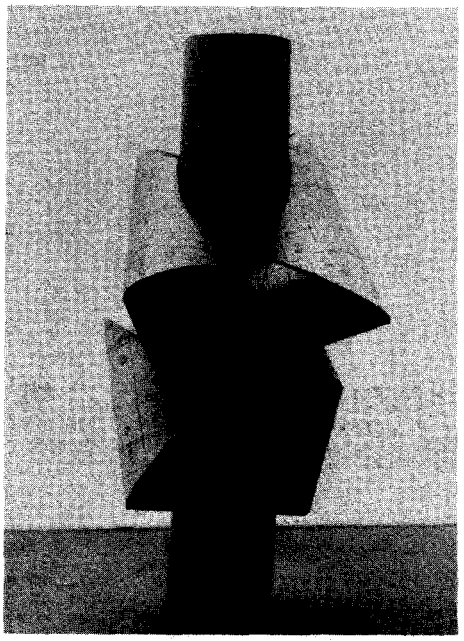


fig. 19b: FIGURE EIGHT-5, 1983, cardboard tubes with computer plotted paper templates, 5"x31/2"x9".

fig. 19c: UTICA OVERHAND, 1980, steel, 142" h.. Fabricated by the artist (shown) at Sculpture Space, Utica, NY.

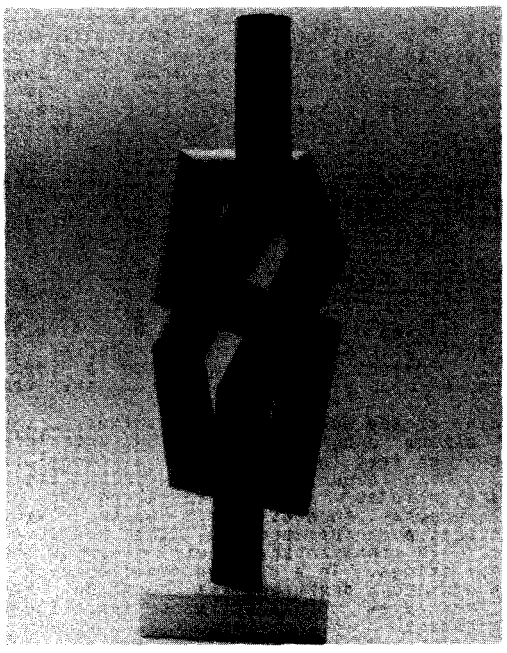


fig. 19d FIGURE EIGHT-3, 1983, cardboard tubes with computer plotted paper templates,61/2"x7"x19".

USING AN ARTISTS' WORKSTATION

by Walter Wright

Digital Image Corporation
Grand Rapids, Michigan

ABSTRACT

In this paper, I will discuss various applications of the artists' workstation developed by Digital Image Corporation. Although designed for a micro-computer this system equals, if not betters, more expensive main-frame systems.

INTRODUCTION

Digital Image Corporation makes computer generated 35mm slides for audio visual & television producers and graphic artists. We aspire to be better and less expensive than the competition. Therefore we use relatively inexpensive hardware and we write our own software. I will list briefly the hardware and then describe in more detail our graphics software.

Our hardware includes a Cromemco Z2D microprocessor with video terminal, dual 5 1/4" floppy disks, 64K bytes of random access memory, the SDI graphics controller, and two 48K image buffers or pages. As inputs we use a Houston Instruments bitpad and a Via Video digitizer. This latter device consists of a b&w video camera and an interface to digitize and store the camera image in either of the two image buffers. As outputs we have an Aydin Controls RGB monitor and a Matrix Instruments camera system. The camera takes the separated red, green and blue video images from the computer, displays them in turn on its own high resolution b&w monitor and exposes each image through the appropriate color filter onto a single frame of 35mm film.

Worth approximately $10,000 (not including the cost of the matrix camera), this system isn't for the average hobbiest, but it is within the grasp of enterprising individuals like ourselves.

Our software is designed to make commercial slides. This means word copy slides including: slides for speaker support, to augment training manuals, title slides, tables of numbers and so on. Also popular are charts, graphs, and various special effects slides. Posterization (digitizing a video image and adding color electronically) is a hot item.

Beginning at the beginning, our software includes image making programs such as FONT which produces word copy or text in a variety of typefaces or fonts. More about designing fonts later. Our software includes image manipulating or 'cut & paste' programs such as MENU. Bitpad driven, this is our most powerful program, the heart of the system. In this article I will describe MENU and its applictions. In a related article David Cook, MENU's designer, describes its operation. Finally, SHOOT retrieves completed images and outputs them to the camera.

FONT - CREATING A TYPEFACE

Ours is a medium resolution system. Cromemco's SDI graphics operates in two modes: 378 x 241 pixels (horizontal by vertical) in 16 of a possible 4096 colors and 755 x 482 pixels in 2 colors only. Attempting to render smooth curves at these resolutions can produce "the jaggies". To avoid this distracting "special effect" in designing a typeface or font, we avoid curves. All our fonts are designed with straight lines. Further, we use only horizontal, vertical, 45 degree and, rarely, 22 1/2 degree lines. Our early fonts such as Digital are rather simple renditions of common typefaces. Then we discovered Machine, a typeface which uses only horizontal, vertical and 45 degree lines. Our later designs such as Teletype are more refined.

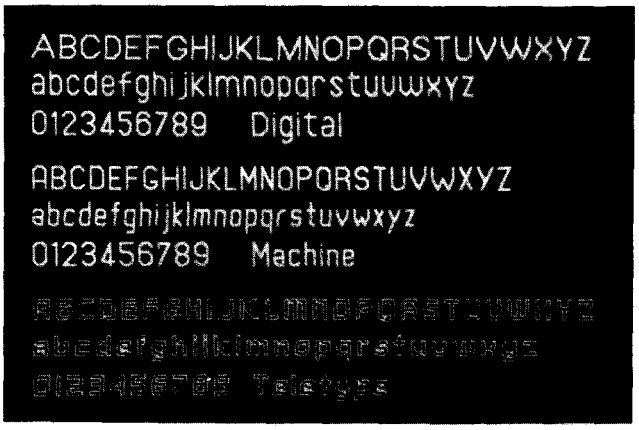


FIGURE 1 - FONTS

Fonts are designed on graph paper. Normally each letter occupies a 10 x 12 grid, however, the width varies with the letter. Fonts are coded using simple commands such as:

```
M(ove), x, y
L(ine), x, y
V(ector), x, y ... (, x, y)
B(ox), w, h
X (end definition)
```

Where: x = horizontal offset
y = vertical offset
w = width
h = height

The result is a table defining each letter in the font. A command (X) to delimit each letter and to link the data to the appropriate alphanumeric A-z, 0-9 etc, is included in the data file. The filename is the font name, for example, DIGITAL.FNT, MACHINE.FNT, TELETYPE.FNT and so on.

We have two types of fonts. First, those that are single lines only such as Digital and Machine. Simple line fonts are suitable for the main body of

word copy slides. They are legible even in small sizes. They include both upper and lower case. Second, the outline fonts such as Teletype are suitable for titles. Some of these fonts don't have lower case letters. As titles they can be outlines or they can be filled. At present we use a separate FILL program; soon we will include a fill command in the font data file. We have experimented with numerous fill algorithms, a subject for another future article.

FONT, the program, asks the user to select a font. The user types in the font name using the video terminal keyboard. The appropriate data file is loaded into memory. From here the user has various options, FONT includes these commands:

```
F(ont)      - Select a new font
E(rase)     - Clear the current page (page 0)
M(ap)       - Set default color map (white on blue)
X n         - Set horizontal scale to n
Y n         - Set vertical scale to n
S n         - Set spacing between letters to n
V n         - Set spacing between lines to n
P n         - Set pen type and size
C n         - Set pen color (0-blue, 1-white)
B(itpad)    - Get X and Y location from bitpad
K(eyboard)- Get X and Y from the keyboard
'text       - Enter single line mode
*           - Redraw single line
U(p)        - Back up one line
N(ext)      - Skip to next line
?           - Display commands and their values
H(elp)      - Get help
"           - Enter/Exit multiple line mode
              ^ - Define a tab
              @ - Go to a tab
              ~ - Begin/end underline
              / - Enter commands
```

I'm not going to explain the complete command structure. Multiple line mode is extremely useful for creating tables and charts. I am going to discuss a simple example.

