MARK WILSON

DRAWING WITH COMPUTERS

THE ARTIST'S GUIDE TO COMPUTER GRAPHICS

- *complete course in fundamental techniques for creative drawing with microcomputers*
- *instructions for writing your own graphics software in BASIC*
- *survey of affordable computer graphics equipment*

DRAWING WITH COMPUTERS

MARK WILSON

DRAWING WITH COMPUTERS

A PERIGEE BOOK

A NOTE ABOUT THE FLIP IMAGES

AS A WAY of illustrating the themes of this book, the upper right-hand corner of each page spread contains a line drawing. These are individual illustrations, but they are also part of an evolutionary sequence. Flip the pages and watch what happens. The device is as old as the hills, borrowed from Eadweard Muybridge's photographic sequences, movie animation, and every schoolchild's dog-eared notebook. Each of these images was generated with the software that will be discussed and explained in this book.

PERIGEE BOOKS
are published by
The Putnam Publishing Group
200 Madison Avenue
New York, New York 10016

Designed by Helen Barrow

Library of Congress Cataloging in Publication Data

Wilson, Mark, date.
Drawing with computers.

Bibliography: p.
1. Computer drawing. I. Title.
NC740.W5 1985 760 85-3670
ISBN 0-399-51136-9

Printed in the United States of America
1 2 3 4 5 6 7 8 9 10

PREFACE

THE PREFACE is intended to be the first thing you read, but it is my final task in writing this book. Computer graphics and art are both vast and complex subjects; to write about the melding of them is not easy. While relatively little has been written about computer artwork, already there is much to be said and there will be even more to be said in the future. Thus, this book can only serve as an introduction to an evolving and dynamic field.

The book contains an overview of the common types of hardware used in computer graphics. Examples of simple software routines are also included. These routines are only examples and do not incorporate error checking or error handling. Most of these programs are not even complete, but rather are intended as samples which you can build, embroider, and embellish to your heart's content.

My interest is in static, two-dimensional artwork, and the discussions and illustrations in this book reflect that fact. A great deal of the power and versatility of the computer lies in its ability to create moving, animated images as well. I have not discussed the possibilities of cinematic imagery, but I would certainly not pretend that they are not very important. If your interest lies in this other dimension, I heartily encourage you to pursue it.

A number of artists who use the computer have graciously allowed me to reproduce their work. I am in their debt. Several manufacturers have also provided photographs for which I am thankful. My three sons, Seth, Webster, and Winfield, have kindly endured and tolerated their father's computer madness. Finally, I must thank my wife, Pamela, for her support and help.

CONTENTS

INTRODUCTION

"A picture is worth 1024 words . . ."
Quoted in *The Computer Image*,
by Donald Greenberg, et al.

A ROAD to Point Reyes stretches toward the bay. The road bends behind a rocky outcropping. Double rainbows arch over islands in the bay. It appears to be a spring day. Puddles of water stand by the roadside, reflecting the blossoms of forsythia that grow next to the highway. The islands and the distant shore are veiled with a thin layer of fog.

Fig. 1, *The Road to Point Reyes*, is a computer-generated picture. It was not painted by an artist standing in front of an easel, carefully studying and sketching the landscape. Nor was the picture made with a camera. This image, *The Road to Point Reyes*, was created in a way that is completely different from all previous techniques of creating images. This picture was generated by a computer using software by a team of computer scientists-programmers-artists. This image is a product of extraordinary technical complexity. Mathematics, art, and computer science were all wedded together in communal picture making.

> This landscape was defined using patches, polygons, fractals, particle systems, and a variety of procedural models. The various elements were rendered separately and later composited. Rob Cook designed the picture and did the texturing and shading, including the road, hills, fence, rainbow, shadows, and reflections. Loren Carpenter used fractals for the mountains, rock, and lake, and a special atmosphere program for the sky and haze. Tom Porter provided the procedurally drawn texture for the hills and wrote the compositing software. Bill Reeves used his particle systems for the grass and wrote the modeling software. David Salesin put the ripples in the puddles. Alvy Ray Smith rendered the forsythia plants, using a procedural model. The visible surface software was written by Loren Carpenter, and the anti-aliasing software by Rob Cook. The picture was rendered using an Ikonas graphics processor and frame buffers, and was scanned on a Color Fire 240 . . . the resolution is 4K × 4K, 24 bits per pixel. [*Computer Graphics*, vol. 17, no. 3, July 1983, p. 420.]

The image *Point Reyes* is an extreme example of computer graphics. It is extreme because it represents the most advanced and technically sophisticated use of the computer to create images. Not all computer graphics rely on a committee to make a picture, and not

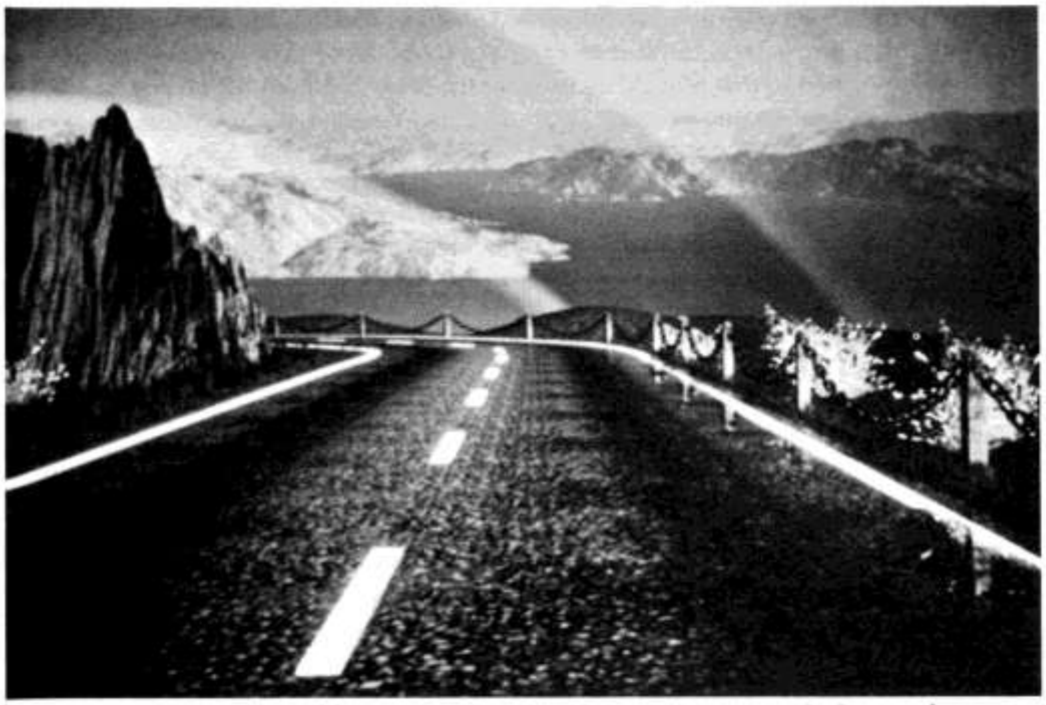


FIG. 1. ***The Road to Point Reyes,*** **1983 screen photograph, © 1983 Lucasfilm Ltd. Photograph courtesy of Lucasfilm.**

all computer graphics are intended to render impressionistic landscape scenes. But this picture may convey some of the subtlety and extraordinary range of computer graphics. Animated science-fiction movies, television commercials, and video games are instantly familiar examples of the uses of computer graphics. Less familiar, but nonetheless widespread, are engineering, architectural, and scientific applications of computer graphics. Slowly and persistently, artists have begun to explore and experiment using computers.

Computers have become an active component of our lives in only the last thirty or forty years. Just as children, when still very young, discover drawing and painting as a way of expressing themselves, the computer was also used to make pictures at an early stage. Visual information has great utility to all, whether children or adult computer programmers.

Since computers are ultimately numeric machines, it has always been vital to have the results—or the output—of the computer in visual form that is readily understandable to the people using the computer. Teletype machines were originally used to type information into and out of the computer. Later, modified television tubes were used to display the output from the computer. About 1955, the first actual drawing machines were used with computers. Both the IBM Corporation and California Computers (CalComp) developed devices called plotters. These machines could draw lines on sheets or rolls of paper and were controlled by computers. About the same time, the oscilloscope tube was modified so that it could be used for the monochromatic display of graphic information. Television technology was developing rapidly, and the television set was adapted to display computer-generated information. Eventually color television technology became incorporated into the multihued displays of computer graphics. As these hardware developments occurred, the software necessary to fully exploit these machines was also being written. By the late 1960s computer graphics was no longer a curiosity but was beginning to have widespread application in industry and science. It was particularly useful in industrial design and engineering.

Parallel to the commercial, industrial, and scientific use of computer graphics, another profound development was occurring. The Intel Corporation, which manufactured miniaturized electronic circuits, developed a new circuit that was etched on a small piece of polished silicon. This new circuit contained the entire main processing unit of a computer. Although this circuit, the 8008, was not very powerful, it was to transform our electronic technology. The small, cheap computer was born. Since the invention of the 8008 in 1973, microprocessing chips have become much cheaper and much more capable. With the advent of inexpensive microcomputers, millions of people have begun to explore the potential of these machines. It is against this background that we will discuss the possibilities of art making with the microcomputer.

WHY TRY TO DRAW WITH A MICROCOMPUTER?

THE SPANISH CAVES of Altamira are a favorite starting point for any history of art. While this is hardly a history, let us not fail that venerable tradition. On the walls of Altamira, prehistoric people painted and drew surprisingly realistic images of ice-age animals. No one is certain what the purpose of these images was, but there is a general assumption that the pictures must have had some connection to the hunt. They may have been believed to have some magical or ritualistic power. They may have been venerated as religious icons. Whatever the true purpose of the drawings, it is notable that they were made deep in the caverns.

Picture making, as far as anyone can determine, is older than any written language and represents an important human activity. With the frenzied proliferation of images in today's world, it is good to remind ourselves that image making is a profoundly important and ancient human activity.

A toddler is learning to draw pictures. How children learn and how they learn to make pictures are complex processes, but one thing is immediately obvious: The tool, whether brush or pen or stick, moves across an empty field and leaves a trace. This elemental process is a source of wonderment to the child. It is also a source of wonderment to the mature artist. Give an artist an unfamiliar tool and watch the pleasure—and perhaps frustration—as he or she learns and discovers the virtues and vices of the new art-making instrument. It is like the eternal satisfaction we all feel when drawing in the untouched sand of the beach. Drawing and painting are activities that, quite apart from the intellectual and aesthetic, have a purely tactile and sensory quality.

While the child—small or large—is delighting in the pleasure of manipulating the brush or stylus or whatever, something else is simultaneously occurring: Pictures depict. The things depicted can be simple or complex, from the logic diagram of a computer, a cartoon of a politician, a preparatory drawing for a painting, to a crude outline that a child regards as a picture of a dog.

People make pictures for many different reasons. The logic diagram of a computer is intended to explain, or perhaps analyze, in a schematic manner, the relationships of different objects or processes. The political cartoon communicates in another way. It may support a narrative description of people and events. It may also satirize these things *not* by faithfully depicting the relationship of people or things, but by selectively exaggerating and distorting them.

Pictures depict in a variety of ways. The depiction can be as simple as the child's drawing or as subtle and complex as Rembrandt's etchings. At the same time, modern life confronts us with a bewildering quantity of images. Consider the supermarket. The shelves are lined with thousands of cleverly and subtly designed packages. Television, magazines, signs, and books all present a similiar spectacle.

What does any of this have to do with computer art? Computer art and computer graphics are an unusually powerful and versatile new tool for the artist to use. Generating pictures with a number-processing machine is a wholly new method of image making. For the visual artist, this is exciting news. It is like having a new set of colors to use, or having a new brush unlike any other.

Because this way of making pictures is new, it is easy to be like the small child who is enraptured watching the trace of the pen. At a very early age, without hesitation, children will draw a square or a circle or at least an approximation of such a figure. But try to program a computer to draw a square or circle! While it is not a profoundly difficult task, it is not a trivial one either. It requires skill and thoughtfulness. And because all computer graphics are governed by the media of mathematics and computer languages, even simply drawing a square calls any previous assumptions about drawing squares into question. Thus, to my way of thinking, computer-generated art offers us an extraordinary opportunity. It teaches us a new viewpoint. We must look at our old assumptions in a new light. In a visual world where images are a cheap commodity, we are forced to look upon the simplest task in a fresh and exciting way.

Fig. 2 shows a drawing by Harold Cohen and his drawing program, AARON. It provides a good example of the complexity that exists behind computer pictures. Cohen is an accomplished British abstract painter and draftsman who became involved with computers

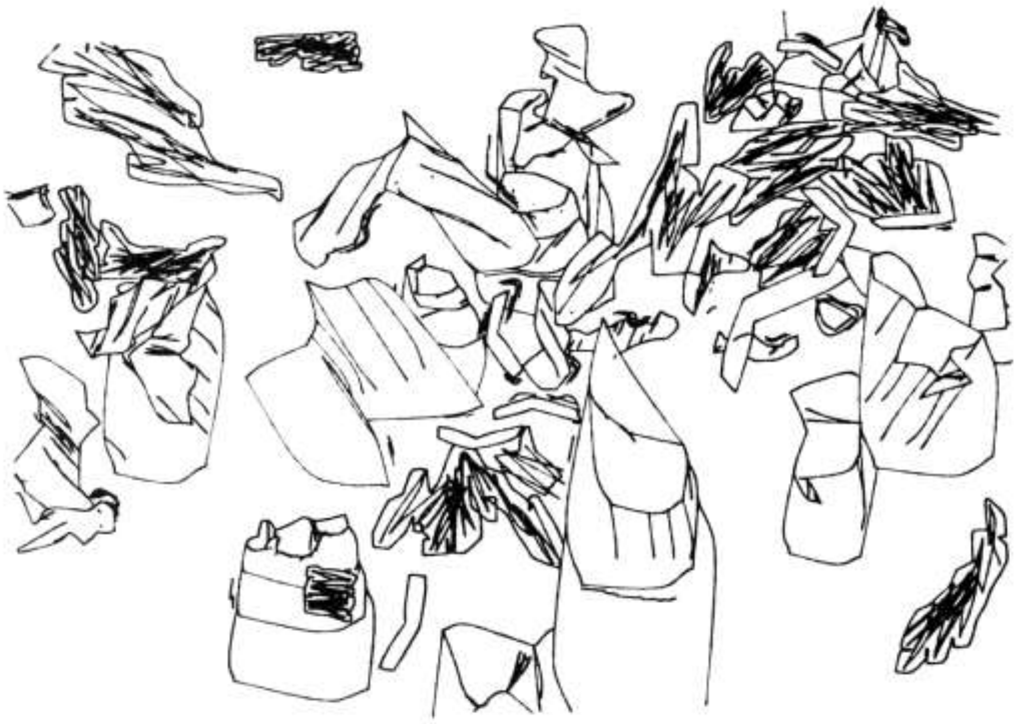


FIG. 2. Harold Cohen. 1983 black-and-white AARON drawing, india ink on paper, 22" x 30". Photograph © 1983 Becky Cohen.

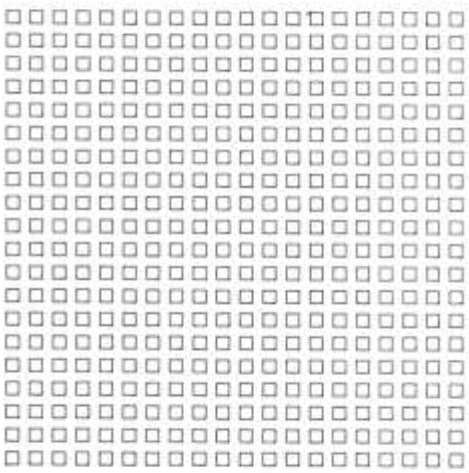
in the late 1960s. AARON could best be described as an automatic drawing program. When the computer and pen plotter are turned on, the machines proceed to make drawings. Aside from paper changing, the program runs without any input or intervention from the artist. The AARON program produces drawings quite unlike what is commonly thought of as computer art. AARON is, perhaps, a bit like our small child.

Designing and constructing the set of procedures—algorithms—that control the AARON program represent an extremely sophisticated level of computer science. In writing this program, Cohen employed many advanced concepts of artificial intelligence. While the drawings may look easy to make—indeed, they hardly look as if they were made by a machine—the assumptions and procedures that comprise the AARON program are anything but simple. Consider the elemental act of drawing a line from one point to another. AARON moves the pen in a series of sectors from point to point.

> SECTORS produce a series of "imagined" partial destinations—signposts, as it were—each designed to bring the line closer to its final end state. On setting up each of these signposts it passes control to CURVES, whose function is to generate a series of movements of the pen which will make it veer towards, rather than to actually reach, the current signpost. [Harold Cohen, "What Is an Image?"]

Suddenly, the trivial task of moving our pen from one point to another takes on a complexity that we may have never before considered. It is not that the computer is stupid, which it is, but rather that we are smarter than we thought.

In the meantime, the task at hand will be somewhat easier if we proceed to use simple, step-by-step procedures to develop computing skills and programs. As we progress, we will discuss the application of these skills to art making.



1. WHY USE COMPUTERS FOR ART MAKING?

THIS QUESTION has both a positive and a negative aspect. The artist has a wide range of media in which to work—oil or acrylic paints, water colors, pencil, pen, chalk, even airbrush. Sculpture employs a similar diversity of methods and materials. But types of art media have tended to be somewhat limited by an embedded tradition, as well as archival considerations. The twentieth century has seen much experimentation with artistic media, and the computer is only one of many recent options available to the artist.

Image making with computers has great advantages. Because of the peculiar way pictures are generated with the computer, we gain benefits unavailable with any other medium. Images can be generated quickly and cheaply. Once images are generated as a series of numbers stored in the computer, they can be manipulated with a bewildering variety of computational techniques. The new visual technologies associated with computer graphics offer new aesthetic insights. The intense inner glowing of the color phosphors on a high-resolution monitor are beautiful. Video artists have awakened our sensibilities to the

FIG. 1. Manfred Mohr. *P161* #6, 1973 plotter drawing, 17" x 17". © 1973 Manfred Mohr.