Presume the user is creating a title slide, he/she has loaded Teletype and is ready to enter the title. Type-

```
E M X 40 Y 40 B ' Shazam! (return)
```

FONT erases the current page, sets the foreground color white and the background blue and sets the horizontal and vertical scales to 4 (40 / 10). A number of defaults are set, for example, the pen defaults to draw lines one pixel wide. FONT waits for the user to enter a starting point on the bitpad, a cursor on the RGB monitor screen indicates the position of the bitpad stylus, pressing down on the stylus enters its location. FONT draws the line of text. We will return to this slide later.

FIGURE 2 - Shazam!

MENU - ONE FROM COLUMN A, ONE FROM COLUMN B

At the creative center of our system is MENU. The user communicates with MENU using the bitpad and, occassionally the keyboard; MENU communicates with the user on the video terminal screen.

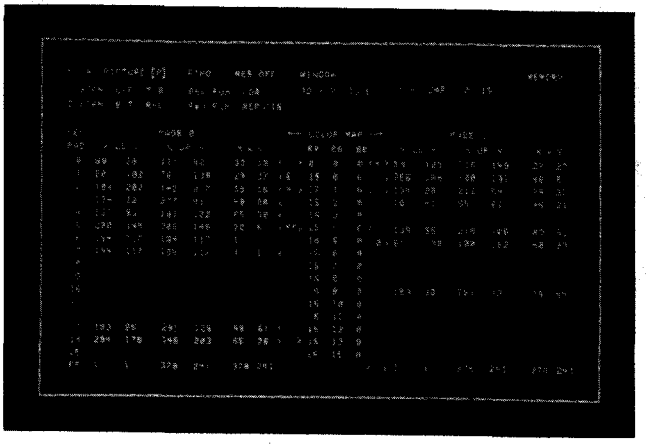


FIGURE 3 - VIDEO TERMINAL SCREEN

As you can see, the screen contains a wealth of data. Miscellaneous control parameters such as current file name, scanner settings, jump increments and so forth appear on the top three lines. Below and moving left to right we find the KEY PAD vector, a column of numbers 0-15,FF. Next are several columns of figures under the heading PAGE 0. The user can define as many as sixteen boxes per page; under subheadings X LL Y and X UR Y we find box coordinate information and under X x Y the length and width for each user defined box. Now the tricky part. To the immediate right of the box cordinates are three columns of box control information; the box select vector (a "<" indicates a selected box), the force vector (a number 0-15,FF) and the alternate page vector (a ">" points a box directly to the alternate page). Box control is enormously powerful , a complete description would bog us down for pages, so we'll deal with these vectors later in some examples of practical and not so practical applications. Moving to center screen we find the heading COLOR MAP and directly under it three subheadings: RR, GG, and BB (red, green, and blue). A maximum of sixteen colors (0-15) are possible in low resolution. These sixteen colors are chosen from a palette of 4096 colors formed by combining differing intensities of red, green, and blue. Each primary color can vary in intensity from black (level 0) to full color (level 15) as indicated in the color map table. Next, on the far right, we find PAGE 1 box control and box coordinate information, similar to PAGE 0.

A final note concerning the screen; headings appear dim while filename, scanner settings and jump increments appear bright. Numbers in the key pad vector and RGB intensity levels in the color map are either dim (OFF) or bright (ON). The box select indicators "<", force values and alternate page indicators ">" are either not visible (OFF) or visible (ON).

Now we have an exhausting list of MENU commands. Those who get bored or confused easily can skip to APPLICATIONS. MENU commands appear as keys on the bitpad. Th user enters commands by hitting the appropriate key while the bitpad is IN POINT MODE. The user enters coordinate information while the bitpad is IN STREAM MODE, see [BOX GRAB].

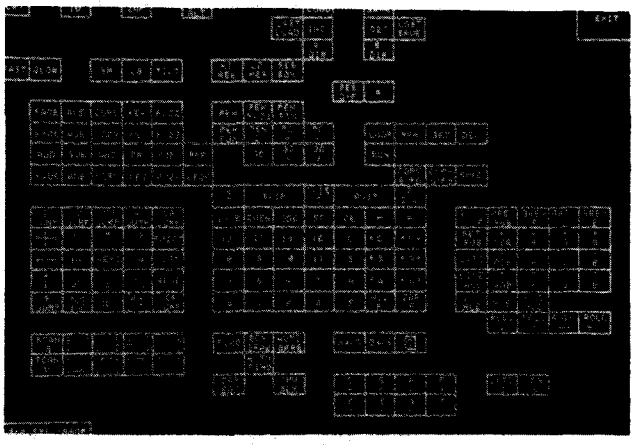


FIGURE 4 - THE BITPAD

BASIC PROGRAM COMMANDS

[CRT] Toggles the display on/off.

[ZAP] Clears the current page and must be confirmed by hitting twice.

[ZAP ALT] Clears the alternate page as above.

[HI RES] Sets high resolution: 755 x 482 in 2 colors.

[LO RES] Sets low resolution: 378 x 241 in 16 colors.

[SAVE] The user enters a filename on the keyboard; MENU compresses and saves the current page on disk. The file appears in the disk directory as filename.PIX.

[LOAD] Loads to the current page from disk as above. A disk load can be made in one of six modes - add, subtract, and, or, xor, replace. More later.

[INC] Increments the last character of the current filename 0-9, A-Z. If the current filename is "T0", hitting [INC] will change the filename to "T1".

[DEC] Decrements the last character of the current filename as above.

[LAST SAVE]

Saves the image from the current page to disk using the last filename entered on the keyboard, or the last filename as modified by the [INC] [DEC] commands.

[LAST LOAD]

Loads an image from disk to the current page using the last filename as above.

[SM] Grabs a small square image from the b&w video camera. The image can have as many as 16 levels of grey. It is loaded in low resolution directly to the current page.

[LG] Grabs a full frame at half the normal low resolution and loads it to the current page.

[TILT] Grabs a small image vertically and a full frame horizontally, as if the full frame image were tilted back 45 degrees. It is loaded to the current page.

[EXIT] Exits MENU and returns to the system monitor.

KEY PAD AND VECTOR CONTROL COMMANDS

The key pad includes color vector, box select vector and alternate page vector keys, as well as several frequently used control keys such as the page key.

[0] - [15]

Are used with related keys to set box numbers, color codes, colors, jump, 3D and loop parameters.

[1-15] Automatically sets number 1 through 15 in the keypad vector.

[EVEN] Sets all the even numbers in the keypad vector.

[ODD] Sets all the odd numbers in the keypad vector.

[FF] Sets "full frame" in the keypad vector.

[CE] Clears the keypad vector.

[-] Is the "through" key, for example,

[-] [0] [7]

will set numbers 1,2,3,4,5,6,7.

[C] [S] [A]

These keys transfer the contents of keypad vector (the leftmost column of numbers on the video screen) to the color vertor, box select vector and alternate page vector respectively.

[<-C] [<-S] [<-A]

These keys reverse the function of the above keys transferring the contents of the color vector, box select vector and the alternate page vector to the keypad vector.

[<-C->] [<-S->] [<-A->]

These keys swap the contents at the keypad vector and the color vector, box select vector and alternate page vector respectively.

[0 C] Sets only color 0 in the color vector.

[1-15 C] Sets only color 0 through 15 in the color vector.

[F] Allows the user to point a selected box to another box on the same page, for example,

[CE] [0] [S] [F] [1]

will point a box 0 to box 1.

[ALL F] Points all selected boxes to a single designated box.

[ZAP F] Cancels all force values on the current page.

[P] Toggles between page zero and page one.

[PSA] Toggles to the alternate page taking along the select box and alternate page vectors. That is, the contents of the select box and alternate page vectors of the old page are transfered to the select box and alternate page vectors of the new page.

[COPY S->D]

Copies the image from the current page (S) to the alternate page (D). This is a high speed copy which ignores color vector and function mode settings.

[COPY D->S]

Reverses the function of above key and copies the image from the alternate page (D) to the current page (S).

[S<>D] Swaps the images on pages zero and one.

[S<-D] Transfers the coordinate information from a destination box to a source box, for example,

[CE] [0] [S] [F] [FF] [S<-D]

sets box 0 to full frame.

[D<-S] Reverses the function of the above key, for example,

[CE] [0] [S] [A]
[BOX GRAB] (bitpad coords) [D<-S]

sets box 0 on the alternate page equal to box 0 on the current page.

[+C] [+S] [+A] [+F] [-C] [-S] [-A] [-F]

Allow the user to move enabled colors, selected boxes, alternate page pointers and force values up (+) & down (-) the control vectors, for example,

[CE] [0] [S] [CE] [EVEN] [C]
[+S] [+C]

selects box 0 and enables colors 0,2,4,6,8,10,12,14. [+s] selects box 1 and [+C] enables colors 1,3,5,7,9, 11,13,15.

BOX COMMANDS

[BOX GRAB]
MENU takes selected boxes, in turn, and sets their lower left & upper right corners to coordinates entered by the user from the bitpad. MENU calculates the length & height of the box and updates the coordinate information on the video terminal. In order to enter coordinates, the user switches the bitpad from point to stream mode. On the display screen a cursor appears. This cursor moves with the stylus. Coordinates are entered by pressing the stylus on the bitpad. Remember to return to point mode after all selected boxes are set.

[AUTO GRAB]

As above, MENU automatically selects boxes 0 through 15. [SKIP] skips the next box and [QUIT] aborts the whole operation.

[FIND] MENU looks inside the box(es) selected and reduces the box(es) to fit its(their) contents. Used to isolate letters, lines or blocks of text for shifting, aligning, copying, etc.

[AUTO FIND]

Automatically finds the contents of a box when it's grabbed. That is, it's the same as,

[BOX GRAB] (bitpad coords) [FIND]

and when used with [AUTO GRAB] saves keystrokes.

[ZAP BOX] Selected boxes are cleared to color code 0.

[INV BOX] Selected boxes are inverted, XORed with color code 15. This makes a negative of an original box.

[<--/UP/DOWN/-->]

Centers selected box(es) and must be confirmed by hitting twice.

[UP/DOWN] Vertically centers as above.

[<--/-->] Horizontally centers as above.

[UP] [DOWN] [<--] [-->]
[UP/-->] [<--/UP] [DOWN/-->] [<--/DOWN]

Direction indicators used in shifting, moving, aligning and adjusting boxes.

[XJUMP] Sets the horizontal increment (number of pixels) used by the shift and move commands.

[YJUMP] Sets the vertical increment as above.

[XYJUMP] Sets the horizontal and vertical increments together.

[1 JUMP] [5 JUMP] [10 JUMP] [50 JUMP]

Preset jump factors used regularly to move text around the screen.

[SCAN S] [SCAN D]

Allow the user to set scanners for source and destination boxes. Scanners are directional, they determine the order in which the contents of the box are read from and written to the display page.

[BT-LR] [TB-RL] [TB-LR] [BT-RL]
[LR-BT] [RL-TB] [LR-TB] [RL-BT]

These keys set eight possible scanner directions. The first four work column by column starting at one side of the box and working towards the opposite side. the second four work row by row starting at the top or bottom of the box and working towards the opposite side. Source and destination scanners default to [LR-TB]. For an example of scanners at their best see MIRROR below.

[SHIFT] Shifts a box (or boxes) AND ITS (THEIR) CONTENTS up, down, left and/or right by the number of pixels set with the jump keys (x and yjump default to 1), for example,

[CE] [0] [S]
[10 JUMP] [SHIFT] 2 x [UP/->]

will shift box 0 with its contents up 20 pixels and right 20 pixels.

[JUST] Justifies one or more boxes as selected,for example,

[CE] [0] [S] [F] [FF]
[JUST] [<-/UP]

will shift box 0 with its contents to the upper left corner of the screen.

[MOVE] Moves a box (or boxes) WITHOUT TAKING ITS (THEIR) CONTENTS, for example,

[CE] [0] [S]
[BOX GRAB] (bitpad coords) [FIND]
[5 JUMP] [MOVE] [<-/DOWN]

will find box 0 containing, say, a title and then move the box left 5 pixels and down 5 pixels without moving the title. See STREAKING below.

[ADJ] Adjusts the size or position of a box definition WITHOUT AFFECTING ITS CONTENTS, for example,

[CE] [0] [S]
[BOX GRAB] (bitpad coord) [FIND]
[ADJ] [EXPAND] 5 x [EXPAND]

will find box 0 and expand it 5 pixels a side. Individual sides can be expanded or shrunk using the direction indicators.

The next group of keys manipulate images by duplicating, combining and transforming the contents of a box or boxes. These operations include fade, average, copy, key, fuzz and color equal color. Each of these operations can be carried out in one of six modes - add, subtract, and, or, xor, replace. These modes apply to the color codes contained by the box or boxes operated on. Box 0 can be copied over box 1 in replace mode and the contents (color codes) of box 0 will replace those of box 1. However, box 0 can be copied over box 1 in or mode and the contents (color codes) of box 0 will overlay those of box 1. The contents of box 1 remain and aren't obliterated as in a replace copy. Similarly the xor mode can invert the copy, add mode sums the color codes in boxes 0 and 1, and so on.

[ADD] [SUB] [AND] [OR] [XOR] [REP]

Are function modes used in disk loads, and fading, averaging, copying, keying, fuzzing and color = color. Only one is enabled at a time (default is replace).

[AVE] "Average" is a means of combining images. For corresponding locations in the source and destination boxes the color codes are averaged and the result affects the destination box. The six function modes create various transformations. Boxes can be averaged within a page or from page to page.

[FADE] "Fade" is a slow average. For corresponding pixels in the source and destination color codes are compared. If source is greater than destination then the destination color code is incremented by 1. If they're equal the destination color code escapes unscathed. If destination is greater than source then the destination color code is decremented by 1. The logic above applies in REP mode; as for other modes, you're on your own.

[COPY] Copies one or more boxes over a corresponding box or group of boxes. Copying uses the color control vector and the page vectors. Boxes can be copied within a single page or from page to page. Therefore these control vectors must be set BEFORE the copy key is hit. Select box 0 and force it to point to box 1 on the same page. Only the colors selected by the color control vector will be copied from the source box(es). For example,

[CE] [0] [S] [F] [1]
[1-15 C] [REP] [COPY]

copies box 0 colors 1-15 over box 1 replacing its contents. But color 0 will NOT be copied therefore those areas of box 1 corresponding to the background of box 0 remain untouched. And

[CE] [-] [0] [7] [S] [A]
[1-15 C] [REP] [COPY]

copies boxes 0-7 from page to page. Again boxes 0-7 on the first page are considered source boxes and because the box control vector points to the alternate page the corresponding boxes on the second page become the destination boxes. In copying source and destination boxes are usually the same size. Boxes of unequal size will "smear" the image. Copy can be used in a macro to make multiple copies. More about this under STEP REPEAT.

[KEY] Keys one or more boxes over a corresponding box or group of boxes. Keying uses all the control vectors. As in copy, boxes can be keyed within a single page or from page to page. And, as in copy, control vectors must be set BEFORE key is hit. Keying can be thought of as the reverse of copying. Only the colors selected by the color control vector will be keyed on. In keying the color vector controls destination boxes not source boxes as in copying. For example,

[CE] [0] [S] [F] [FF] [0 C] [REP] [KEY]

will key box 0 into the background (color 0) of the full frame. ONLY color 0 will be replaced and colors 1-15 will remain unaffected. And

[CE] [0] [S] [A] [0 C] [REP] [KEY]

Keys box 0 from page to page as in copy. If the destination box is smaller than the source only the area defined by the destination box will be affected but if the destination box is larger than the source box will be repeated until it fills the entire destination box. This is very handy for creating background patterns.

[FUZZ] "Fuzz" is a powerful means of transforming an image. A more complete description appeared in an article by David Cook in Creative Computing, January 1983. The page control vectors determine source and desination boxes as in copy. Fuzz averages each source box color code with its immediate neighbours and uses the result in transforming the destination box. In replace mode fuzz smooths the image. It can generate patterns and surface texture. Boxes can be fuzzed with themselves (a single box is both source and destination), or from page to page. Same page as opposed to page to page fuzzing produces different results.

[CEQC] "Color equals color" is copy with additional color modification. For pixels in the source box enabled by the color vector, corresponding pixels in the destination box are set equal to the pen color. Color equal color is very useful, here's an example which converts a high resolution image to low resolution-

[LO RES]
[1-15 C] [PEN C] [15] [CEQC]

As in fade, the logic applies in REP mode; other function modes produce rather exciting results.

[3D] [3D X] [3D Y]

"Translation by intensity" describes this peculiar transformation. The pixels within a selected box (or boxes) are translated or shifted horizontally and vertically using this formula- number of pixels shifted = color code x [3D XorY]. If the X & Y multiplication factors are set to 1, the brightest areas of the image will be shifted 15 pixels horizontally and vertically, the background will not move. This creates the illusion of a three dimensional surface, hence the name 3D. This transformation redraws only thos e pixels enabled by the color vector and works best if color 0 is NOT enabled. Scanners and function modes can be invoked to produce wierd & wonderful effects. Setting the X & Y multiplication factors to large values produces instant abstractions.