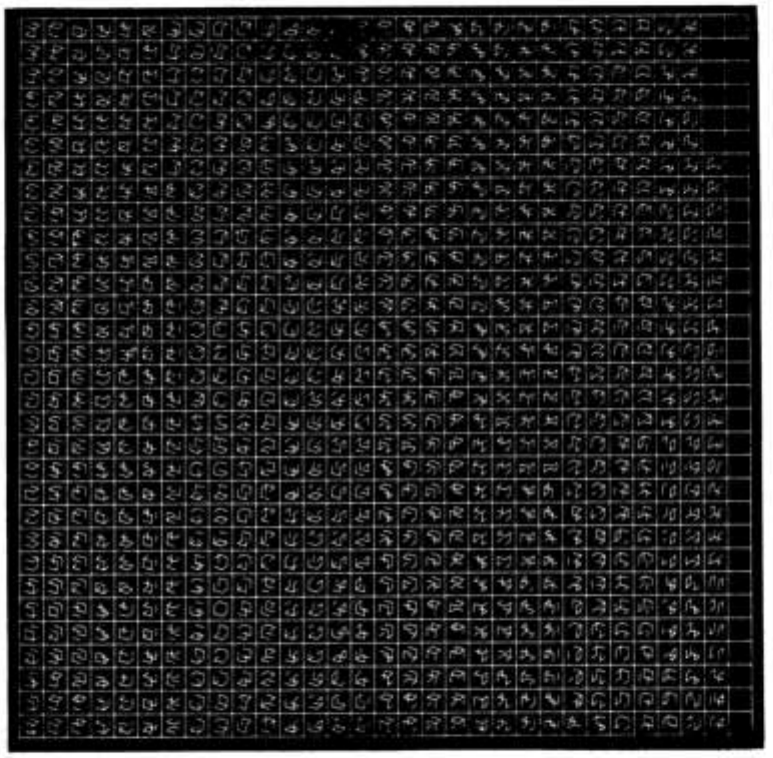
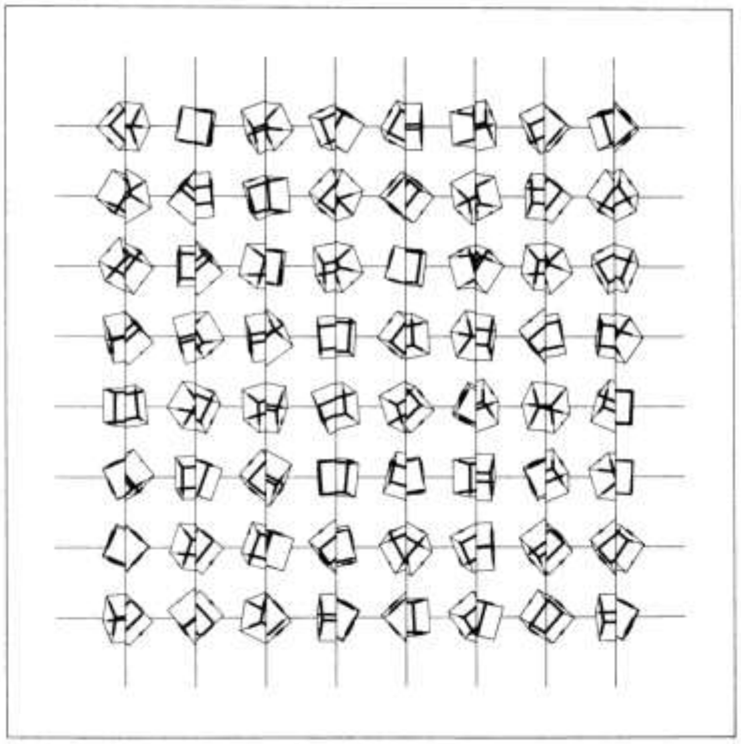


FIG. 2. Manfred Mohr. *P197 A*, 1977 plotter drawing, 24" x 24". © 1977 Manfred Mohr.



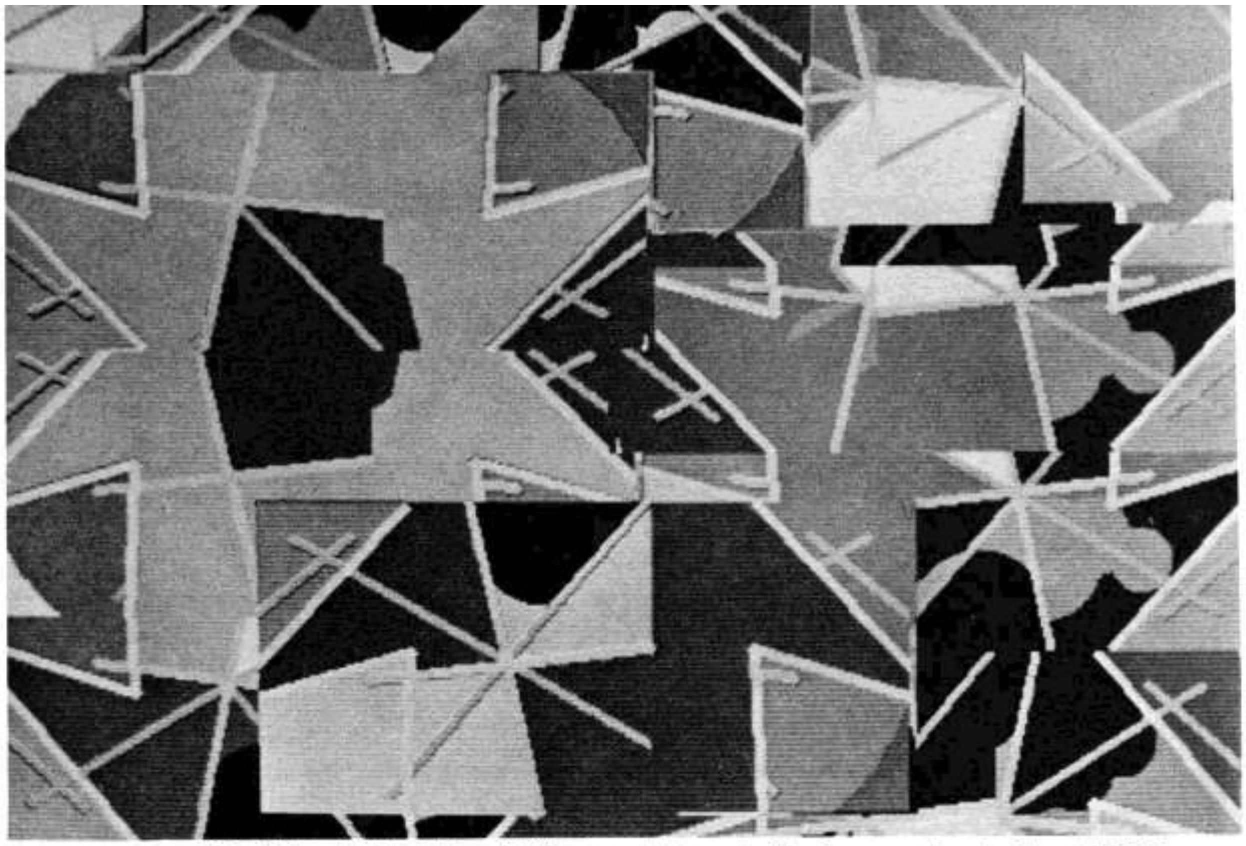


FIG. 3. Harry Holland. *Krazy3,* 1983 screen photograph of an image produced with an AED 512 color display controller, using a DEC LSI-11 as host processor. © 1983 Harry Holland.

possibilities of the moving image on a television screen. And computer art is still embryonic.

Computer-generated artwork may in the future continue to exploit exotic devices, such as electrostatic color plotters and laser printers. It is also possible that some type of ink-jet technology can be adapted for artistic uses. In the meantime, the possibilities for the visual artist are fertile and wide ranging. Artists have always valued technical advances in the media at their disposal. The invention of oil and acrylic paints, the discovery of photography have all triggered aesthetic advances in the visual arts. Computers and computing will have a similar impact.

COMPATIBILITY WITH OTHER TECHNIQUES

PRESENTLY, the available techniques for an artist using a computer have only limited compatibility with traditional artistic media. The computer and associated peripheral devices that are commonly available cannot make an oil painting. We can approximate the qualities of a painting using a high-resolution color display or a multiple-color printer or plotter. There are photographic techniques, such as the 3M Company Scanamural process, that will produce a large painting from any type of artwork. Plotters produce drawings on paper that are essentially indistinguishable from traditional handmade drawings. But artists have painstakingly explored the qualities of paint over the centuries, and there is no way that a graphics display system can reproduce all of the physical richness of paint.

Commercial demands of the marketplace determine what type of peripheral devices will be produced. Business needs printers. Computer manufacturers have responded by manufacturing many types of excellent printers. But there exists at least presently a minuscule demand for a machine that can be hooked to a computer to move a brush around a canvas. As artists, we can only hope that someday such machines will exist, and that they will not cost a fortune.

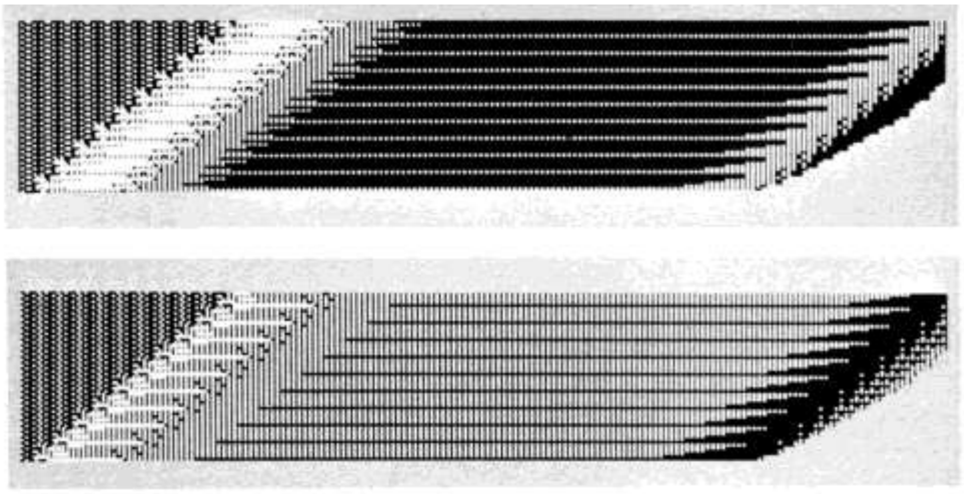
Many artists simply do not know much about computers and computing. Obviously, as artists become acquainted with the unique capacities of the computer for creating pictures, their comfort with these machines will increase. But computing may not hold the answer for all artists—and why should it? All artists are not interested in using photography as an adjunct to picture making, either.

COST AND AVAILABILITY

IN THE ANCIENT DAYS of computing, twenty or thirty years ago, computing was not only very expensive, it was also a very scarce resource. Early computing also suffered from the fact that hardly anyone knew how to make computers do anything. Today, computers are cheap. They are easy to use, and BASIC is simple to learn and universally available in microcomputers. Still, not everyone can afford a computer system. A realistic budget for someone interested in producing low-cost computer graphics or artwork would include a computer in the $500 to $1500 range, such as an Apple II or an IBM PC*jr*. This type of system would include at least a single disk drive, which is absolutely essential. Someone once said, "Life begins with a disk drive." Cassettes have a very large capacity and are very inexpensive, but they are simply too slow.

If the computer does not have an integral screen display and has color graphics capability, then a color monitor or color television set with an RF modulator—to convert the video signal—would be highly desirable. The great stumbling block in any sort of computer graphics system is the question of hard copy. Hard copy is defined as graphic information from the computer that is permanently stored in nonelectronic form. This could be a photograph, a printout, a plot, or anything else that offers a visually accessible record of what the computer did. Printouts are usually thought of as the output of a line printer that has been noisily printed on a long sheet of paper. But printouts could also be made on thermally sensitive paper; they could be produced by an ink-jet printer or made by an electrostatic plotter. Hard-copy devices are very diverse, both in their electronic and mechanical operation and their cost. A minimal base cost for a hard-copy device would probably be about $250, with a realistic median price in the range of $500 to $1000. The minimal expense for a mid-range system for computer graphics would, therefore, probably be in the area of $1000 to $2000. In addition, there will be a certain overhead in the form of consumable supplies, such as paper, pens, and ribbons.

Any sort of discussion about the costs of more expensive systems is interesting but essentially futile. All of the components of such systems—computer, graphics display system, and hard-copy device—increase in cost on a smooth curve that simply has no upper limit. Naturally, prices have fallen dramatically for much equipment and software. But, nonetheless, much computer graphics equipment is beyond the financial reach of most individuals. The future holds much promise, however. The rapid increase in computing power from new microprocessors, and the plummeting cost of memory should result in great improvements in cost-effectiveness of computer graphics devices.



FIGS. 4a and 4b. Terry Blum. *Folded Structures,* 1983 sequential screen photographs. The hardware is a Cromemco Z-2HGS, and the software was written by John Dunn. © 1983 Terry Blum.

Another significant but mostly hidden cost of any computer system is the expense of the training to use it. This cost may be quite large if the user is unfamiliar with computers. The training costs may consist of fees for actually attending classes or they may be in the form of the time that a person spends playing with the machine in order to make it perform some constructive task. Any new peripheral device that you add will always require learning time and perhaps training in its use.

Computer graphics equipment may not seem so expensive when the cost of more conventional art-making equipment is considered. A sculptor, for instance, might easily have many thousands of dollars invested in tools, equipment, and materials. A video artist or photographer would be another good example of one who must invest heavily in technological machinery. Expenditures and investments are required no matter what type of art you make. Often the outlay a computer artist must make will not be more than that required for other types of art making.

IMPROVED PRODUCTIVITY

ARTISTS MAY CRINGE at having such a word as productivity associated with their exalted endeavors, but the plain fact of the matter is that artists make things with their hands. Performing a repetitive task very accurately, over and over and over again, is grist for the computer's mill. Computers can make certain types of pictures much faster than the artist's hand. Not only can these machines make pictures faster, but they reduce or eliminate the drudgery of making the pictures.

Word processing is an extremely useful tool for writers. Word processing does not write books, reports, or letters. The writer writes them. Word processing makes writing easier and faster. Editing becomes much swifter and retyping is eliminated. Making artwork with the computer is similar. The computer can make pictures faster. This has several important ramifications. The artist can produce a larger number of works in a given time than would be possible using conventional techniques. If computers were incapable of anything else, it seems that this fact would justify and recommend their use. But whether the images are simple or complex, the improved productivity of the art-making process means that more possibilities are opened for the artistic imagination. Works that you might never have considered because of their complexity and the potential drudgery of making them can be attempted with the digital computer.

Another important virtue is to minimize the time spent on false starts or dead ends. Suppose you embark on a drawing and, in the process, it becomes apparent that the work is a failure, or you simply lose interest. While the computer cannot eliminate the artistic false start or wrong direction, it can reduce the time committed to such unsuccessful works. Because it enables the artist to explore different avenues without the onerous constraint of large time commitments, the computer really fosters creativity and experimentation.

THEME AND VARIATIONS

COMPUTER IMAGES can be manipulated with a variety of techniques. Making pictures usually involves a series of studies or sketches in which a concept is developed and elaborated upon. The possible variations are often very large. Even when a final design is settled upon, the artist may continue to produce variations on a theme. Josef Albers's series of color experiments entitled *Homage to the Square*, or Frank Stella's themes and variations among his large abstract paintings are notable examples.

After the creation of a model (the theme), the variations or permutations of that model are natural consequences of the mathematical and procedural nature of computer graphics. As an illustration of this ability to produce many possible variations on a single archetype, it might be useful to look at an example slightly removed from the world of fine arts. Donald Knuth, a distinguished computer scientist, wrote a computer program that is capable of designing a multiplicity of type fonts. METAFONT, as the program is called, can create wide and unexpected variations upon the simple theme of our ancient alphabet.

The alphabet, reduced to its most simple geometric elements, could be defined with a few constituents: the circle, the half-circle, a horizontal line, a vertical line, and the two 45° diagonal lines. Over the centuries, these schematic elements have served as the basis for a multitude of variations and alterations. In many cases, the alterations have been made for purely visual reasons—to make the type more legible. In other examples, the development of printing technologies has served as the incentive for change. Often, merely the passage of time and with it the change in fashion, has pressed the evolution of letter forms. Whatever the case, a casual glance at a printer's sample book of fonts will reveal an extraordinary variety of typography. Knuth's METAFONT addresses this proliferation of typefaces with a mathematical analysis of the elements involved in type design. The program can

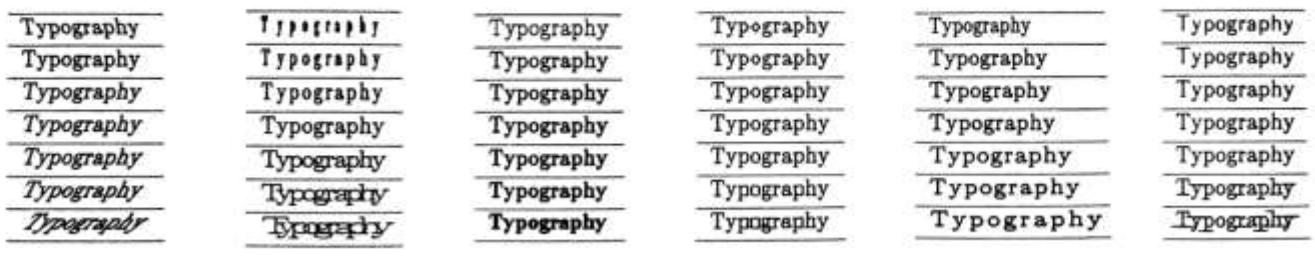


FIG. 5. Donald E. Knuth. 1983 sample output from METAFONT. © 1983 Donald E. Knuth. Image created by Scott Kim, © 1983 Scott Kim.

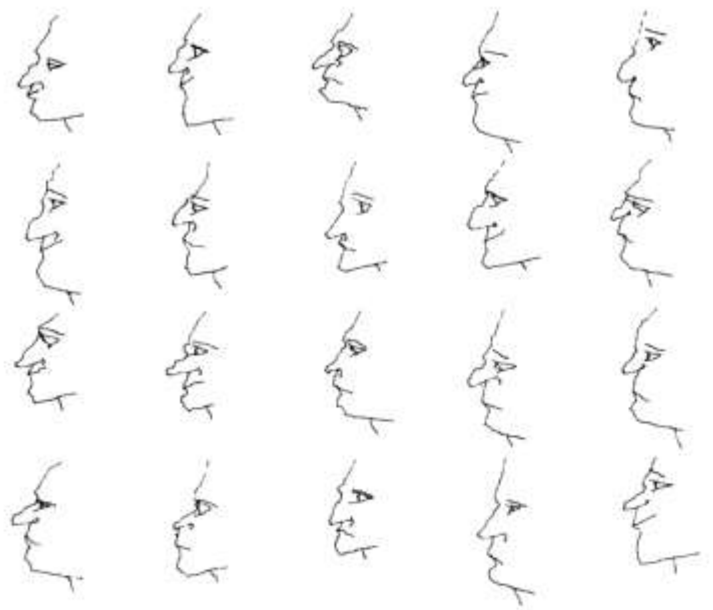


FIG. 6. METAFACE

mimic the great variety of existing fonts and is also capable of generating entirely new and unanticipated designs.

METAFONT creates typefaces. Obviously this process is not the same as making an artwork. But consider the problems and processes that are involved in depicting things. The human face could be schematically rendered with a sparse set of lines and circles similar to the minimal description of the alphabet. It would be possible to write a program—let's call it METAFACE—that would emulate some of the extraordinary variations of the face. The parameters for the various visual descriptions of the face would be given to the program: the size of the eyes, the location of the eyes, and so forth. Depending on the ambitiousness of the programmer, the program could become exceedingly complex. The various dimensions and proportions of something as familiar as the human face are hardly a simple matter. The actual rendering of the face would further complicate things. Do we shade? Do we use thick or thin lines for the features? Even a few variables would produce a vast number of different faces. Needless to say, the biochemical algorithm encoded in DNA, which is the true creator of faces, would make our algorithm look pale indeed.

Just as METAFACE could create an infinite series of faces, consider, if you will, the spectrum of human faces that artists have rendered over the centuries. Cultural and aesthetic pressures have influenced the artist in the stylization of the drawing or painting. The artist is constantly exploring and experimenting with new possibilities of representation. If the METAFACE program were cleverly designed, it might actually be able to mimic various historical styles of art. Perhaps it could also hint at novel styles of art.

It is this capacity of computer hardware and software to quickly and effortlessly produce great variation within some common framework that offers the artist vast power. All artists study the thematic possibilities of their work. Monet's *Rouen Cathedral* paintings, in which he systematically explored the variation of light on the church facade, is a particularly good and relevant instance. While we have been using the human face as an immediate example, the issue of theme and variation could be applied to any subject matter or style.

THE TOOLBOX

COMPUTER GRAPHICS, in the words of Aaron Marcus, is a totally new toolbox for the artist. Artists love tools. They make pictures and images with their tools, so it is natural that they have a great deal of tactile involvement with their implements. Because this tool is very different from the previous means of fabricating art objects, it seems natural that artists will use it in unforeseen ways. It is impossible for the inquisitive artist to avoid being fascinated by the possibilities of this medium. But it is always difficult and often frustrating to learn the ways of a new tool. When the tool is so generalized and can be utilized in so many ways, there is certain to be a period of learning and readjustment.

The artist Ed Oppenheimer uses a computer to study the possibilities of textile designs as well as to set up the loom for his weavings (see Fig. 7). The speed of the computer allows Oppenheimer to spend more of his time studying visual possibilities.

Whether this new tool is used for specialized applications, such as designing textiles, or to augment traditional artistic activities, its potential is very broad. The art-making strategy can be simple or complex. Given the individualistic nature of the twentieth-century artist, computer art will become not one genre, but many.

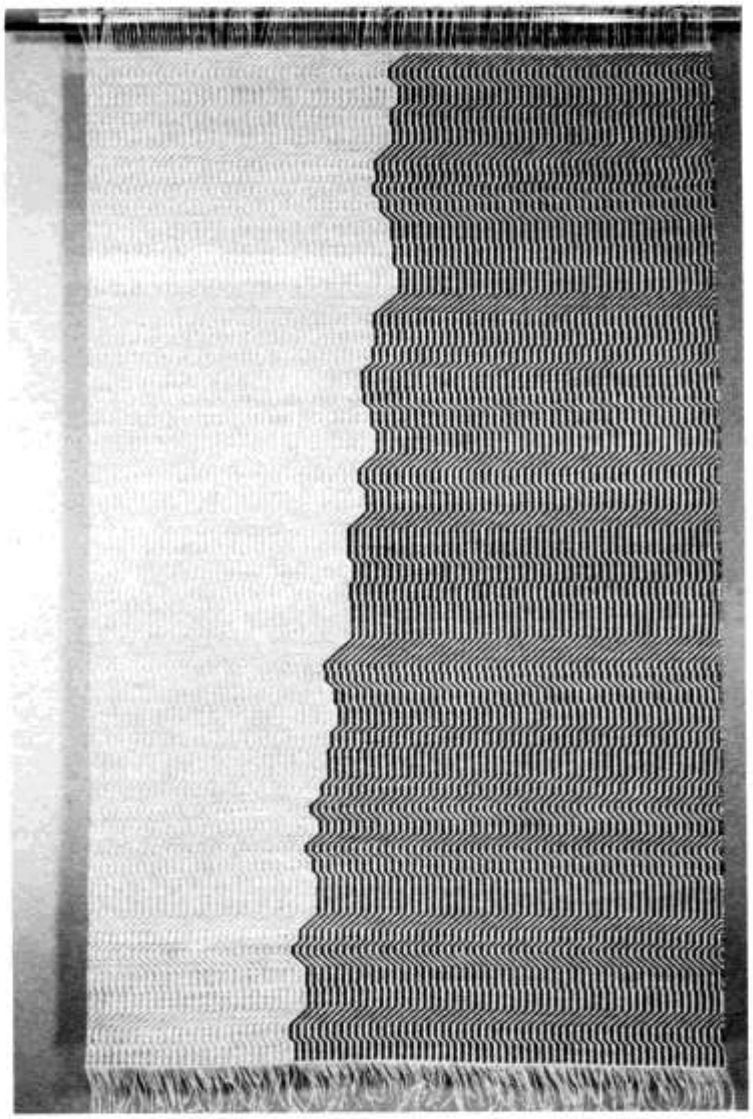
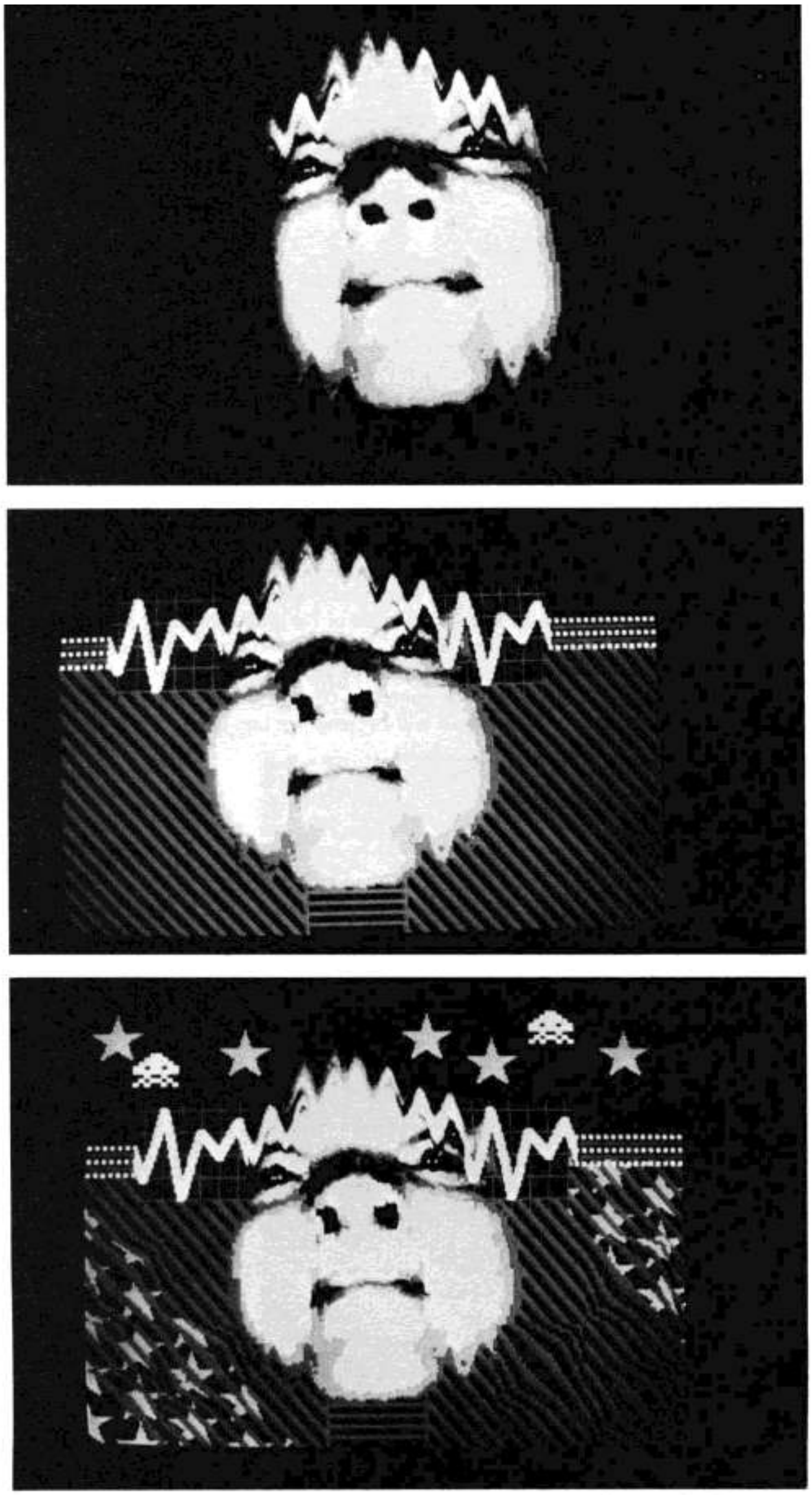
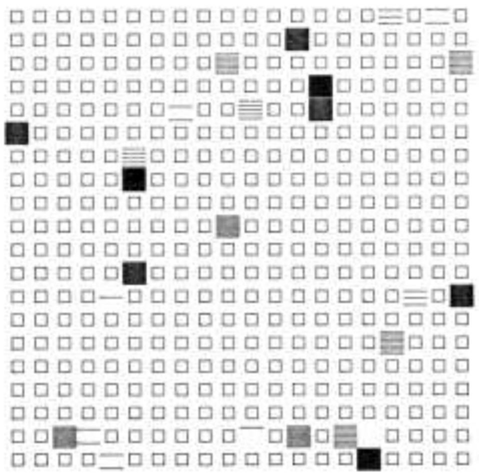


FIG. 7. Ed Oppenheimer. *Geologic,* 1983 weaving, 41″ x 67″. Design created with Apple II. © 1983 Ed Oppenheimer.



FIGS. 1a, 1b, and 1c. Jane Veeder. *Warpitout,* 1982 sequence of screen photographs. Zgrass computer language running on a Datamax UV-1 computer and digitizer. © 1982 Jane Veeder.



2. WHAT MEETS THE EYE: DISPLAYING COMPUTER GRAPHICS

MANY THINGS can be said about the incredible popularity of the ubiquitous video game, but one of the central reasons for the games' success is the appeal of computer graphics. Although computer graphics have been actively used in industry for a decade, the video game literally brought them into the home. This has several ramifications for the artist interested in using the computer. Most important, the video game has driven the market to produce inexpensive machines with graphic capabilities, so excellent graphics are available in many personal computers. Most of these computers also allow the access of these graphics through the resident BASIC language. More software, even staid business programs are utilizing color graphics, simply because they represent a more efficient way to communicate information.

Jane Veeder, a computer and video artist, has taken the concept of the video game and used it as the basis of her artwork. At the 1982 SIGGRAPH computer graphics conference in Boston, her interactive video game, *Warpitout,* was one of the most intriguing artistic events. Viewers, or participants, would first have their faces photographed by a video camera. That image would be digitized, or converted into a series of digital values. Once digitized, the image could be manipulated and transformed with wonderful variety. Areas of the face could be magnified and distorted. Lines and symbols could be added to the image.

Warpitout is a dynamic, real-time artwork. It is a good example of the types of manipulations that can be readily applied to images, using the power of the computer. Veeder's work also demonstrates the kinds of artistic possibilities that an interactive computer work possesses. Instead of an artwork that exists as an entity separate from the viewer, *Warpitout* suggests the possibility of an ongoing, active participation of the viewer. This, of course, is hardly a new notion, but the nature of the computer makes it possible for the viewer to become actively involved in a new way.

THE CREATION OF GRAPHIC INFORMATION

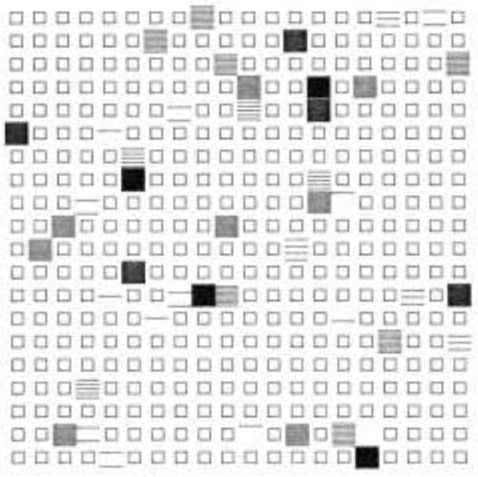
THE COMPUTER has no intrinsic method of creating graphic information. Indeed, even creating alphanumeric information is not necessarily an inherent part of the inner workings of a computer. But because these machines are intended to be utilitarian devices, not digital abstractions, we must be able to obtain information that we can easily interpret. We must be able to apprehend either visually or aurally the information the computer produces. The most obvious example of readily interpretable output is the information displayed on the monitor screen or printed or drawn on a sheet of paper. We will confine our discussions to these two common types of output, which are analogous to the two most common forms of two-dimensional art making—drawing and painting. The video display is like a painting; the printed or plotted output is like a drawing. In the case of a painting, at any given point we can see essentially only one color. Paints are usually, but not always, opaque. When we paint, each new stroke of the brush covers and hides what is underneath. The same is true of video displays. Each given point on the screen can display only one color at one time. Drawings are linear. They are composed of a series of narrow discrete lines. The output of printers and plotters is likewise composed of discrete lines or characters. It is possible to draw over other lines—in fact, it may often be desirable to do so—because, in effect, the lines are transparent. However, the images can become muddy and confused by repeated overlaying.

VIDEO AND COLOR MIXING

THE TELEVISION SET is universally familiar both for broadcast video and as an output device for computers. What is not so familiar is the rather extraordinary technology packed into what has become a mundane electronic object. A stream of electrons is emitted by a hot cathode at the rear of an evacuated glass tube. This cathode-ray tube—or CRT—contains magnetic windings that can deflect the electron beam. The electronic circuitry of the set controls these windings or plates. The electron beam reveals itself to our eyes when it strikes the phosphorescent coating on the face of the CRT.

The light emitted by the phosphors decays after a short period of time. This period of time can be anywhere from a few thousandths of a second to several seconds. This effect is called persistence. In a broadcast television picture, with a constantly moving image, obviously the persistence cannot be long lasting. However, if the persistence is too short lived, the image will appear to flicker. If the persistence is too long, the image will appear to be smeared.

The electron beam creates a spot of light on the screen, but the whole image is created with a raster scan. The raster scan is a zigzag pattern that the electron beam traces across the screen as it moves from top to bottom. The beam moves from right to left and is automatically turned off as it moves back from left to right to begin scanning the next line. The first raster scan traces across the odd lines in 1/60 second. The second scan traces across the even lines in the next 1/60 second. This scanning of alternate lines is called interlacing. Thus, one complete image or frame has been generated on the screen in 1/30 second. Thirty images a second are created on the CRT, but our eyes and brain perceive this rapid succession of pictures as a single, continuous image. This phenomenon is called the persistence of vision. Motion pictures rely on the same effect, except that they commonly use a slightly lower rate of twenty-four frames, or images, a second.



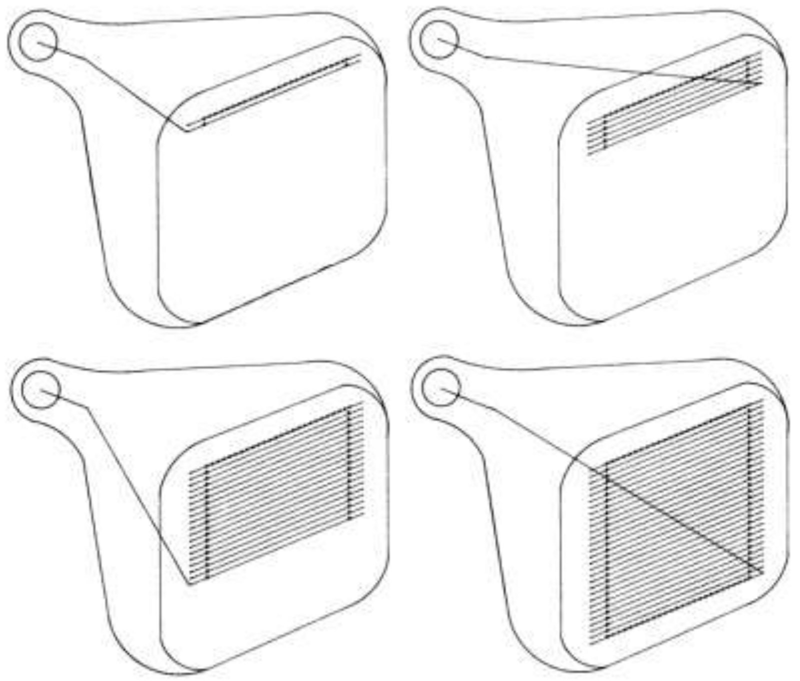


FIG. 2.

The creation of the broad spectrum of colors in the CRT or color television set is similar in principle to the previous example but, obviously, is a more complex matter. The arrangement of the electron gun in the evacuated tube is the same, except that there are three guns—red, green, and blue. Some systems have these guns arranged in a line and some arrange them in a triangle, or delta. These guns project their individual beams toward the screen where they intersect a metal grid, called the shadow mask. The shadow mask is immediately behind the viewing surface of the CRT and consists of a matrix of very small holes or slots. The three electron beams are directed toward these holes. The phosphors on the CRT have been laid down in a pattern to correspond with the holes in the shadow mask. Thus, immediately behind each hole is a triangular pattern of the three phosphor colors—red, green, and blue. The function of the shadow mask is to prevent any stray electrons from reaching adjacent holes in the mask.