The observant reader will note that [FADE] [AVE] [COPY] [KEY] [FUZZ] appear in more than one place on the bitpad. Directly above the function mode keys is the STAMP COMMAND AREA. "Stamping" allows the user to paint on the current page. Brushes are defined as boxes selected on the current page. The user touches the bitpad stylus to one of the stamp keys, sets the bitpad to stream mode and touches a brush. After identifying a brush the user stamps by touching the stylus to the bitpad. Wherever the stylus touches, the brush/box is copied centered on the stylus position. Stamping is used to create diagrams, for example, adding components to an electrical circuit or arrows to flow chart.

Above the stamp keys is the PAINT COMMAND AREA. "Painting" uses both pages; one page contains the brushes, the other becomes the canvas. The user defines his/her brushes on the brush page. It's possible to load from disk a selection of previously defined brushes. Then the user toggles to the canvas page and points to a brush (or brushes) with the alternate page vector, for example,

(1) [AUTOFIND] [AUTO GRAB] 16 x (bitpad coords)
(2) [P] [CE] [1] [A]

picks sixteen brushes, toggles to the canvas page and enables brush 1. As in stamping the user touches the stylus to one of the paint keys, sets the bitpad to stream mode and "paints" by touching the stylus to the bitpad. For stamping & paiting the brushes are treated as boxes, therefor ADD, SUB, AND, OR, XOR, REP funcion modes are available. And any combination of scanners can be set. The color vctor can be used to enable specific colors, for eaxmple,

(1) [0 C] [KEY PAINT]

will paint only on the background leaving the foreground intact.

[PEN] Allows the user to paint with a single pixel. [PEN C] sets the color of the pixel and [PEN FUN] enables one of the six function modes.

[PEN FUZZ] Allows the user to fuzz a 3 x 3 pixel neighbourhood centered on the stylus. The center pixel is averaged with its immediate neighbours. Thus the user can smooth or blend specific areas of an image, like touching up a photograph.

[PEN FIX] Allows the user to modify specific colors with the stylus. Pixels pointed to with the stylus and enabled by the color vector are set equal to the pen color. The user can clean up an image with this command, as above, like touching up a photograph.

A final observation on the power of fade, average, copy, key, fuzz, color equal color and load . These seven commands operate in any of six modes, a total of forty plus combinations. Any single color or group of colors can be selected. Multiple copies, fuzzes and averages can be created with macros. These operations can be applied to single boxes or groups of boxes. Various scanner combinations are available. There must be hundreds of ways to apply these operations. We'll examine a few after dealing with color and macros.

COLOR COMMANDS

[1/2] "One or two" toggles between a single color vector applied to both pages, and two color vectors, one per page. Indicators left & right of the COLOR MAP heading point to the active page(s).

[SET RGB] "Set red green and blue" allows the user to select the color(s) for the color code(s) enabled by the color control vector. Colors are determined by combining red, green and blue. These primary colors vary in intensity from 0 (no color) to 15 (full color). For example,

[CE] [-] [0] [7] [C]
[SET RGB] [0] [0] [0] - black
[15] [0] [0] - red
[0] [15] [0] - green
[0] [0] [15] - blue
[0] [15] [15] - cyan
[15] [0] [15] - magenta
[15] [15] [0] - yellow
[15] [15] [15] - white

There are 16 x 16 x 16 = 4096 possible colors.

[HUE] Allows the color map to be shaded; for example,

[CE] [0] [15] [C]
[SET RGB] [0] [0] [15]
[15] [0] [0]
[HUE]

creates a color map evenly shaded from blue (color code 0) to red (color code 15).

[LITE] Blacks out the screen then hilites each color code enabled by the color control vector.

[LITE ALL] Blacks out the screen and then hilites one by one all colors 0-15.

[GREY RGB] [GREY R] [GREY G] [GREY B]

"Grey scale" keys shade the color map for the appropriate primary color(s) from [0] through [15] or full color.

[? RGB] [? R] [? G] [? B]

"Random color" selects the color(s) for the color code(s) enabled in the color control vector by using the random number generator.

[+R] [+G] [+B] [-R] [-G] [-B]

Allow the user to fine tune any color(s) enabled by the color control vector. "R, G, B" stand for red, green and blue; + & - increase and decrease their intensity.

[ROLL DOWN] [ROLL UP] [<- ROLL] [ROLL ->]

These keys roll or rotate the intensity levels for enabled colors in the color map according to direction indicated. For example,

[CE] [ODD] [C] [? RGB] 8 x [ROLL DOWN]

randomly assigns colors to odd numbered colors. These colors are rolled through the map such that 1->3, 3->5, 5->7, 7->9, 9->11, 11->13, 13->15, 15->1. Eight rolls restores the original color map.

MACRO COMMANDS

[RUN] Followed by a number 0 - 15 invokes a predefined series of commands. The user can define up to 16 macros.

[LOOP] "Loop" sets the counter used for repeated execution of a macro.

[MEM] "Memory" causes MENU to remember the following commands.

[SET] Followed by anumber 0 - 15 terminates a macro definition. See RUN.

[DEL] "Delete" followed by a number 0 - 15 deletes the specified macro. The macro can be redefined.

APPLICATIONS

Remember FONT? We left FONT with 'Shazam!' on the screen. Let's find a box containing the word and center it on the screen -

```
(1)  [0] [S] [BOX GRAB] (bit pad coords) [FIND]
(2)  2 x [UP/DOWN <--/-->]
(3)  [GREY RGB]
(4)  [SAVE] "T0 (return)"
```

In line (1) we find "Shazam!" using the bit pad stylus. Line (2) centers the box in the frame. Note the centering key is hit twice to confirm a potentially destructive operation. Line (3) sets the standard grey scale, in this case white letters on a black background. Line (4) saves the centered word in disk file "T0.PIX".

Although the current version of MENU doesn't fill lettering, future versions will -

```
(1)  [FILL] (bit pad coords)
(2)  [INC] [LAST SAVE]
```

Hit fill and touch the bitpad stylus to the interiors of each letter in turn, line (1). When all the letters are filled, save the image in disk file "T1.PIX", line (2).

FIGURE 5 - Shazam!

We have created both outlined and filled versions of 'Shazam!' and saved them on disk. These two files become the building blocks for various special effects.

In the following applications I have included a number of sample programs. For clarity, these programs use more than the necessary number of keystrokes (the liberal use of clear entry for example). Since related keys are grouped together, the extra keys do little to decrease speed and probably increase overall efficiency by preventing errors.

SHADOW

This is a simple and widely used effect. Menu creates shadows as follows-

```
(1)  [REP] [LOAD] "T1 (return)"
(2)  [0] [S] [BOX GRAB] (bitpad coords) [FIND]
(3)  [5 JUMP] [SHIFT] 3 x [<--/DOWN]
(4)  [-S] [0 C] [PEN C] [7] [CEQC]
(5)  [+S] [CE] [15] [C] [PEN C] [0] [CEQC]
(6)  [OR] [LAST LOAD]
```

Line (1) loads the filled version of "Shazam!". Line (2), selects and finds box 0. We set a jump of 5 and shift box 0 15 pixels left and 15 pixels down, line (3). Line (4) selects full frame and replaces color 0 with color 7. Line (5) reselects box 0 and replaces color 15 with color 0. Now we OR load the original image over the background, line (6), and we have a shadow.

FIGURE 6 - SHADOW

STEP REPEAT

This is an effect used in slide animation and is made by repeatedly exposing an image while stepping it horizontally and/or vertically on the animation stand. This is how it's done with MENU-

```
(1)  [REP] [LOAD]   "T0 (return)"
(2)  [0] [S] [A] [BOX GRAB] (bitpad coords) [FIND]
(3)  [PSA] [S<-D] [P]
(4)  [1-15 C]
(5)  [MEM]
(6)       [COPY] [P] [MOVE] 3 x [UP/-->] [P]
(7)  [SET] [0]
(8)  [LOOP] [2] [0] [RUN] [0] [P]
(9)  [OR] [INC] [LAST LOAD]
(10) [XOR] [LAST LOAD]
(11) [OR] [DEC] [LAST LOAD]
```

Line (1) loads the outline replacing the image on the screen. Line (2) finds box 0 and points it to the opposite page. Line (3) transfers box 0 control info and coords to the alternate page. Line (4) sets the colors to be copied. Lines (5-7) create a macro 0 which copies "Shazam!" to the alternate page then moves the target box on the alternate page up three pixels and to the right three pixels. This macro is run twenty times, then we go to the alternate page, line (8). Finally, to clean up the image we cut out the original word, lines (9-11).

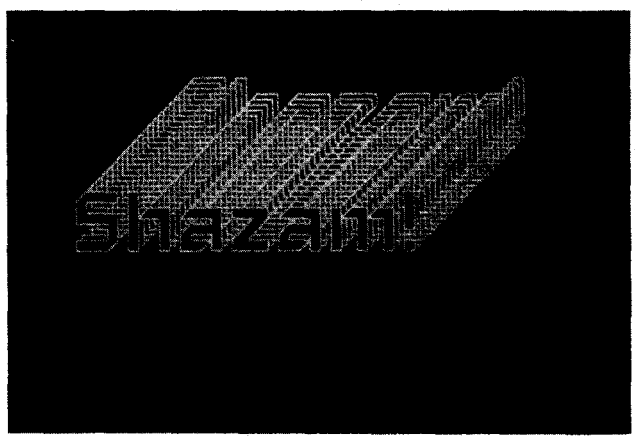


FIGURE 7 - STEP REPEAT

MIRROR

This effect is made by using two scanners moving in opposite directions, for example,

```
(1)  [REP] [LOAD] "T0 (return)"
(2)  [0] [S] [BOX GRAB] (bitpad coords) [FIND]
(3)  [F] [1] [D<-S]
(4)  [+S] [MOVE] (height + 1) x [DOWN]
(5)  [-S] [1-15 C]
(6)  [SCAN D] [LR-BT] [COPY]
```

Line (1) loads the outline replacing the image on the screen. Line (2) selects and finds box 0. Then we force box 0 to point to box 1 and transfer coordinate information, line (3). We select box 1, move it one pixel below box 0, line (4). Reselect box 0 and set colors 1 - 15, line (5). Finally, line (6), the destination scanner is set to move from bottom to top and box 0 is copied bottom up into box 1. We can enhance the mirror effect with fuzzing-

```
(7)  [SCAN D] [LR-TB] [CE] [S] [ZAP F]
(8)  [1] [S] [A] [D<-S]
(9)  [CE] [0] [1-15] [C] [FUZZ]
(10) [PSA] [COPY]
(11) [P]
```

In line (7) we reset the destination scanner and clear the select box and force vectors. Line (8) selects box 1, points it to the alternate page and transfers coordinate information to the alternate page. Then we set color 0 - 15 and fuzz box 1 to the alternate page, line (9). To complete the effect toggle to the alternate page, taking select box and alternate page, and the fuzzy box 1 back over the original, lines (10,11).

FIGURE 8 - MIRROR

NEON

Another common effect, neon is an interesting application of smoothing. It is usually made by combining focused and defocused versions of the same image. MENU does it this way -

```
(1)  [REP] [LOAD] "T1 (return)"
(2)  [0] [S] [A] [BOX GRAB] (bit pad coords)
     [PSA] [8<-D]
(3)  [1/2] [1-15 C] [P] [CE] [0] [1-15] [C]
(5)  [MEM]
(6)       [FUZZ] [P] [CEQC] [P]
(7)  [SET] [1]
(8)  [LOOP] [6] [RUN] [1]
(9)  [MEM]
(10)      [FUZZ] [P]
(11) [SET] [2]
(12) [LOOP] [6] [RUN] [2]
(13) [XOR] [LAST LOAD]
```

We begin, lines (1-3), by loading "Shazam!" filled and defining a box around the word. This box is larger than the word itself by about ten pixels a side. Box control info and coords are set on both pages. Line (4) sets seperate color maps, colors 1 - 15 on the alternate page and colors 0 - 15 on the original page. Now we define macro 1, lines (5-7), which fuzzes the image to the alternate page using all colors 0-15, then goes to the alternate page and sets colors 1-14 equal to color 15. We run this macro six times, line (8), expanding the original word to the maximum size of the neon glow. Now define a short macro to smooth the edges of the glow and run it six times, lines (9-12). Finally, we cut out the original word and voila a neon glow.

FIGURE 9 - NEON

We can refine the effect by adding in the original word outline and coloring the result as folows,

```
(14) [CE] [15] [C] [-PC] [-A] [CEQC] [+PC]
(15) [OR] [DEC] [LAST LOAD]
(16) [CE] [0] [14] [15] [C]
(17) [SET RGB] [ 0] [ 0] [ 0]
(18)           [ 0] [ 0] [15]
(19)           [ 0] [15] [15]
(20) [HUE]
```

Line (14) sets color 15 to 14. We reserve color 15 for the outline which we load in line (15). Set colors 0, 14 and 15 to black, blue and cyan respectively, lines (16-19), and finally shade the image, line (20).

STREAKING

Similar to step repeat, the title leaves behind a trail of copies each decreasing in intensity. MENU's version of streaking is as follows -

```
(1)  [REP] [LOAD] "T1 (return)"
(2)  [0] [S] [A] [BOX GRAB] (bitpad coords) [FIND]
     [PSA] [S<-D]
(3)  [P] [1-15 C] [PEN C] [1]
(4)  [MEM]
(5)       [CEQC]
(6)       [P] [MOVE] 6 x [UP/-->]
(7)       [P] [PC+]
(8)  [SET] [3]
(9)  [LOOP] [1] [3] [RUN] [3]
(10) [P]
```

As in neon we start with "Shazam!" filled, line (1). In line (2) we select box 0, point it to the alternate page, find it and toggle to the alternate page taking along box control and coordinate information. Returning to the original page, set colors 1 - 15 and set the pen color to 1, line (3). Lines (4-8) create macro 3 which copies the original image to the alternate page using the pen color, moves the box on the alternate page up 6 pixels and right 6 pixels, and last but not least increments the pen color. Run macro 3 thirteen times and toggle to the alternate page, lines (9,10).

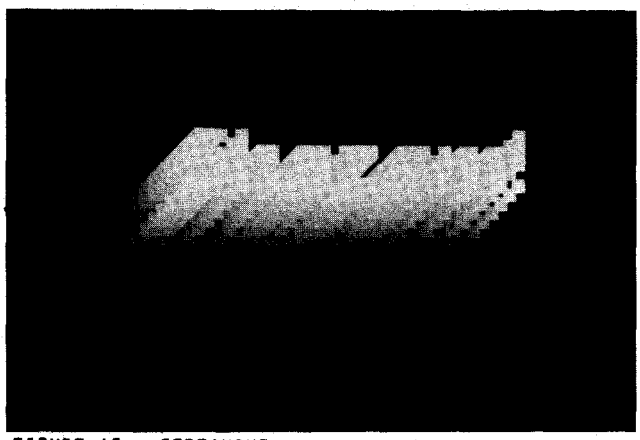


FIGURE 10 - STREAKING

METAL

Metal is a popular effect for jazzing up titles, logos, etc. Simulating a metallic surface is quite a challenge, we have several procedures which represent a good start. For example,

```
(1)  [REP] [LOAD] "T1 (return)"
(2)  [CE] [0] [S] [BOX GRAB] (bitpad coords) [FIND]
(3)  [CE] [15] [C] [PEN C] [1] [CEQC]
(4)  [OR] [DEC] [LAST LOAD]
(5)  [PEN C] [9] [CEQC]
(6)  [+S] [BOX GRAB] (bitpad coords)
(7)  [ADJ] [SHRINK] (height / 2) x [UP] [DOWN]
(8)  [ADD] [CE] [0] [1-15] [C] [PEN C] [2]
(9)  [MEM]
(10)      [C=C]
(11)      [ADJ] [EXPAND] [UP] [DOWN]
(12) [SET] [4]
(13) [LOOP] [5] [RUN] [4]
(14) [F] [0] [CE] [1] [C] [KEY]
(15) [ZAP BOX]
```

We load the filled "Shazam!" and make it box 0, lines (1,2). We change color 15 to color 1, OR in the outline "Shazam!" and change color 15 to color 9, lines (3-5). Below "Shazam!" make box 1 any number of pixels long but only one pixel high, lines (6,7). In line (8) we set ADD mode, enable all colors and set pen color to 2. LInes (9-12) are the metal bar macro which adds 2 to the contents of the box and expands it up & down. Line (13) runs the macro five times. Line (14) points the metal bar at "Shazam!" and keys the bar over color 1. Line (15) erases the metal bar.