It is plain that there is more than meets the eye in this cursory description of the shadow mask CRT. The technology involved in creating an image on the screen of the television set or monitor is complex and requires great precision. The electron guns must be

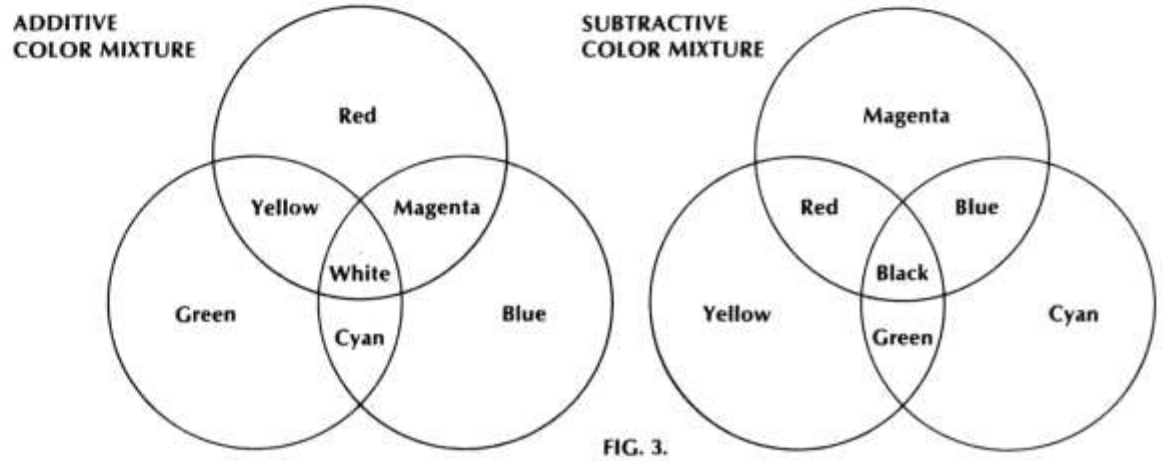


FIG. 3.

very precisely aligned over the entire area of the screen. The degree of alignment is called convergence. While the beams may be highly convergent in the center of the screen, the convergence will most likely decrease toward the periphery of the CRT.

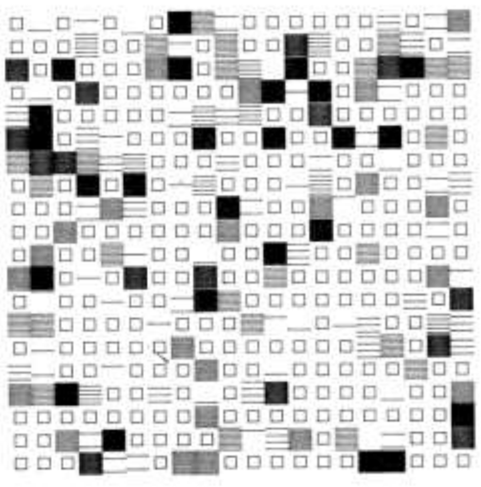
The rainbow variety of colors that light the screen is created by what is known as an additive process. This is the opposite from the subtractive process that creates intermediate hues when we mix paints. If we mix yellow and red paint we obtain orange. Red and blue make purple, and so forth. Theoretically, in the subtractive process the three primary colors when mixed together should produce black. In practice, a muddy brown is usually the result. In the additive color process, the three primary colors of light are red, green, and blue. Combining these three colors together—as in the case when all three electron guns are simultaneously on—will produce white light. Green and red produce yellow; green and blue produce cyan (a bluish tone); and red and blue produce magenta. Naturally, by varying the relative intensity of the electron beams, many variations of hue, value, and chroma are reproducible on the CRT.

The conventional color television set requires a video signal that is transmitted by high-frequency radio. The video output of most lower-cost personal computers produces a signal that is compatible with this broadcast standard, which is defined by the National Television Standards Committee, or NTSC. When this NTSC signal is broadcast, it must be modulated to the appropriate radio frequency. When a microcomputer outputs this NTSC signal, an ordinary television set must first change the computer's signal into the equivalent of a broadcast signal. In other words, the NTSC signal must look like a broadcast signal. Usually this change is performed with an RF modulator.

The NTSC video signal is also commonly called composite video. Some types of monitors, called composite monitors, can only utilize this NTSC signal. These monitors produce a somewhat better quality image than a conventional television set. The best type of video image is produced with RGB (red, green, blue) video signals. These signals are generated directly by the computer. They drive each electron gun individually. Because the computer directly drives the monitor, images that have much more detail and a much greater variety of colors can be produced.

PHOTOGRAPHING THE VIDEO SCREEN

IT IS OFTEN very useful to be able to record the image on the screen. There is a variety of methods that can be used, but the most direct is simply to photograph the screen with a 35-mm camera and color slide film. Because the screen image is refreshed every 1/30 second, it is important to use a slow shutter speed so that at least one full screen frame will



have been displayed while the shutter is open. At any shutter speed greater than 1/30 second only a portion of the image will be captured. Thus, about 1/15 second is a desirable speed. For static images, even slower speeds can be used. It is also helpful to photograph the screen in a darkened room to eliminate extraneous reflections from the screen. Because most monitor screens are relatively small, the standard 50-mm focal-length lens of a 35-mm camera will produce distortion at the close distances necessary to photograph the screen. A slightly longer focal-length lens, say 100 mm to 135 mm, will eliminate any of this "pincushion" distortion.

Technically sophisticated camera systems are used for RGB color monitors. They photograph each of the three color signals individually to make color images on the film. In other words they first photograph the image created with the red gun, then reexpose the film to the green image, and finally make a third exposure of the blue image. Such systems produce superlative results but are also expensive. Some units cost in the range of $1000, but most cost considerably more. For the casual user, direct photography with a good 35-mm camera will usually produce more than satisfactory results.

Video cassette recorders are also excellent for recording the dynamic screen output from the microcomputer. Since the video cassette recorder is designed for use with broadcast video, the computer must provide an NTSC video signal. Recording images from the computer is easy. Plug the video output into the VCR. If a separate audio output from the computer exists, it can also be recorded by the VCR. Recording the output from an RGB video output requires specialized equipment.

RESOLUTION

RESOLUTION is a very important definition of image quality in computer graphics. Very simply, resolution is a measure of the number of picture elements relative to the overall image size. Most commonly, the term is applied to the amount of detail seen on the screen of a video monitor. However, because we are dealing with digital machines, it is equally applicable to any type of graphics output device, whether that is a video monitor, printer, or plotter.

To say that the video resolution of a computer system was 320 × 200 would mean that the display would have 320 pixels along the horizontal, or *x*, axis and 200 pixels along the vertical, or *y*, axis. Usually, but not always, the horizontal axis dimension is given first, followed by the vertical axis. Such a 320 × 200 display would contain 64,000 individual pixels. The terms "high," "medium," and "low" resolution are invariably encountered in any sort of descriptive literature about computers. They are highly subjective terms that must be interpreted carefully. While 320 × 200 resolution in a low-cost personal computer might be described as high resolution, the same display would probably be considered low resolution in an industrial or scientific computer graphics system.

Resolution is an important measure of how much detail, and how much fidelity to a

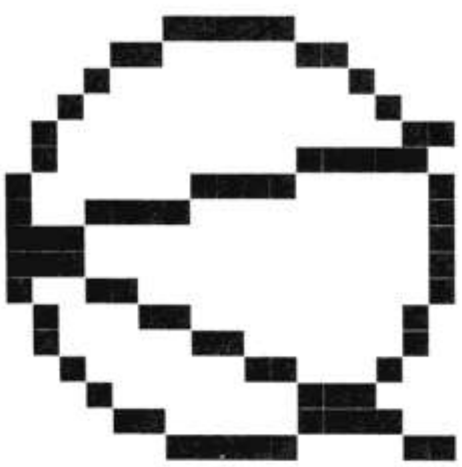


FIG. 4a.

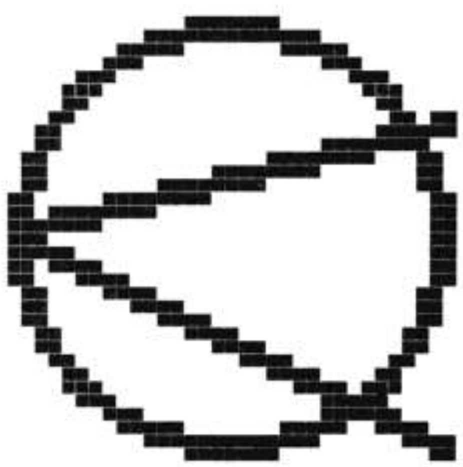


FIG. 4b.

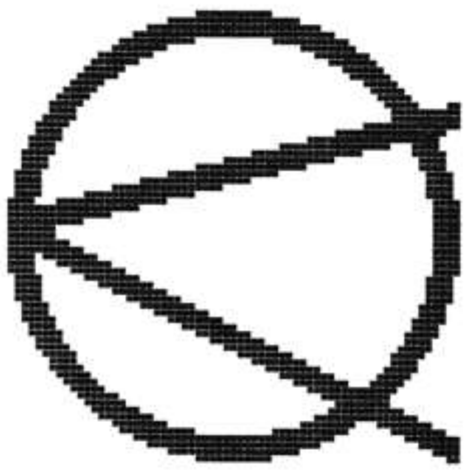


FIG. 4c.

depicted object, a computer-generated image will have. Almost all computer graphics output devices rely on a rectangular grid of points. Within this matrix of points, individual points can be addressed by the display device. This grid is fine for dealing with horizontal and vertical elements. But in fact, diagonal, circular, and irregular lines and shapes dominate the reality of our visual world. To depict these elements in a computer display, they must be approximated with a series of horizontal and vertical picture elements.

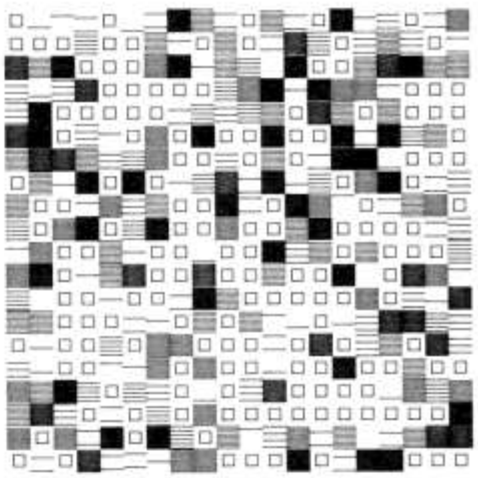
To demonstrate this effect, Fig. 4 shows three illustrations of an identical figure composed of circles and diagonal lines. At the lowest resolution, we have only a coarse approximation of a circle, and, although the fidelity at the highest resolution is much improved, the image still remains an approximation. Resolution is an important factor in the transmission of visual information. The letter formed by a crude dot matrix printer is read more slowly and with greater difficulty than the carefully formed letter that a letter quality printer or typesetting would produce. High resolution is a necessity for rapid visual communication; while desirable for artistic purposes, it is not a necessity.

Some machines can only display alphanumeric characters. The American Standard Code for Information Interchange (ASCII) defines a set of numbers, punctuation marks, a space character, assorted control characters, and the upper- and lower-case alphabet. This 128 ASCII character set is universally available on all personal computers. There are no special graphics characters in this set. Some microcomputers have special graphics characters in their character set. The Radio Shack Model I is an example. The Model I graphics simply consist of solid block characters. Other manufacturers have used the strategy of having characters that are definable by the user. Within the 8 × 8 pixel grid of the Texas Instruments 99/4a computer, it is possible to specify whether any of the 64 pixels will be either on or off. Computers with user-definable characters are more desirable than those having only the alphanumeric ASCII but they are still limited in capabilities. For instance, it is very difficult to use them to draw a line from one point to another.

Because the widespread use of color television sets as displays for personal computers, the effective resolution for characters is about 40 characters horizontally and 24 or 25 characters vertically. This is about the maximum usable resolution obtainable with NTSC video. Computers designed for business or word processing will usually have an 80 × 24 or 25 display. Even wider displays, up to 132 columns, are becoming common.

BIT-MAPPED DISPLAYS

MOST USEFUL to the artist, or for any type of computer graphics application, is the bit-mapped display. This formidable phrase means that individual pixels in a video display



can be turned either on or off. This type of graphics mode is also known as all-points-addressable graphics. In the case of a monochrome display, each of the pixels can be either black (turned off) or white (turned on). The actual color of the monochrome display may be amber or green instead of white. In the example of a 320 × 200 monochrome display, each pixel would be represented by one bit of memory. The memory in each case would be either 1 or 0 corresponding to on or off.

A color display requires more memory. By using two bits of memory for each pixel location, four colors can be defined.

00 = first color
01 = second color
10 = third color
11 = fourth color

In the example of a 320 × 200 display with four colors for each pixel, we arrive at the following calculation: 320 × 200 × 2 = 128,000. For these 128,000 memory locations we can then arrive at the number of bytes of memory required. For a machine that uses eight-bit words, the memory required for one four-color display would be 16,000 bytes: 128,000/8 = 16,000. This segment, or section, of memory is called the frame buffer or frame store. Because of the geometric nature of the screen and the number of colors to be displayed, high resolution and multiple colors require extensive use of memory. For example, a 640 × 400 screen resolution with sixteen possible colors for each pixel (four bits for 2^4 colors) would require—640 × 400 × 4 = 1,024,000—more than a million memory locations. Using eight-bit memory would mean 128,000 bytes for such a display. Very high resolution systems that feature a large number of possible colors not only use massive amounts of memory but they must also continuously transfer this information between the computer and the display thirty times a second. These very high data transmission rates require specialized processing. Because of the memory required and the problems associated with the very high speeds required by the data transmission, very high resolution graphics systems are still quite expensive. Fortunately, the cost of memory has dropped radically, and the computing power necessary to control these graphics displays has increased in availability. We can expect the cost to continue to fall for computer graphics displays.

To increase the number of possible colors in a display, a common device called a color lookup table (see Fig. 5 on p. 28) is frequently employed. A color lookup table represents a preset index of analog color values that will be sent to the electron guns in the monitor. While the graphics display may have, for instance, only sixteen colors that can be displayed at a given moment, those sixteen colors may in fact be able to be selected from a much larger palette.

This device has been used to produce cost-effective displays. But with the decreasing cost of memory it is probable that more devices will use a direct definition system, where each possible color is represented in memory without the use of a lookup table.

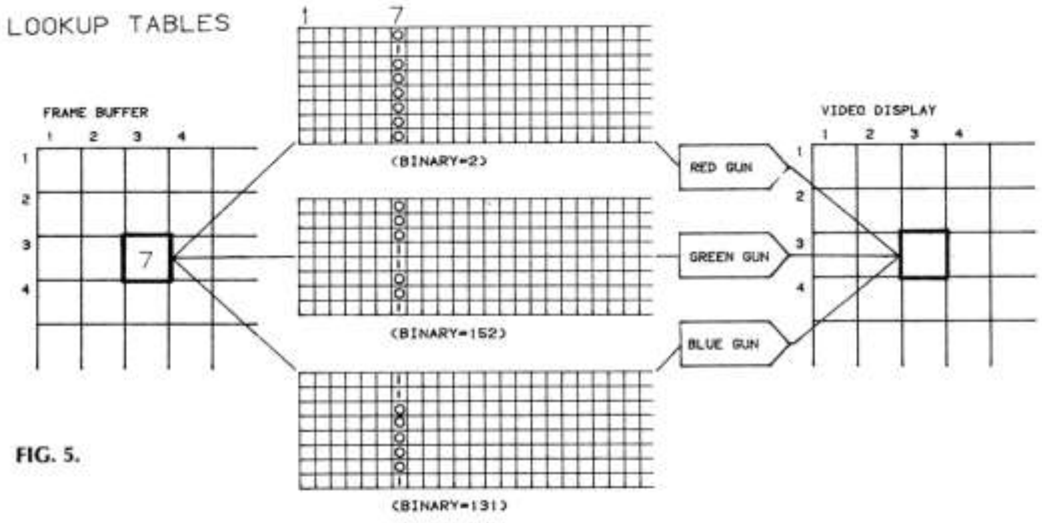


FIG. 5.

THE FUTURE OF THE SHADOW MASK CRT

THE SHADOW MASK CRT is a remarkable technological device. It has clearly gained dominance over all other display devices. Yet it has important technical limitations. While 1024 × 1024 CRTs are common, the upper range of resolution is limited. Presently it is less than the 2000 × 2000 resolution of a conventional photograph. Color monitors with resolution exceeding 1024 × 1024 are very expensive; often their cost exceeds that of an entire microcomputer system. Progress continues on the development of the CRT, and, for the time being, there are no immediate prospects that it will lose its dominance.

However, several other types of display devices deserve mention. Before the raster scan displays became widespread, the calligraphic or vector displays were common. These machines are similar in principal to the raster scan displays, but instead of drawing the entire image with the raster pattern, the electron beam would be moved directly from one point to another. Instead of retracing the linear pattern, a supply of electrons maintains the phosphor glow on the surface of the tube. For many years, such direct-view storage tubes, as they are known, were the mainstay of computer graphics. They are still used in some industrial settings, but are not used in any personal computer products.

Presently, there appear to be no potentially dramatic breakthroughs in display technology. Two areas of technological development have the possibility of producing interesting devices in the future. Plasma displays and liquid crystal technology have limited current application but will undoubtedly become more important. Plasma displays are flat, with an embedded grid of fine wires. The intersections of the wires, when activated, cause a gas to glow. In addition to the advantage of being flat, plasma displays are capable of very high resolution. Currently, only monochrome plasma displays have been manufactured. Liquid crystal displays have been very widely used in wristwatches, but they too are monochrome. They are also flat and compact but have relatively poor contrast and can be difficult to see in certain lighting situations. Color liquid crystal displays have been produced, and it is to be expected that there will be future developments in this area.

PRINTERS

AFTER THE MONITOR, the most common device associated with computer output is the printer. Printers are available in a bewildering variety of types. Their great utility arises from the fact that they create a permanent copy of computer-generated information, whether it be an address label, a letter, program listing, or graphic output.

Printers range from those that are relatively inexpensive to ones that are more expensive than a complete microcomputer system. Although they are not necessarily the most desirable type of machine with which to display artwork, their universality offers a natural and immediate possibility for artistic use.