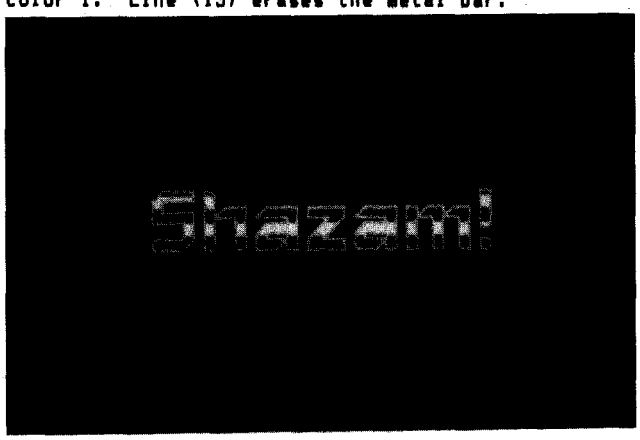


FIGURE 11 - METAL

LINES, BOXES AND GRIDS

So much for titles and logos, how do we create diagrams, flow charts, organizational charts and their ilk? We thought about writing a large and obscure software package but realized we already had one, MENU. With a few simple macros lines, boxes and grids can be created in any size and at any position; then these graphic elements can be moved, copied, keyed and combined in any number of ways. We have created a special font or typeface of arrowheads and other graphic symbols. Using FONT we combine these symbols and normal word copy with lines and boxes created with MENU to produce finished charts. To create a line define a box in the approximate position of the desired line, adjust its length and width and, finally, fill it with an appropriate color code. For example,

```
(1)  [CE] [0] [S] [BOX GRAB] (bitpad coords)
(2)  [ADJ] (as required)
(3)  [1-15] [C] [PEN C] [7] [CEQC]
```

Creating a filled box is exactly the same as creating a line. Creating an outlined box is slightly more complex, as follows -

```
(1)  [+S] [BOXGRAB] (bitpad coords)
(2)  [ADJ] (as required)
(3)  [INV BOX]
(4)  [ADJ] 2 x [SHRINK]
(5)  [INV BOX]
```

This procedure creates a box with an outline one pixel wide. The outline is in color 15. A variation on this technique creates an outline around a title or label, as follows -

```
(1)  [+S] [BOX GRAB] (bitpad coords)
(2)  [ADJ] 6 x [EXPAND]
(3)  [INV BOX]
(4)  [ADJ] 2 x [SHRINK]
```

To hilite a word or number, find the box containing the copy, expand it, and invert. Grids are created by embedding the line definition inside a macro. To create a grid of vertical lines use the following procedure-

```
(1)  [+S] [BOX GRAB] (bitpad coords)
(2)  [ADJ] (as required)
(3)  [INV BOX]
(4)  [10 JUMP] [MEM]
(5)     [MOVE] [-->]
(6)     [INV BOX]
(7)  [SET] [5]
(8)  [LOOP] [2] [0] [RUN] [5]
```

A grid of horizontal lines is made by altering the original line definition, line (1), and the macro, line (5). A grid composed of both horizontal and vertical lines can be made by combining the two macros, as above, or as follows -

```
(1)  [+S] [BOX GRAB] (bitpad coords)
(2)  [ADJ] (as required)
(3)  [F] [FF] [JUST] [<--/UP]
(4)  [INV BOX]
(5)  [ADJ] [SHRINK] [UP/-->]
(6)  [INV BOX]
(7)  [ADJ] [EXPAND] [UP/-->]
(8)  [C 0]
(9)  [REP] [KEY]
```

We begin by defining a box equal in size to the horizontal and vertical spacing of the grid, lines (1-2). Move the box to the upper left of the frame, line (3). Invert the box, move the left and bottom edges in one pixel, and reinvert the box, lines (4-6). This produces lines on the left side and bottom of the box. Restore box 5 to its original size and key it back over the full frame on color code 0, lines (7-10).

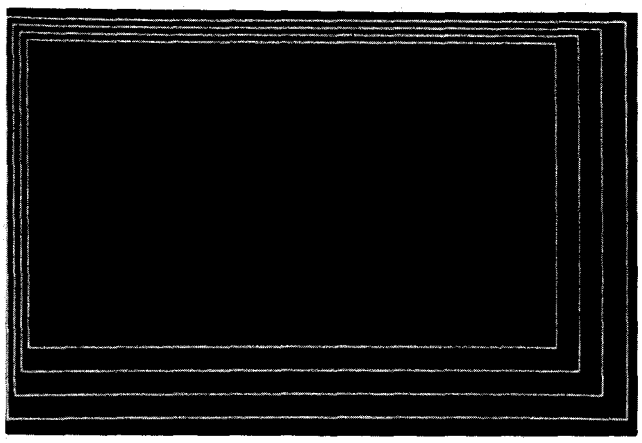


FIGURE 12 - GRIDS

With these examples we have scratched the surface of MENU. The user controls 4096 colors and 16 boxes per page. He/she can recall any number of images stored on disk. Function modes include: add, subtract, and, or, xor, replace. These modes can be invoked for any disk load, fade, average, copy, key, fuzz, and color equal color. These operations apply to boxes individually or in groups, they can be filled, inverted, cleared, shifted, justified, adjusted, equivalenced, assigned and found. Colors are set, shaded, raised and lowered in intensity, separated into red, green and blue components, equivalenced and transformed. Color codes are used by the function modes to produce effects such as smoothing, edge detection, surface texture, etc. MENU allows the user to define macros which become higher-level operations and can be used to define grids, shading, animation effects, etc. Everyday we learn something new.

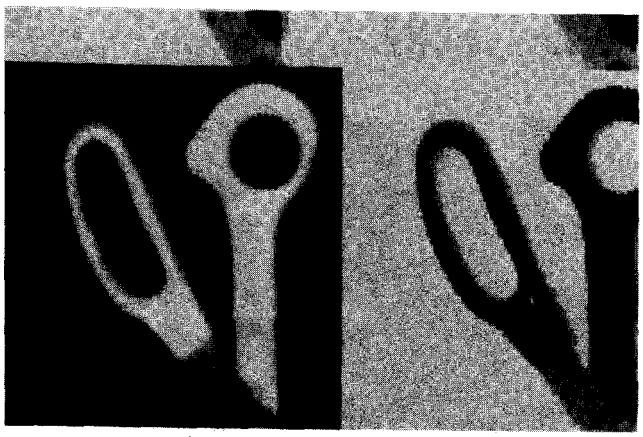
Artistic expression is difficult to achieve in any medium. Computer art generally emphasizes technique over expression. Artifice is not art. Artistic expression is akin to personal expression, an artist develops his/her own style. Technique is important as a starting point and as a foundation to build upon. MENU demands this technical skill. In turn, MENU offers the potential for developing individual style. To illustrate the how and why, I will describe some of my own work in more detail.

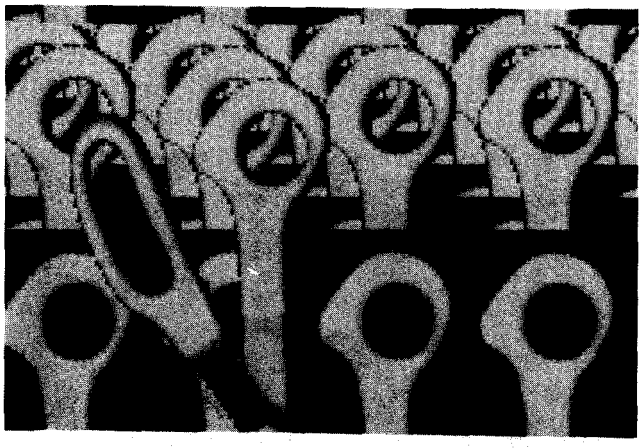
SCISSORS - STILL LIFE IN THE COMPUTER AGE

An important component of our system is the Via Video digitizer. It digitizes the image from a b&w video camera and loads that image onto either page 0 or page 1. In the process of digitizing, the original video image is separated into 16 grey levels, color codes 0 through 15. With the color map shaded from black to white, we can take snapshots of any subject.

An artist has to start somewhere and still life seemed like a good starting point. There weren't any flowers in the studio so I used scissors. Computer graphics like any other art form is structured by its medium. The medium is an important part of the message, to amend McLuhan. The artist creates meaning in his work by relating structure (the medium) and content (the subject). What do scissors connote? Scissors cut. Scissors are made of metal, hard and shiny. Scissors are mass-manufactured, all conforming to a single mold. Standing upright, they remind me of soldiers. Scissors imply paper.