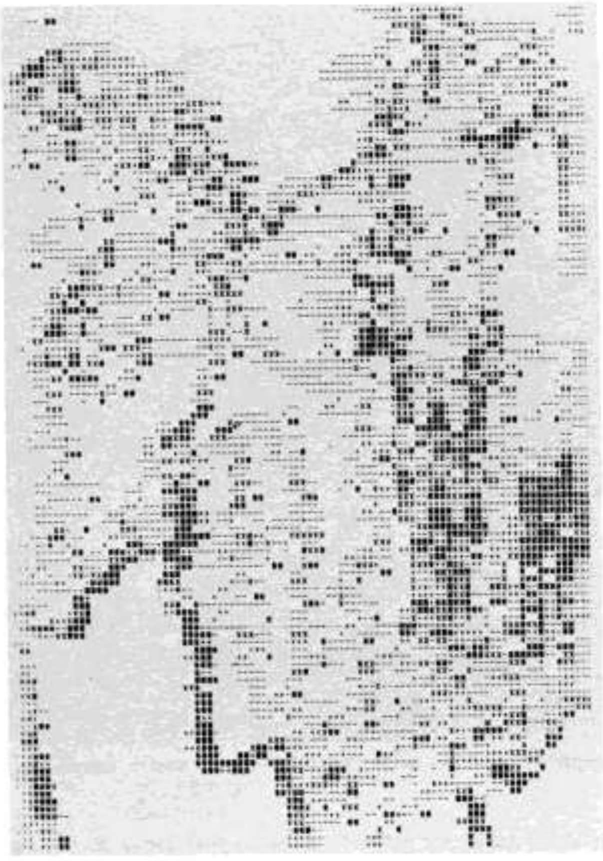
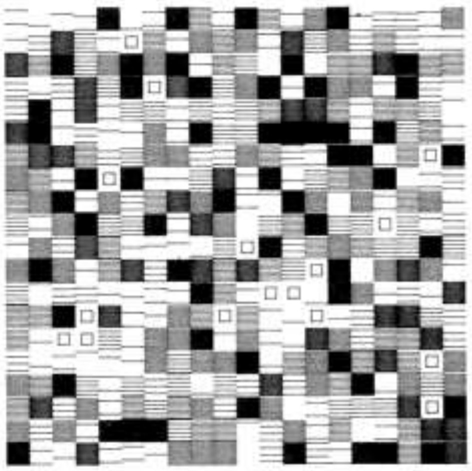



FIG. 6. Waldemar Cordeiro. *Derivate from an Image, Transformation No. 2,* 1971, line printer. © 1971 Waldemar Cordeiro.

IMPACT PRINTERS

The two major types of printers are the dot matrix and the letter quality printer (LQP). The dot matrix printer forms characters on the paper by using a graphic equivalent of the screen generation of characters. The alphanumeric characters are composed of small dots—analogous to pixels on the screen. The print head of the printer contains a series of small wires or needles that are actuated by electric solenoids. The wires are struck by the solenoid, and in turn strike a carbon ribbon, which makes a small dot on the paper. The print head moves laterally across the carriage to create a line of printed characters. The letter quality printer operates on a similar mechanical principle except that a plastic or metal print element with completely and precisely formed characters is struck by the solenoid hammer. The printing elements are arranged in a variety of configurations: daisy wheels, thimbles, and balls. In most cases, letter quality printing elements will only contain the standard ASCII character set with no special graphics characters. Letter quality printers, despite the superior printing quality, have generally not been used for artistic endeavors. Because both the dot matrix and LQP printers rely on the mechanical impact of a printing element, they are known as impact printers.

Dot matrix printers are widely used for graphics purposes. Because individual dots comprise the image, with the appropriate firmware—software stored in read-only memory (ROM) in the printer—it is possible to print individual dots. In such dot matrix printers with graphics capability, the effective resolution on the paper may be fairly high. Depending on the accuracy of the printer, the resolution may range from 50 dots per inch to over 150 dots per inch. As with video displays, this ability to print individual dots is called bit-mapped or dot-addressable graphics.

NONIMPACT PRINTERS

Impact printers operate on the same general principle as ordinary typewriters. Because of the speed and special technological demands on a computer-driven printer, new

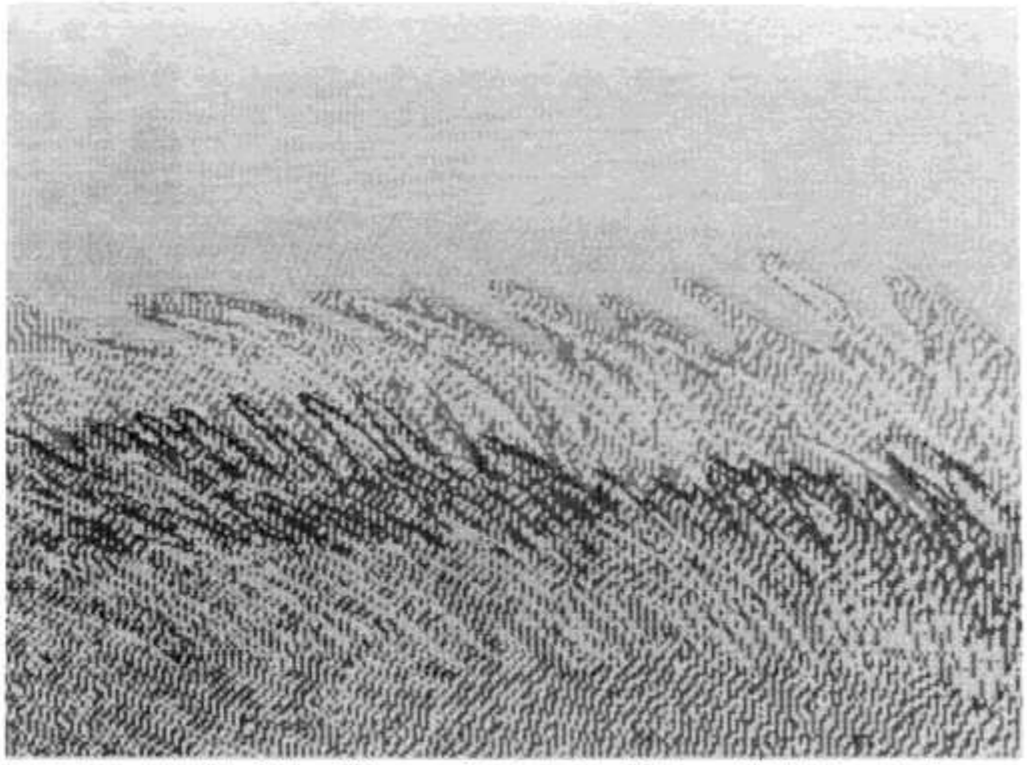


FIG. 7. Richard Helmick. *Glades*, 1983 screenprint. Apple II+ with Epson MX-80 dot matrix printer. © 1983 Richard Helmick.

printing technologies have developed. Many of these technologies do not use a mechanical impact to produce an image. Among these nonimpact technologies are laser, electrostatic, thermal, and ink-jet printing. Some of these techniques are still in their infancy and, in many cases, are very expensive. But for low-cost graphics applications, thermal and particularly ink-jet printers are of interest.

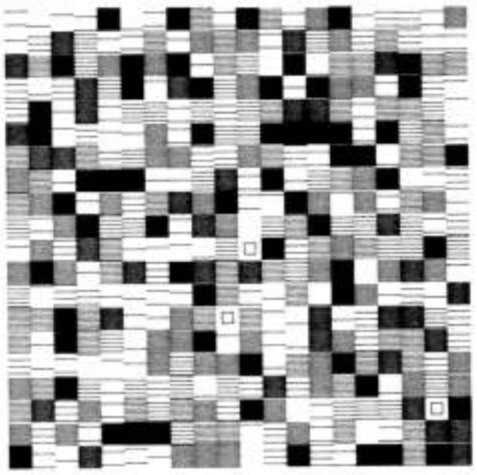
Thermal printers are very common in personal computer systems because of their low cost and low noise levels. They operate by selectively heating small semiconductor elements that are arrayed across the carriage. A stepper motor moves heat-sensitive paper across these elements. After several steps of the motor, a dot matrix pattern is formed on the current line of printing. Some thermal printers have graphics capabilities. The main disadvantage for graphics applications is the fact that most thermal paper is unstable. After several years of even very careful storage, the paper will tend to discolor and the image will fade.

Ink-jet printers operate on the principle of shooting very small droplets of ink at the paper surface. The print head does not touch the paper surface. The original ink-jet technology used a continuous stream of ink droplets that were electrostatically directed. The ink droplets that were not used for printing were recycled. A more recent and lower-cost technology uses a single drop of ink that is shot at the paper when commanded by the printer. This is known as drop-on-demand ink-jet printing.

The ink droplets correspond to the individual dots that the dot matrix printer uses. Thus the final image on paper, whether it be an alphanumeric character or some type of graphic image, is formed in the same manner as by the dot matrix printer. The ink drops are propelled from the print head with several different types of mechanisms. Piezoelectric impulses are used to fire ink droplets in some systems, while others use miniature pumps.

COLOR PRINTING

Most of the new ink-jet printers have multiple-color capacity. Some systems use as few as three colors in the print head while others may use eight or more individual ink colors. The inks are translucent and dry quickly. This means that color images of great complexity can be generated on paper. Color mixtures are possible by combining and overlaying the dot patterns of several different colors. For instance, a green area might be



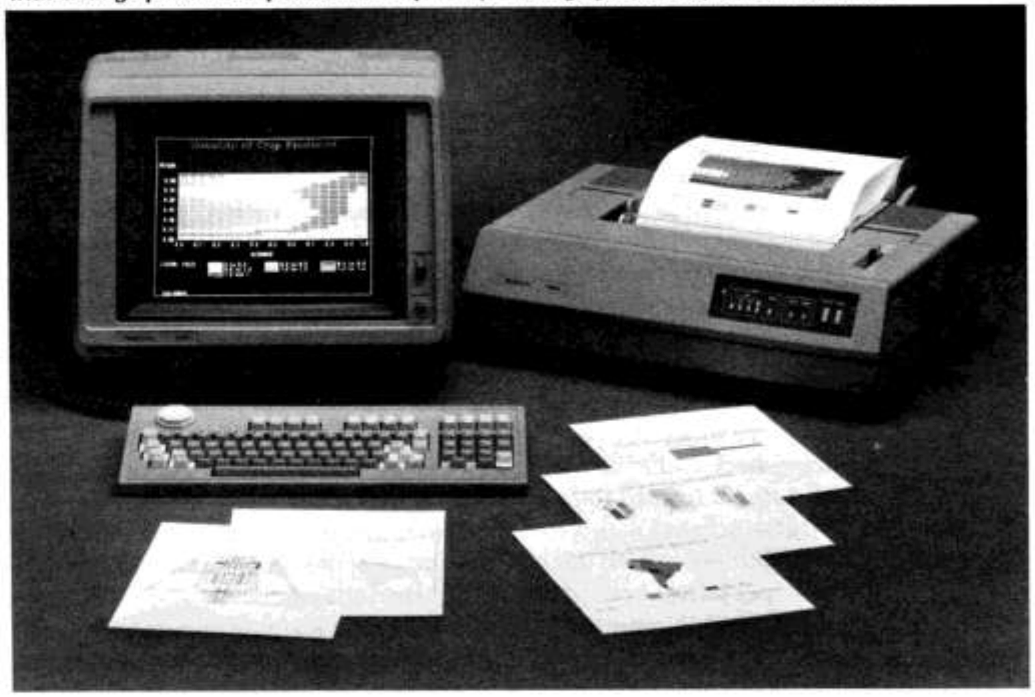
represented by combining a series of yellow and blue dots. This process is calling dithering. Not only is it used in ink-jet printing but it is also frequently used in high-resolution displays to create a wide variety of different colors from a limited range of available colors.

Because of color printing and the bit-mapped graphics of many ink-jet printers, these machines may have potential for creating artistic graphics. Conventional paper can be used in these machines, although some of the manufacturers recommend using specially coated paper. The ink-jet printers also have the virtues of being roughly comparable in cost to impact dot matrix printers and of being quiet in operation.

Color printing has also become available in some of the impact dot matrix printers. These printers utilize the same type of print head as a monochrome dot matrix printer, but use a multicolored ribbon that can be struck with multiple passes of the print head. Dithering is also used with impact color printers to produce additional colors.

A potential problem with both ink-jet and impact color printers is that the colors may not be permanent and lightfast. Often the special technological requirements of creating, say, an ink that can be used in a piezo-ink-jet gun may not also be compatible with the standards of lightfastness. The criteria of permanence for commercial purposes

FIG. 8. Ink-jet printer with 120 dots-per-inch resolution and a 480 x 360 color graphics terminal, with both graphics and alphanumeric capability. Photograph courtesy of Tektronix, Inc.



may be quite different from the criteria of the artist. Unfortunately, the artist may have to rely on his or her own test of lightfastness when it comes to many of the exotic materials used in computers. A very easy and practical test is simply to expose a small swatch of whatever type of material is in question in a south-facing location. Place or tape the swatch to the inside of a window. Keep an identical swatch of the material in a dark place. After a few months, compare the two pieces. If the exposed sample has faded substantially, the color is probably not suitably lightfast. This can hardly be said to be a scientific test, but it is certainly easy enough to make, and will often reveal relative impermanence.

PLOTTERS

THE MOST VENERABLE DEVICE that has been used with computers to produce graphics output is the plotter. Plotters have been used for over twenty-five years. They are essentially automatic drawing machines that are controlled by a computer. The drawing surface of the plotter is divided into series of small divisions. The plotter can move to these locations along the horizontal and vertical axes. Typically, these divisions or units may be 200 or more steps per inch. Thus, an 11″ × 17″ plotter with an actual drawing surface of 10″ × 15″ would have a resolution of 2000 units by 3000 units. The pen carriage of the plotter is physically capable of moving to each one of these points. Upon command the pen, or pens, can be moved up and down. When the pen is lowered to the surface and moved, it leaves a trace on the paper. Like the other types of graphics output devices we have discussed, the plotter pen is incapable of making a true curved or diagonal line. The plotter must approximate such lines with an appropriate series of horizontal and vertical steps. If the plotter is sufficiently accurate, these steps—or stair steps—cannot be perceived.

Plotters are differentiated by the manner in which paper or other media are held by the machine. Flatbed plotters hold a sheet of paper upon a horizontal bed. The paper is not moved. The pen carriage of the flatbed plotter is moved to various locations on the paper. Generally speaking, flatbed plotters are more accurate than other types. They also have the advantage of allowing the user to view the entire plot at one time. But because of mechanical considerations, they tend to be the most expensive.

Drum plotters have a drum—or cylinder—on which is affixed the drawing medium. The pen carriage moves back and forth along one axis, usually the *y* axis, and the paper moves back and forth along the other axis. A recent variant of the drum plotter does not use a drum. Instead the paper is gripped between two rollers. One of the rollers—the grit wheel—is coated with a rough material. The other roller is smooth. When the paper passes through these rollers, the grit produces a series of very small impressions in the paper. As the paper passes back and forth between the grit wheel and roller, it settles into the previously formed impressions, thus maintaining accurate registration of the paper. The manufacturers of these types of plotters claim that this design permits the use of smaller and less expensive motors and control electronics.

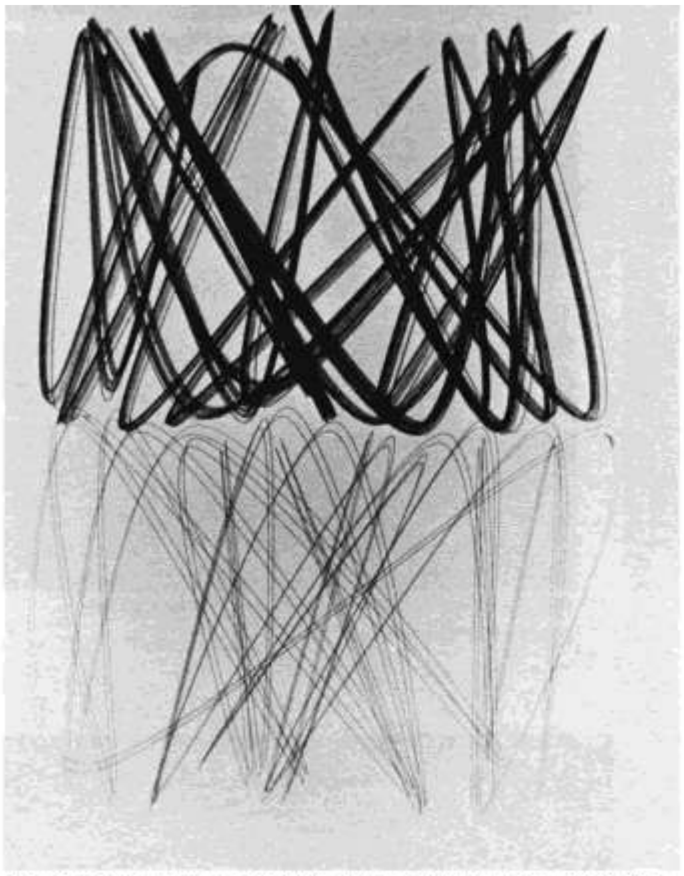
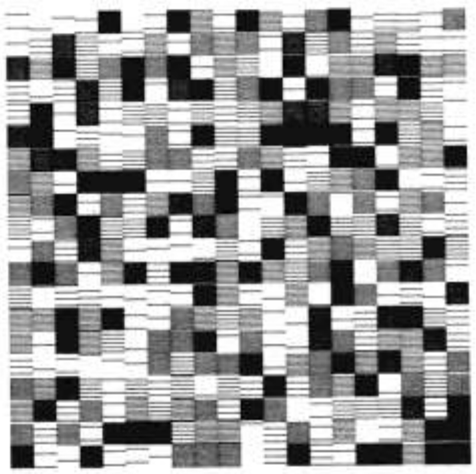


FIG. 9. Robert Mallary. 1983 two-color plotter drawing. SCRIBBLE software running on a CDC CYBER, with a four-pen CalComp drum plotter. © 1983 Robert Mallary.



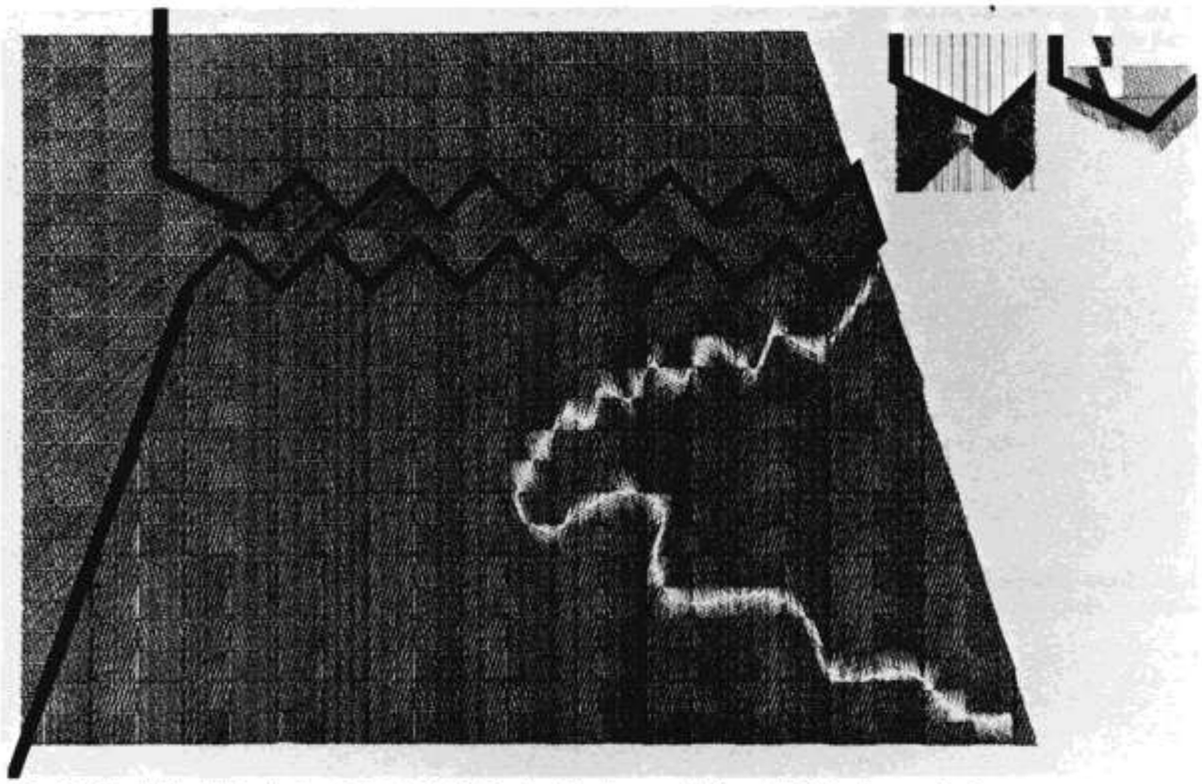


FIG. 10. Gerald Hushlak. *Mechanical Aesthetic for the Workers, or, A Record of Development of Surface,* 1982 plotter drawing on paper. © 1982 Gerald Hushlak.

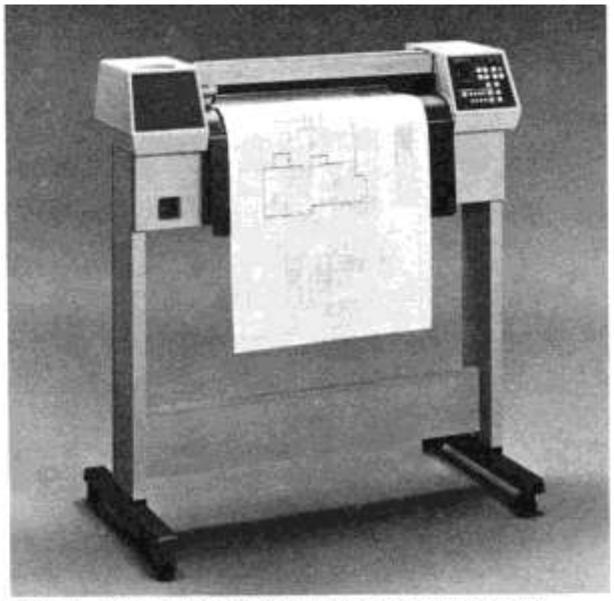


FIG. 11. Drum-type plotter, using a grit wheel paper transport. This high-speed, high-resolution plotter can accommodate paper 24 inches in width. Photograph courtesy of Hewlett-Packard Co.

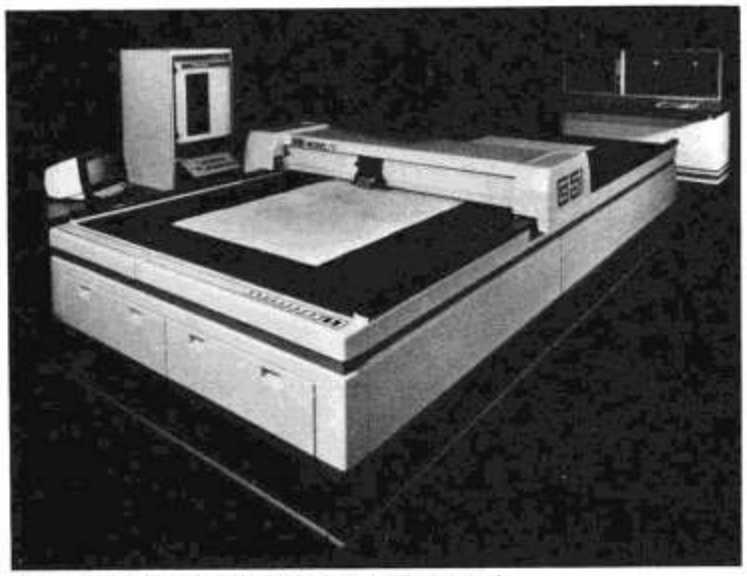


FIG. 12. A 60" x 120" flatbed plotter. Photograph courtesy of Gerber Scientific, Inc.

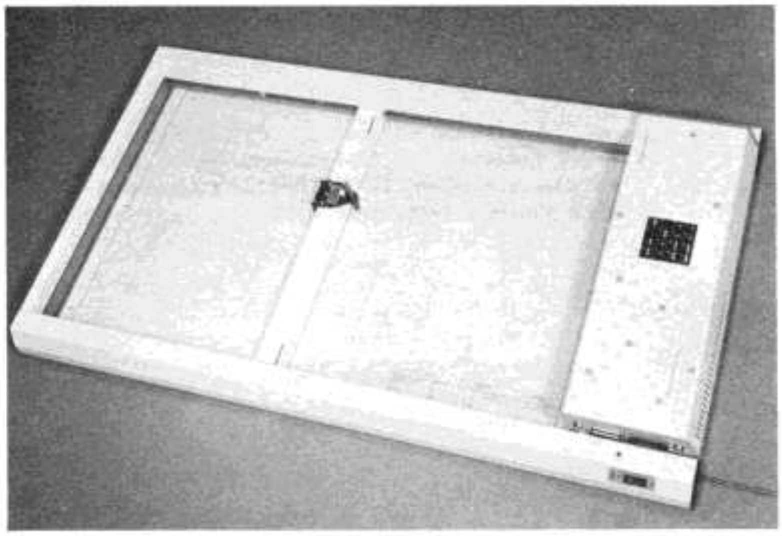


FIG. 13. An open flatbed plotter that can be freely moved about the drawing surface. Photograph courtesy of Alpha Merics Corp.

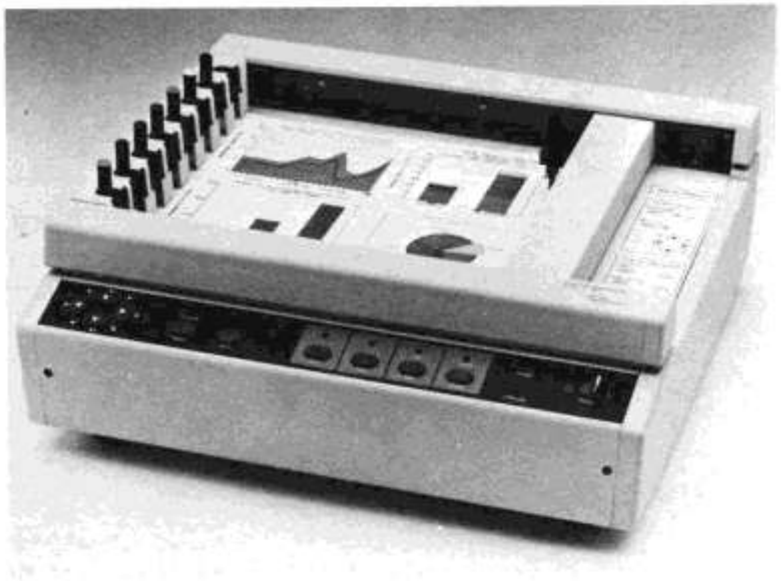
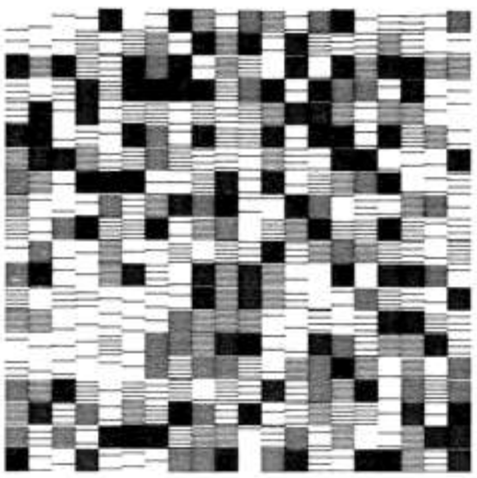


FIG. 14. Small multipen flatbed plotter. Photograph courtesy of IBM Instruments, Inc.



There are a variety of other types of plotters, but they differ in the means by which a pen trace is left on the paper. Electrostatic plotters operate mechanically in much the same fashion as a drum plotter but use the electrostatic deposition of a pigment to produce the image. Most electrostatic plotters are large and expensive. They are generally monochromatic, although recent high-resolution four-color models have been introduced. Photoplotters plot with a light head that exposes photographic film. Obviously, photoplotters operate in a dark environment. Large flatbed plotters have been used in the garment industry to cut pieces of garments. Multiple layers of fabric are placed on the bed of the plotter. Lasers or high-speed reciprocating miniature saws cut through the fabric. While some of these plotter applications are beyond our present interests, they illustrate the versatility of the plotter.

PLOTTER OPERATION

A small inexpensive flatbed plotter is controlled by two stepper motors. Stepper motors are designed especially for use in computer applications. A digital pulse is sent to the motor. The windings of the motor are so constructed that the motor shaft turns a precisely determined amount. These discrete steps—or indexing—that the motor makes are controlled by the inherent design of the windings. Thus, very accurate rotational motion is produced.

The stepper motors are connected to a pulley and cable system that moves the pen carriage back and forth along the horizontal and vertical axes. The cabling along each axis forms a continuous loop, so that, as the motor moves, the pen will be pulled in the proper direction. The motion of the pen itself—up or down—is controlled by some type of solenoid coil.

Because of the digital signals sent to the stepper motor, theoretically, the pen should always be able to move very precisely to any given position. In practice, of course, the mechanical aspects of the machine will limit the true accuracy. Imperfection of mechanical components, as well as wear and tear, will decrease the accuracy of the plotter.

The motion of these mechanical components, often in simultaneously horizontal and vertical directions, produces vibrations. When the pen carriage moves at certain angles, or vectors, these vibrations may become fairly large. If the plotter is not robustly constructed, the vibrations will degrade line quality. More expensive plotters attempt to reduce these problems by using stoutly constructed components. In some cases, vibration-absorbing devices, called viscous dampers, are installed on motors.

The artist may well ask at this point why such mechanical precision is necessary. Such precision may be desirable for making engineering drawings, but it may not seem important in making artistic drawings. But consider the example of drawing a square or a circle. The pen makes an excursion and then should return to the point where it started—to close the circle or square. The pen carriage may make hundreds or thousands of steps as it moves. While the digital commands to the stepper motors should move the pen carriage back to its starting position, small incremental mechanical errors caused by the motors and

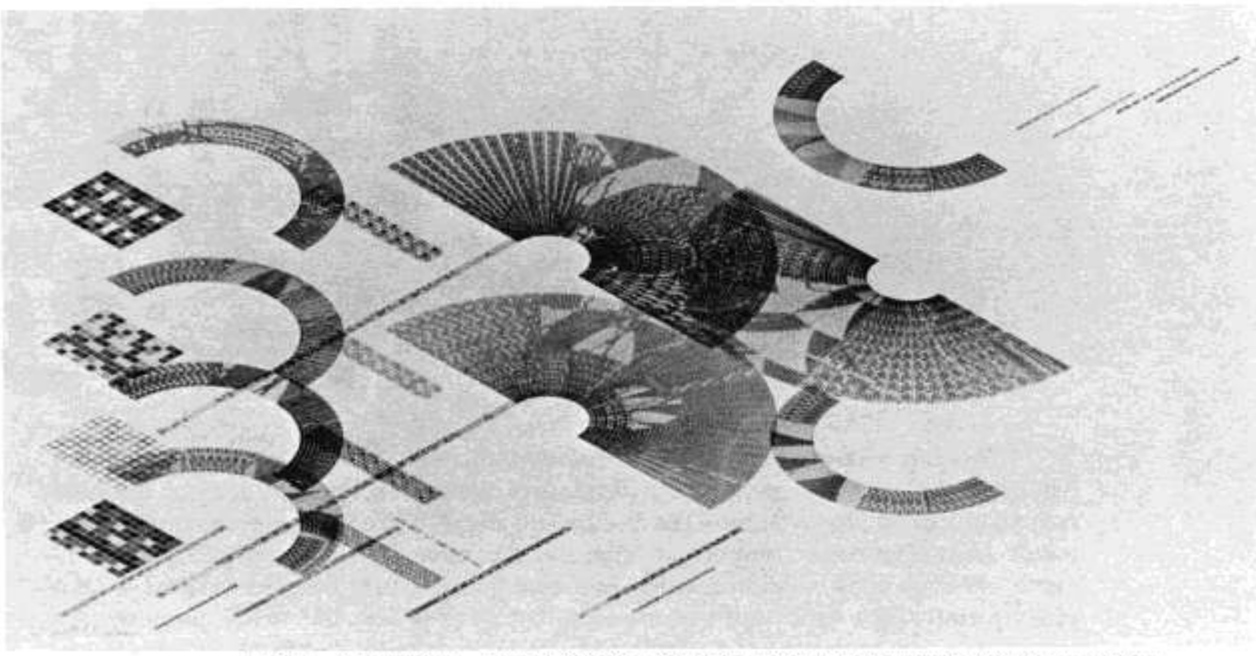


FIG. 15. Mark Wilson. *Skew BB12,* 1984 plotter drawing on paper, 20" x 38". IBM PC and a Tektronix 4663 plotter. © 1984 Mark Wilson.

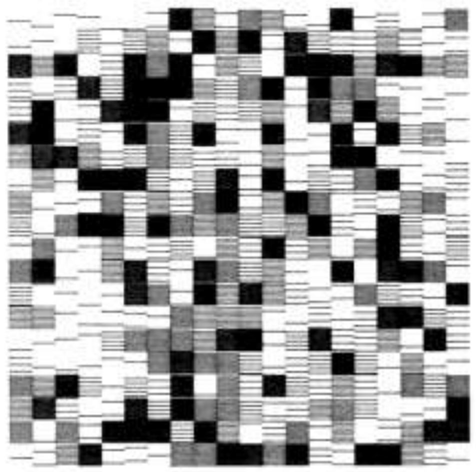
pulley system may cause the pen to end up in a position that is not coincident with its starting position. The measure of how accurately the pen can repeat an earlier position after making a series of moves is called repeatability.

The resolution of a plotter is also an important measure of its accuracy and precision. The resolution means how small a step the pen can make. Like the electron beam in a video monitor or the solenoid hammer in a dot matrix printer, the pen is capable of moving to a discrete number of points on the plotter surface. In our earlier example of a plotter with a resolution of 200 steps per inch and a plotting surface of 2000 by 3000, the pen would be capable of moving to any one of 6 million discrete points. Sometimes resolution is further defined by the concept of addressable and mechanical resolution. Addressable resolution means the limits to which the pen can be moved under software control, while mechanical resolution means the actual resolution of which the motors are capable. In other words, it may take ten digital pulses to cause the stepper motor to move one step. Thus, the plotter was commanded to move one step or unit, and in order to accomplish that one step the motor actually had to move ten smaller steps. In this case, the mechanical resolution is superior to the addressable resolution. Naturally, this is a desirable feature, but it also makes the machine more expensive.

Speed is also an important factor in measuring the quality of a plotter. Two inches per second is about the minimum, while some plotters are capable of moving at over 20 inches per second. Bear in mind that even 20 inches per second is only a little more than 1 mile per hour. But a complex plot may contain literally miles of lines, and when you add the time spent moving with the pen up to a new position, such a plot may take many hours to draw. While high pen speeds are generally desirable, many types of pens are not capable of feeding ink to the pen point at such speeds. For instance, a technical pen may not be able to produce a good quality line at much more than 10 inches per second.

PLOTTER FIRMWARE, PENS, INKS

The typical small plotter contains a microprocessor with ROM and RAM memory. This small microcomputer within the plotter controls the communication with the host—the computer that is controlling the plotter—as well as providing all the necessary control signals to drive the pen carriage. The plotter's ROM contains the necessary instructions to



generate vectors, and any other special graphics capabilities the plotter may have. Typical of these capabilities would be the drawing of circles and alphanumeric characters. ROM and its associated instructions are sometimes known as firmware.

Despite the exquisite complexity of the plotter, software, and firmware, the bottom line is that an ink pen is mechanically drawing a line on paper. In some sense, the pen, ink, and paper are the least important—and the most important—part of making a picture. The trail of ink is only a record of where the pen passed, and yet this trail is the absolute essence of the picture.