How are these impressions translated into MENU commands? Depicting scissors cutting paper is easy. A piece of paper is made by assigning a box and filling it with color 15. One blade of the scissors is removed by assigning a second box and then erasing it. The scissors are keyed over the paper. An edge is enhanced using edge detection. Load the image normally in replace mode, then load it in xor mode and average the result back into the original image. A metallic surface is produced by shading all, or part, of the color map from black to white. In the scissors series, colors 1 through 13 are shaded and colors 0, 14 and 15 are used to add color to the background. Creating a feeling of regimentation is done with keying. A box is assigned around the handles and this box keyed over the full frame. Scissors imply paper and together they imply collage. By creating mattes, keying patterns over these mattes and recombining the resulting 'cutouts', I created a series of collages.





FIGURES 13,14 - SCISSORS

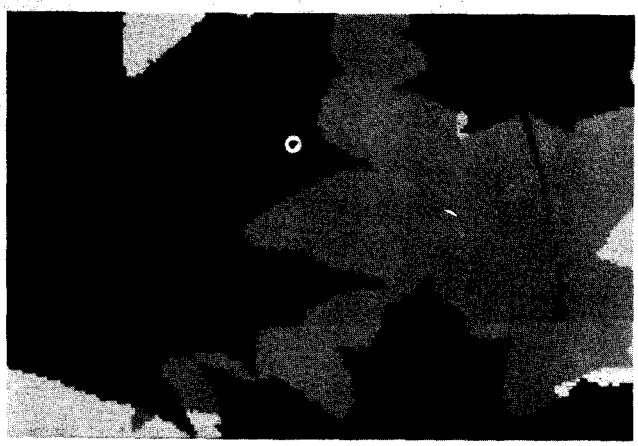
LEAVES, TREES AND CREEPER VINES

On to greater challenges, I began playing with leaves. This led to trees, creeper vines and, finally, a forest. Starting with real leaves and the video digitizer, I collected several images. I was thinking about graphic design and the posibility of using computer generated images as illustrations for advertising, brochures, posters, etc. And this means simple, straightforward, probably representative images.

I decided on a structured approach. for each real leaf image, I created a two color mask setting colors 3 through 15 equal to color 15 and colors 0,1,2 equal to color 0. Then I "edge detected" the masks to produce outlines. There are two ways to do this, either a SUB fuzz to the alternate page or

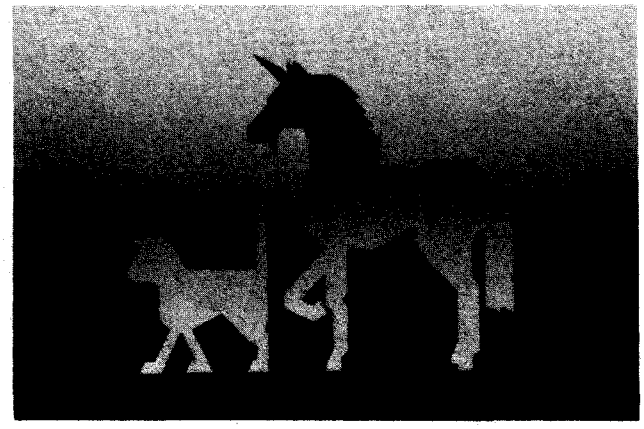
an XOR fuzz to the alternate page. Again the outline is saved in two colors 0 & 15. Finally, I saved XOR, SUB and AND fuzzes for each image, to be used as texture and pattern. I produced four masks, four outlines and twelve textures. Now to combine them in interesting ways.

I tried four color patterns. Taking two masks, one on each page, I set mask 0 to color 5 and mask 1 to color 10. Adding them together the background remains color 0 but the areas of overlap become color 15. I played with positive and negative space. REP load a mask then XOR load a second mask over it. The areas of overlap are inverted to become background. I experimented with combining pieces of images with masks and outlines. REP load real leaf 0. OR load the mask of leaf 1 immediately followed with an XOR load of the same mask. Now OR load the outline of leaf 1. This cuts leaf 1 out of leaf 0 leaving only the outline of leaf 1. I tried more complex combinations.



FIGURES 15,16 - LEAVES

Now for the forest. I cut tree shapes and leaves out of graph paper and arranged them on the copy stand. I saved several images of leaves and tree trunks. For an added touch of realism I added string to the graph paper tree trunks, et voila, creeper vines. I used XOR, SUB and AND fuzzes to create leaf and bark textures. I developed a series of forest scenes by combining all or part of masks, outlines and textured images.

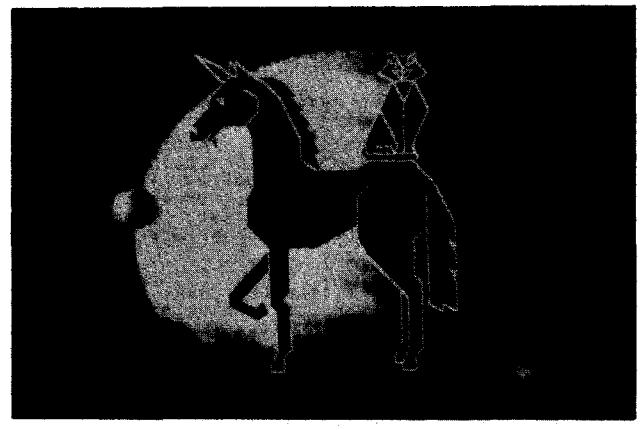


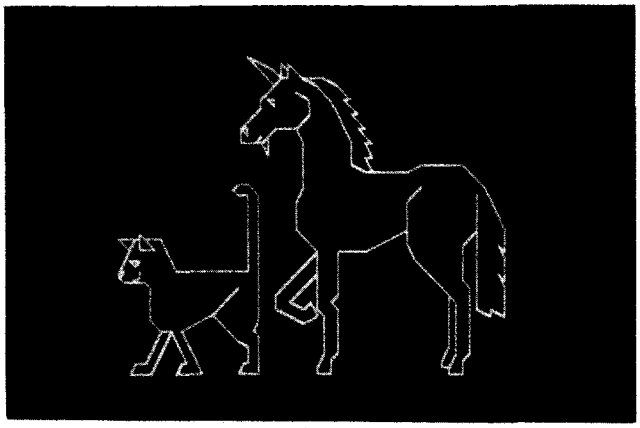
FIGURES 17-20 - FOREST

ANIMALS - UNICORN & CATS

A forest is populated by animals. Obviously, I needed animals. Using FONT, remember FONT, I created a unicorn and four cats. They reside in a file called ANIMALS.FNT and being symbols can be used like text, that is, scaled, rotated, drawn with a variety of pens, filled, etc.

I began with a unicorn and a cat. I was thinking about illustrations or perhaps images to be used in an interactive fantasy or adventure game. Using MENU's box commands I perched the cat on the unicorn's back. I saved the outline, filled it and saved the mask. David had been playing with a "dither" algorithm for shading a box. I took a dithered frame, shaded up and down from black to white, and XORed in the unicorn & cat mask. Simple but effective. I XORed in a planet mask. I generated various fuzzes using the animals and the planet. And I pieced together masks, outlines and fuzzes.





FIGURES 21-23 - UNICORN & CAT

Then it was time for lots of cats. Starting with a frame of several sized cats, I arranged rows of same sized cats; the largest cats at the bottom of the screen and the smallest at the top. Each row was horizontally centered and saved seperately as an outline. I filled the outlined images and saved a mask for each row. Beginning with the smallest cats, the row at the top of the screen; REP load the mask, set color 15 to color 6, OR load the outline and set color 15 to color 14. Then I lay in the next row; OR load the mask, set color 15 to color 7, OR load the ouline and set color 15 to color 14. Repeat this process for each row. Now we have a lot of cats. Color 14 can be reset to color 15. The rows are shaded from color 6 in the background to color 10 in the foreground (there are five rows).

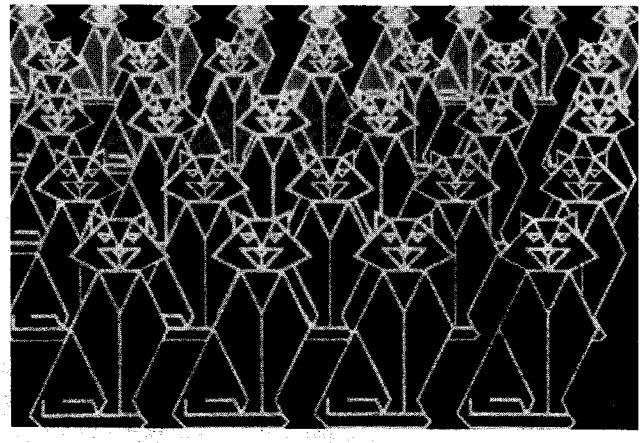


FIGURE 24 - LOTS OF CATS

So far, I've discussed representative images but my personal preference is for abstract images.

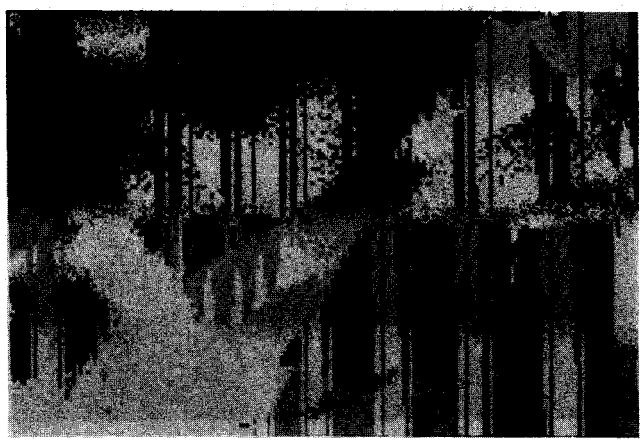
FEEDBACK - AN ELECTRIC PAINTBRUSH

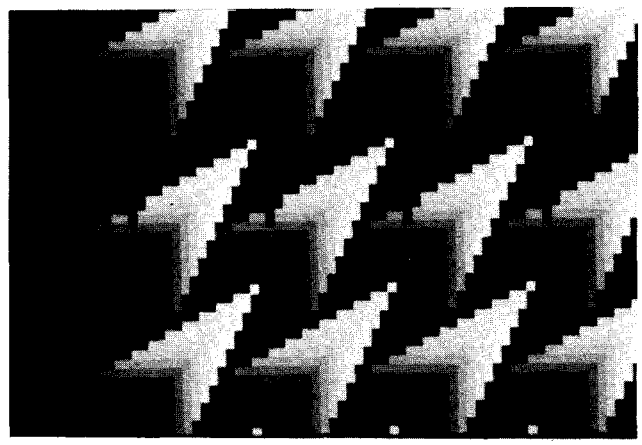
Abstract Expressionism is my favorite school of painting. The content or subject is spiritual, the soul of the artist. It recognizes feelings and emotions; defines structure in terms of balance, pattern, rhythm, harmony, contrast, counterpoint, etc; and defines beauty as appropriateness of form (content plus structure).

Video feedback has interested many artists in recent years. It produces detailed and dynamic images which symbolize for many cybernetics, the computer revolution, and so on. Its basic structure is recursive, an important programming technique of some philosophical interest. A feedback image is produced through a combination of controlled and random elements. Again, for many this symbolizes the electronic age.

My feedback images are created by pointing the b & w video camera at its own monitor. Noise and sources of illumination reflected on the monitor screen will be repeated within the loop, like a hall of mirrors. This convential feedback loop is modified by our column by column scanning process. The image is controlled by varying camera position, lens settings, and monitor brightness and contrast. After considerable tinkering, acceptable images are selected and stored.

I use a panoply of techniques to develop these basic b & w images. Fuzzing with either the subtract or xor mode, produces an edge detected outline. This outline can be averaged back into the original image in various modes. Fuzzing in add mode then reloading the original image in and mode, produces cutouts. These can be used as mattes for collaging. Sections of the image can be assigned to boxes and these boxes keyed over the full frame to produce foreground and/or background patterns. Color codes can be transliterated. I clean up a complex image by setting color codes 1 - 3 to 0, 5 - 7 to 4, 9 - 11 to 8 and 13 - 15 to 12. Selecting colors can be approached in many ways. Often I randomly select colors 0 and 15, shade the color map to produce an overall tone and, finally, randomly select only the odd or even colors.





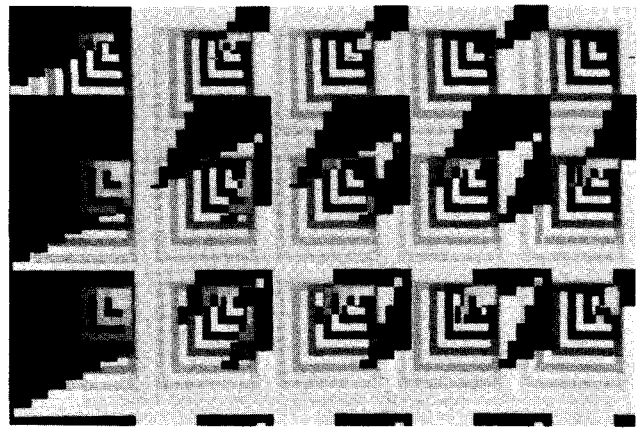
FIGURES 25-28 - FEEDBACK

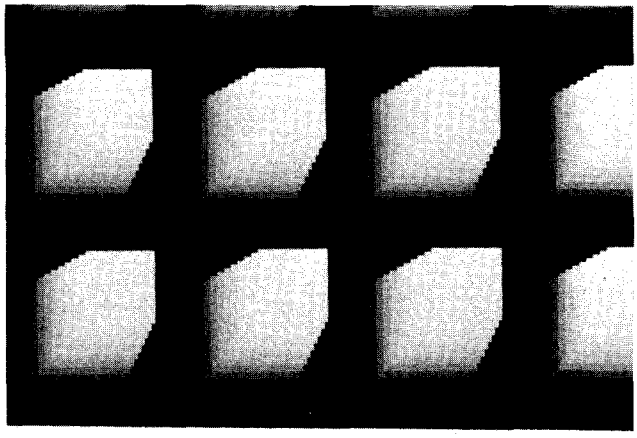
For me, using MENU in this manner parallels the process of painting. I am directed by my intuition. I respond to the image as it develops on the color monitor. I can store an image I'm having trouble with and return to it later. I can make false starts, back up, and start again. I can develop several images simultaneously. The computer is my studio.

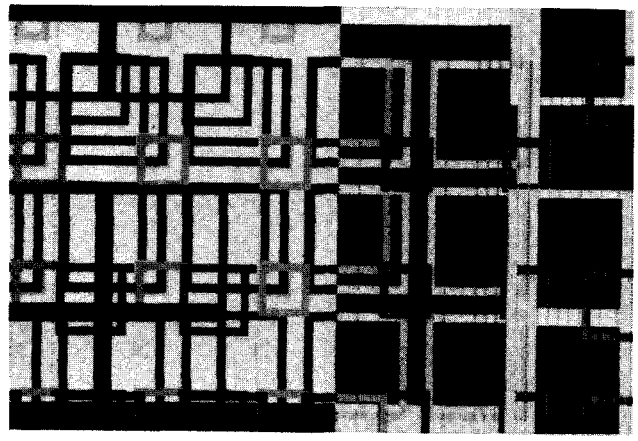
GEOMETRICS - THE ART MACHINE

In an effort to popularize the artistic potential of our system, I am creating a series of images called geometrics. These lie somewhat closer to applied or commercial art. They can be used as book covers, illustrations, wall panels, silk screens, fabric patterns, and so on.

Appropriately, the techniques used to create these images closely resemble those used in producing commercial slides. Basic rectangular shapes are created by defining boxes. These boxes are shifted, adjusted and filled with color. I make extensive use of MENU's macro capability to develop simple repetitive series and progressions. Edge detection, outlining and keying are used to enhance the basic shapes. Rarely, images are softened by smoothing.







FIGURES 29-32 - GEOMETRICS

I feel there is great potential in the geometric image. We will add new shape to MENU. This will expand MENU's vocabulary and capabilities in art and commercial graphics.

A BRIEF EPILOGUE

The moral is computer graphics has unlimited potential for artists. It demands a great deal from the artist, it returns an even greater reward.

Author Index

PROCEEDINGS

3rd symposium on small computers in the arts

ISBN 0-8186-0499-9
IEEE CATALOG NO. 83CH1930-7
LIBRARY OF CONGESS NO.83-81625
IEEE COMPUTER SOCIETY ORDER NO. 499

IEEE COMPUTER SOCIETY PRESS