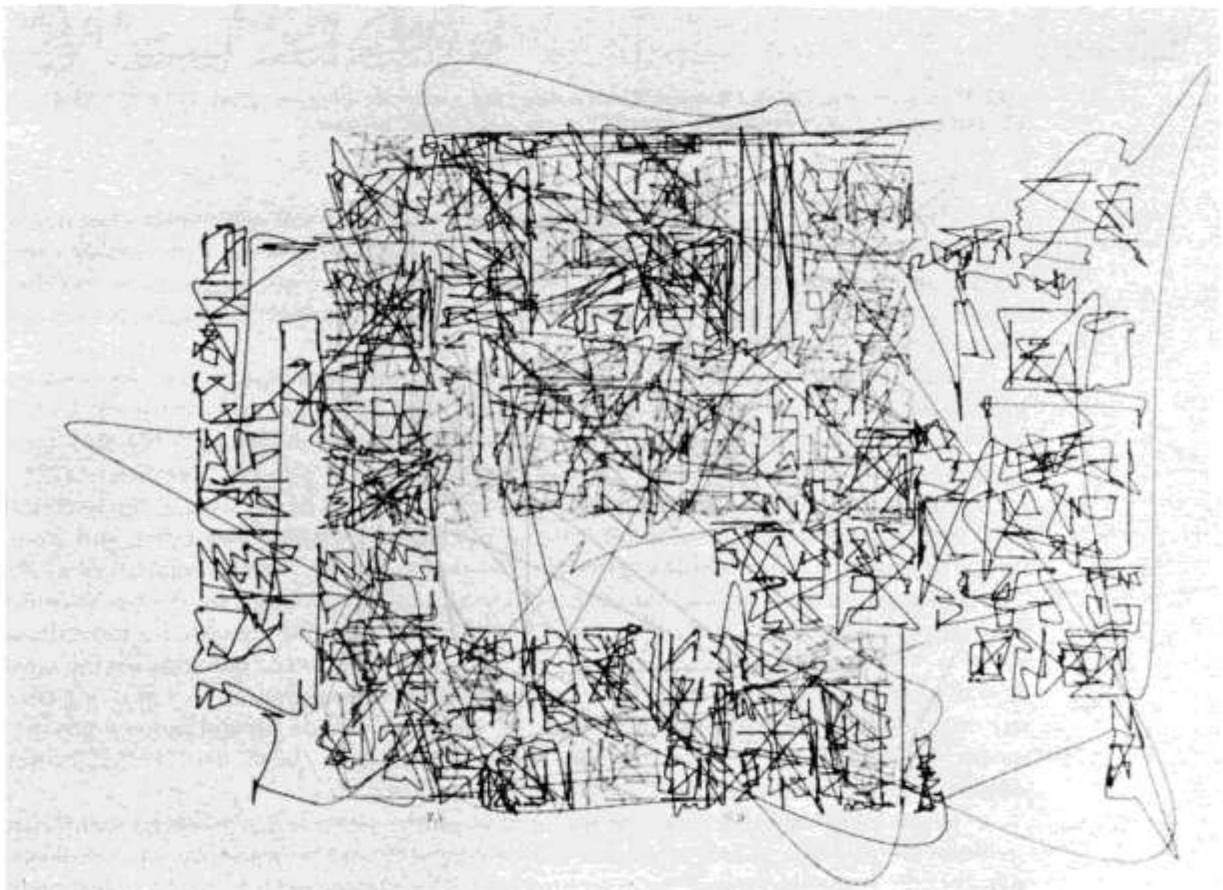


FIG. 16. Colette and Charles Bangert. Grass series. 1983 plotter drawing on paper, 11" x 17". IBM PC and a Hewlett-Packard plotter. © 1983 Colette and Charles Bangert.

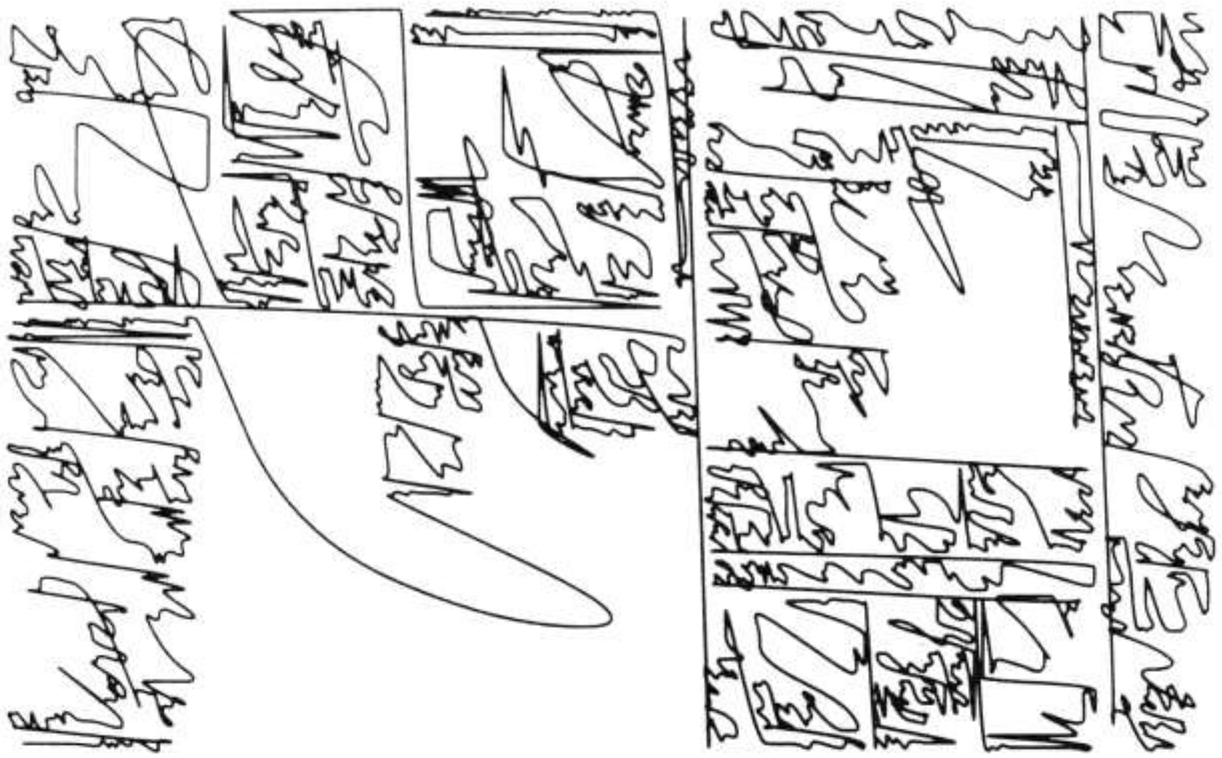


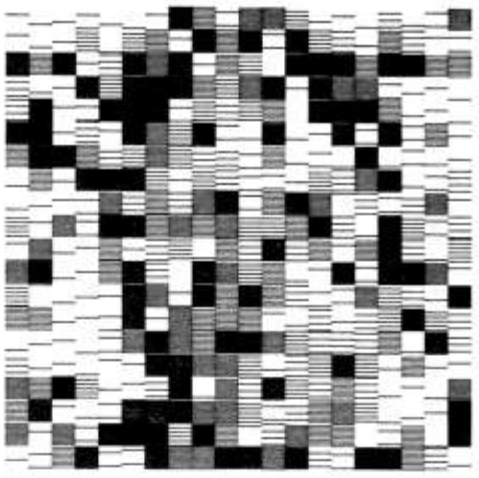
FIG. 17. Charles and Colette Bangert. Circe series. 1984 plotter drawing on paper, 11" x 17". IBM PC and a Hewlett-Packard plotter. © 1984 Colette and Charles Bangert.

There is a wide diversity of papers and pens available for use with plotters but not all are suitable for artistic purposes. Fiber-tip pens are probably the most commonly used. They produce good results in a plotter, even at relatively high plotting speeds. Yet they really are not suitable for any type of work where longevity is important. Most fiber-tip pen inks simply are not permanent.

Ball-point pens of various types are available for plotter use. Some use a water-soluble ink similar to that used with fiber-tip pens, and others use an oil-base ink. Unlike fiber-tip pens, ball-points produce very consistent line widths during their life. But, once again, their inks are of questionable longevity and should be tested for permanence.

The best type of pen for plotter use, from the standpoint of the artist, is the technical pen. Technical pens have been in use for many years by draftsmen, engineers, and architects. They have refillable wet-ink cartridges. The pen points are hollow metal tubes with a wire plunger inside the tubing. Materials of varying hardness are used for the tip. Stainless steel is the standard tube material, but jewel and tungsten carbide are used for more durable pens. Because of the special demands of plotter use, tungsten carbide tips are the most desirable. As the pen will literally travel many miles over the paper, as well as being constantly raised and lowered onto its surface, the tungsten carbide tip will provide the best service. Technical pens come in a variety of line widths, from about 0.01" to 0.05". They will maintain very consistent line widths throughout their life.

Black india ink is the standard ink used in plotter pens. It is pigmented with finely divided carbon particles and possesses excellent opacity and permanence. Many colored inks are also available. Several manufacturers produce pigmented inks for technical pens. These pigmented inks have superior permanence and are excellent for use in plotter pens. They are also intermixable and compatible, which means that a very large range of colors



can be mixed from the standard colors. Compatibility between different manufacturers' products may not exist, causing problems with ink clogging in pens.

PAPER

Vast reams of paper are consumed by printers and plotters. Most of this paper is quite adequate for its intended purpose and for artists beginning to experiment with computer graphics. The primary problem is one of longevity. We are all familiar with the rapid degradation of newsprint. Ordinary white bond paper will undergo a similar degradation, but on a time scale of years rather than a period of days or weeks as with newsprint. Such paper is usually made from wood chips using the sulphite process. The paper manufacturing process leaves a residue of acidity in the paper, which eventually causes self-destruction. Higher quality paper is manufactured from cotton fiber or reprocessed cotton rags. It is known as rag paper and will frequently be described as having a certain percentage of rag content. Obviously, 100 percent rag paper is the highest quality.

It is easy to find 100 percent rag art papers, but obtaining 100 percent rag paper for use with a computer is another matter. For instance, it may be very difficult to find such paper with sprocket holes for use with printers or drum plotters. Flatbed plotters and drum plotters that do not use sprockets to move the paper allow a choice from a large selection of conventional art papers. The main limitation on their use may be the amount of clearance under the pen carriage. Many art papers, such as a heavy watercolor paper, may be thick enough to prevent proper operation of the pen. When the pen goes down, it may dig into the paper and be unable to move. Some of the other problems encountered may include clogging of the pen because of fiber accumulation in its tip, and either excessive ink absorption or inadequate ink absorption by the paper. Papers have a great variety of surface textures. They range from very smooth to very rough, with every gradation between. For my personal use, I have found that relatively smooth paper, with a surface akin to hot-press paper, produces good results with technical pens.

Plotters in commercial and industrial applications use other types of media such as transparent film and polyester drafting film. Drafting film is semiopaque and is coated to accept ink. The ink tends to lie on the surface of the film rather than being absorbed into fibers as with paper. Depending on its chemical composition, transparent film varies in permanence. Polyester drafting film has excellent durability and permanence and is generally considered to be permanent, although the quality of line is very different from what is obtainable on paper.

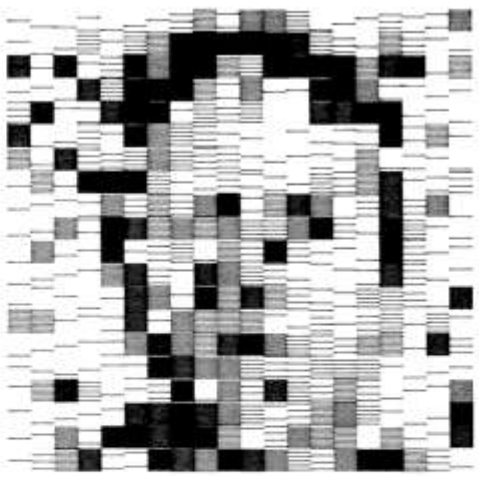
3. UNDERSTANDING GRAPHICS SOFTWARE

IN SOME WAY a computer without software is a bit like a canvas and palette of colors without an artist. Forgive this obvious simile, but there is, nonetheless, a great deal of truth in it. The computer hardware, even in a very simple microcomputer, is capable of prodigious tasks; but it remains absolutely mute and inert unless there is software to control and direct it.

If you are a business person and want to make a chart of sales for the last three years, lovely software is available for this purpose. If you are an engineer and want to lay out a printed circuit board, finding the software should be no problem. If you are an artist and you want to make pictures with your computer, things become somewhat more difficult. There is some software available, most of it aimed at creating pictures on the color monitor. These software packages for drawing on the screen are known as paint programs or paint systems. Some of these are very good. Most operate on the principle of mimicking the conventional artistic processes of drawing and painting. In other words, the cursor on the screen is controlled through the keyboard or some other input device, such as a digitizer pad or joystick. Images are drawn on the screen and can then be manipulated in a variety of ways.

Paint programs are developing rapidly. We should see an increase in both the quantity and quality of these programs for microcomputers. The ultimate problem with paint programs is that while they can very cleverly manipulate images, they are incapable of generating images on their own. The artistic vision in the twentieth century has produced an extraordinary variety and diversity—from Malevich's all-white paintings to the photorealist paintings of Richard Estes. Between these poles lies a seemingly endless proliferation of styles. In fact, the twentieth century may be viewed as a period in which stylistic diversity is the norm. The wild medley of visual styles from which the artist may draw is not necessarily available from existing computer software. So, to fully exploit this new medium, the artist must learn as much as possible about its potential.

While paint programs may serve some needs, at least some knowledge about programming is necessary. Artistic creation is not at all impossible if you do not write your own programs. But even a limited amount of programming experience will help you understand both the limitations and the potentials of computer graphics. If you have never programmed a computer, the thought of writing software may seem onerous. Yet the possibilities greatly outweigh the drawbacks. In fact, necessity may force you to become a programmer, for at this point there is simply a limited amount of software available for the artist, and many programs do not support hard-copy devices, such as printers or plotters.



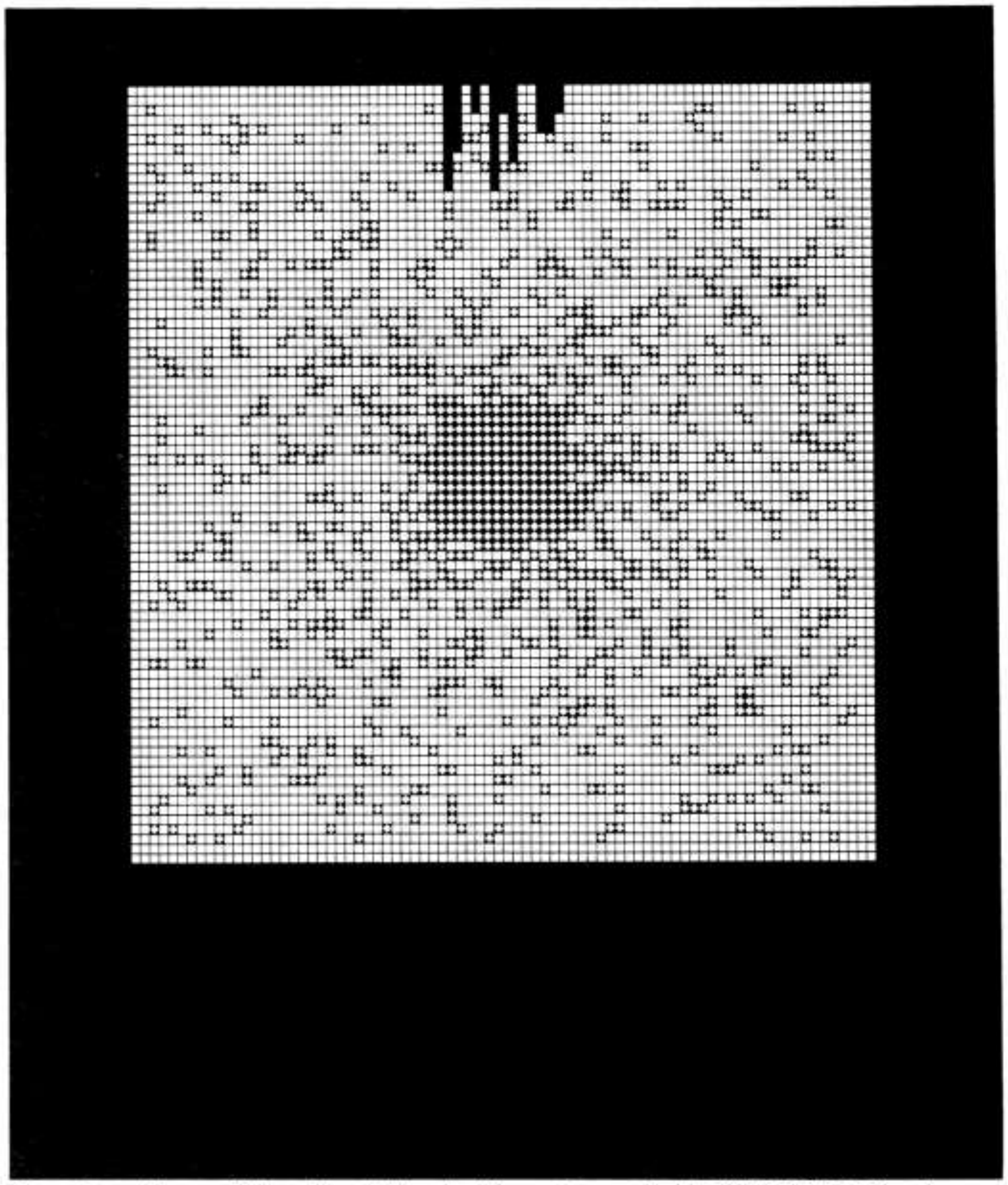


FIG. 1. Aaron Marcus. *Urbane Nova,* 1972 color lithograph converted in 1984. DEC PDP-10 and a Mergenthaler Linofilm Quick Phototypesetter. © 1972, 1984 Aaron Marcus.

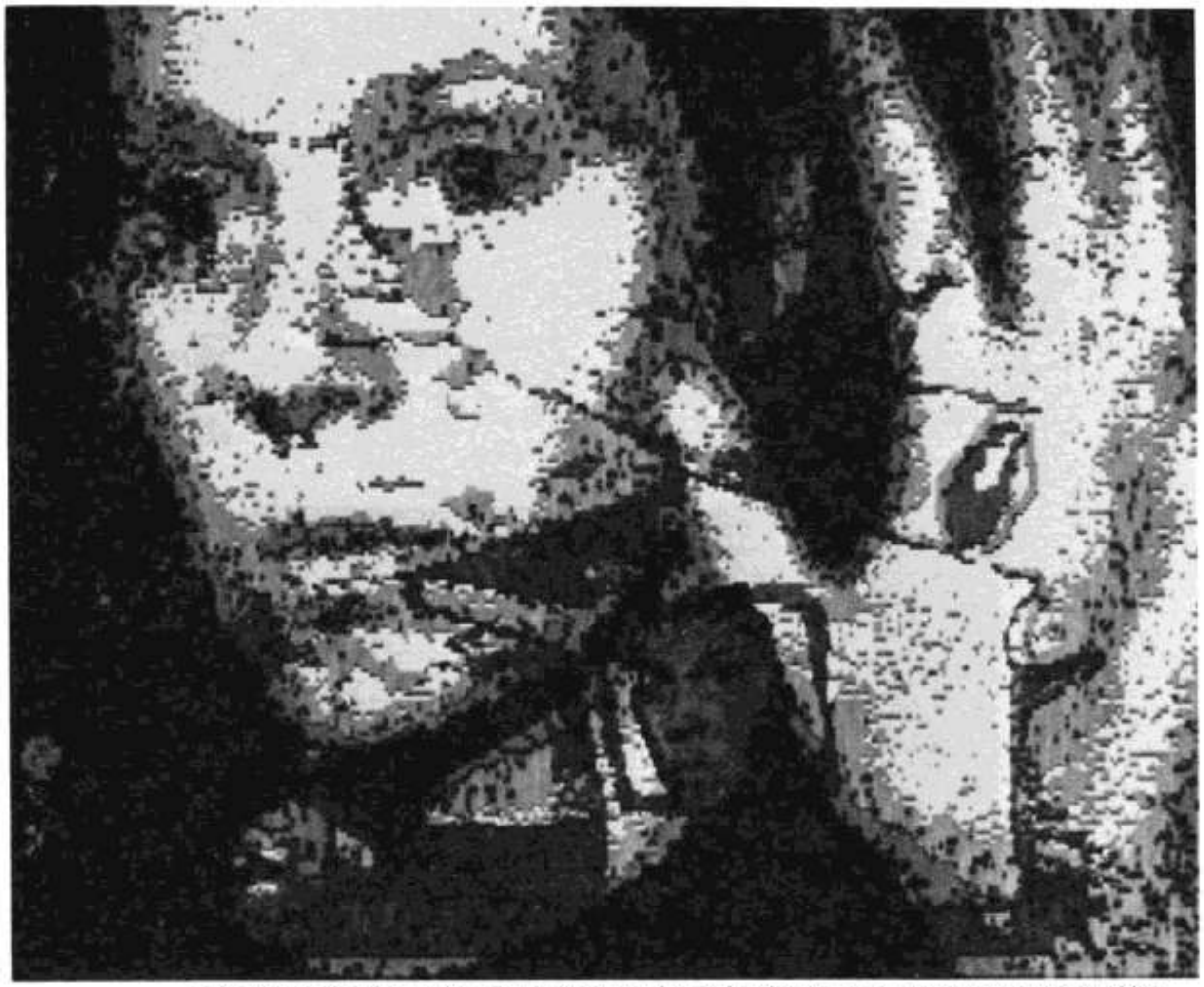


FIG. 2. Copper Giloth. Wisconsin series. 1983 plotter drawing on paper. UV-1 Zgrass system, video camera, video digitizer, and a Hewlett-Packard plotter. © 1983 Copper Giloth.

Most importantly, programming represents a positive opportunity. Programming offers individualistic twentieth-century visual artists the means to expand their own artistic expression. The nature of computer graphics provides a fundamentally different way of making pictures. If the artist is curious about exploring this medium, programming is the means to do so.

Programming is simple. Since BASIC is available in almost all low-cost microcomputers, we will proceed with a series of simple BASIC programs that can be used with a plotter or adapted for a video display. While BASIC may not be the choice of an advanced programmer, it is a universal microcomputer language, it usually has graphics primitives, and it comes free with your microcomputer.

BASIC DIALECTS

MOST MICROCOMPUTERS BASICs are similar and roughly compatible with each other; but there are occasionally differences that will have important programming consequences. The Microsoft BASIC that is used in the IBM Personal Computer and PC*jr* was utilized in all of the program listings that will follow. These listings may require minor modifications to allow them to run on other microcomputers. However, the primary difficulty in programming compatibility is interfacing with the graphics hardware of the individual types of microcomputers. Each specific machine will have its own command to draw a line on the screen. There are no graphics standards among microcomputers. The com-

puter graphics industry has proposed that standards be adopted so that all such commands, say for the drawing of a line from one point to another, would be uniform. Progress is being made toward the adoption of such standards, but in the meantime it is still necessary to convert and interpolate graphics commands between one machine and another. For instance, the Apple II will put a point on the screen with the command PLOT X,Y. A similar command for the IBM PC would be PSET X,Y.

High-level languages, like BASIC, FORTRAN, and Pascal, have no intrinsic graphics commands or primitives associated with them. The graphics primitives are added features of the language. These primitives are directed toward the specific graphics display hardware in each microcomputer. Obviously, all high-level languages have some sort of method of outputting ASCII characters to the screen. But very few except the resident

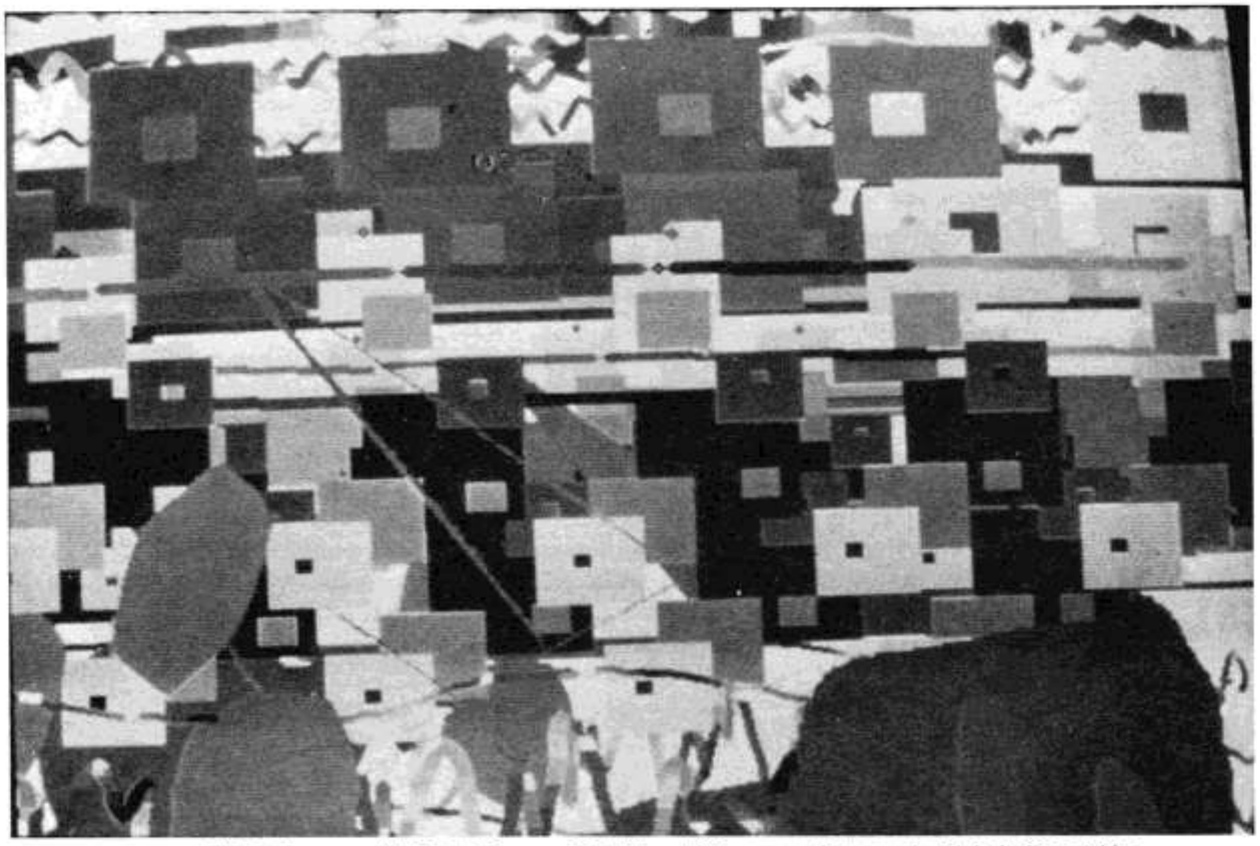


FIG. 3. Laurence M. Gartel. *Forever Television,* 1981 screen photograph. Digital Effects Video Palette paint system. © 1981 Laurence Gartel.

BASIC have any graphics commands. One remedy for this problem is to write your own graphics primitives that will access the graphics display hardware in your computer. This requires a thorough knowledge of a low-level language, like assembly or machine language. Needless to say, such an approach is not for the casual user. In some cases, professional programmers have written routines and packages of graphics primitives, which can be obtained for certain languages and computers. It might be useful to point out that in the world of larger computers—such as minicomputers and mainframes—the question of graphics primitives is usually not a problem for the user. Typically such systems will use high-resolution terminals that have graphics firmware.

LOGO

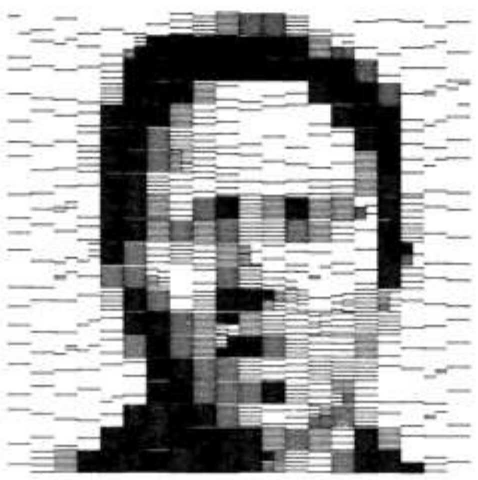
AN IMPORTANT EXCEPTION to the paucity of graphics primitives built into high-level microcomputer languages is LOGO. Developed at the Massachusetts Institute of Technology in the 1970s, LOGO is an interpreted language. LOGO has its roots in LISP, or List Processing Language, which has been closely associated with artificial intelligence research. LOGO was specifically developed as a language that computer novices could use to learn programming. It is easy to learn and it is also very different from BASIC. Most important, LOGO includes a set of graphics primitives. LOGO's approach to graphics is rather unconventional. The central feature of the graphics operation is the use of a "smart" cursor, called a turtle. The turtle draws lines on the display. The turtle always knows exactly where it is and in what direction it is pointing. Thus, the turtle can be moved by merely directing it forward, backward, left, or right. Turns are specified by angular quantities. This conception of the graphic space is in marked contrast to that underlying the rest of computer graphics, which use the Cartesian coordinate system. We will discuss this coordinate system soon, but the LOGO turtle does present a very interesting and powerful alternative for the artist interested in computer graphics.

Another advantage of LOGO is that it is rapidly becoming available for almost all microcomputer systems. It is simple to learn, usually inexpensive, and, some say, has the potential to rival the universality of BASIC in the microcomputer world. A disadvantage with some of the current implementations of LOGO is that not all have the ability to control output to an external device, such as a plotter.

COORDINATE SYSTEMS: HOW TO DEFINE SPACE WITH NUMBERS

THE CARTESIAN COORDINATE SYSTEM

René Descartes (1596–1650), a French philosopher and mathematician, created a method which united elements of geometry and algebra. His work, in *La Géométrie* (1637),



(100,100), that point will become our active point, or pen position. In relative mode, the same box would have coordinates of A (0,0), B (0,100), C (100,0), D (0,−100). (See Fig. 6.) Note that each new set of coordinates is always relative to the last point.

WORLD COORDINATES

Another type of coordinate system that is frequently encountered in discussions of computer graphics is world coordinates. Up to this point, we have been considering the actual device or physical coordinates of whatever type of machine we happen to be using. World coordinates refer to a coordinate system unrelated to the graphics display. The units of such a world system might be feet or meters, seconds or years. In many cases the software or firmware of a graphics display device will allow the user to specify the world units of the display. These world units could be different from the device units. The part of the world coordinate system that can be seen on our actual device is called the window. Obviously, the world coordinates must be converted into device coordinates by software or firmware.

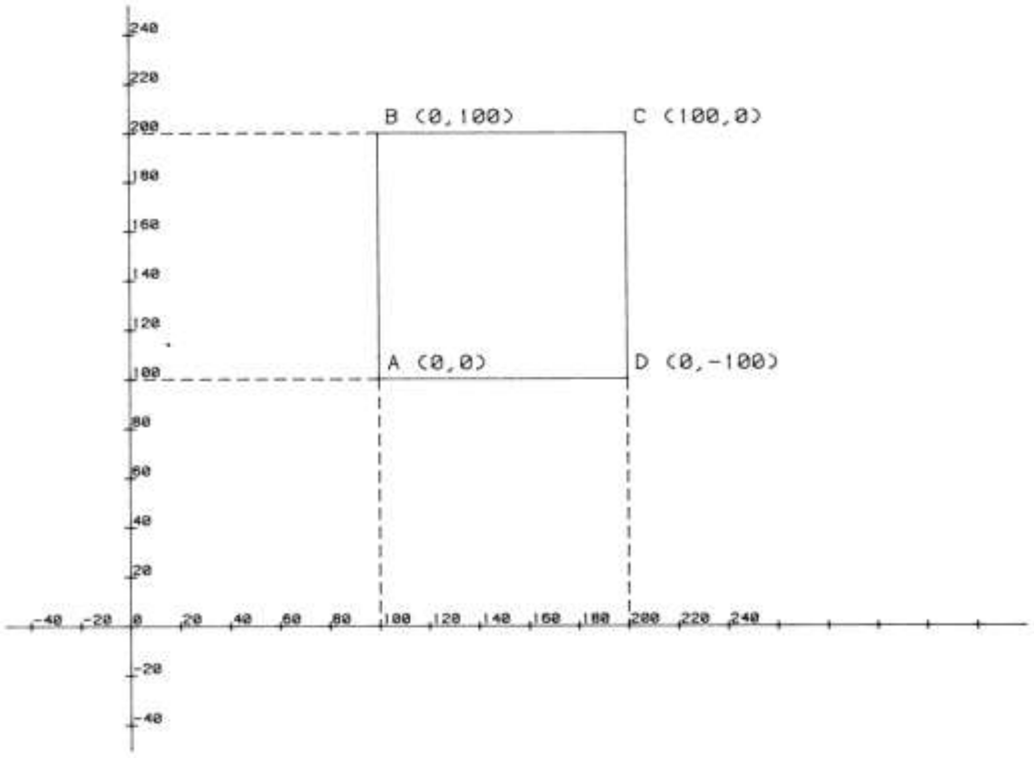


FIG. 6. Relative Coordinates

Fig. 4 shows four points plotted in the same respective positions in each of the four quadrants; A = 100, 100 (100 positive *x* axis units, 100 positive *y* axis units); B = 100, = 100; C = −100, −100; and D = −100, 100.

ABSOLUTE AND RELATIVE

Frequently, graphics devices will have two modes of specifying the method of naming the coordinates. These two modes are absolute and relative. Absolute mode refers to the physical coordinate system of the device. In other words, if the device locates the origin (0,0) at the lower left corner, as in Fig. 5, we could specify the coordinate pairs of the corners of a box by these numbers: A (100,100), B (100,200), C (200,200), and D (200,100).

Relative mode means that coordinate pairs are specified relative to the location of the pen—in the case of a plotter—or to the last point referenced—in the case of a video display. Relative mode simply means that wherever the pen or active point is, that point, in effect, becomes the origin (0,0). Thus, any new point will be in reference to that point. In the previous example of a box with its lower left corner at the absolute coordinates of

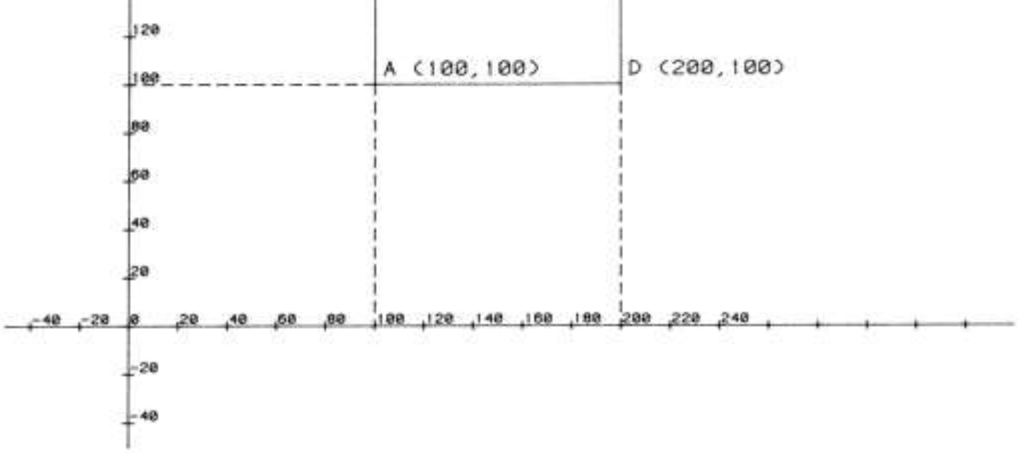
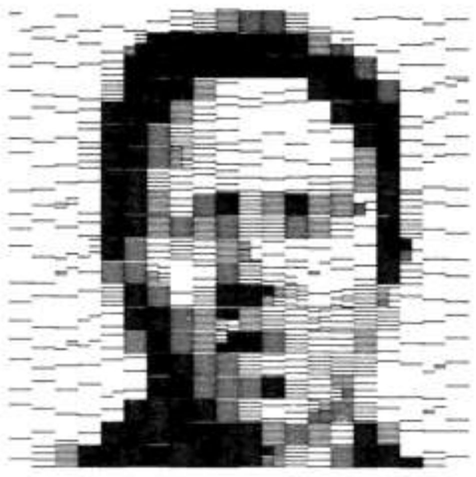


FIG. 5. Absolute Coordinates



(100,100), that point will become our active point, or pen position. In relative mode, the same box would have coordinates of A (0,0), B (0,100), C (100,0), D (0,−100). (See Fig. 6.) Note that each new set of coordinates is always relative to the last point.

WORLD COORDINATES

Another type of coordinate system that is frequently encountered in discussions of computer graphics is world coordinates. Up to this point, we have been considering the actual device or physical coordinates of whatever type of machine we happen to be using. World coordinates refer to a coordinate system unrelated to the graphics display. The units of such a world system might be feet or meters, seconds or years. In many cases the software or firmware of a graphics display device will allow the user to specify the world units of the display. These world units could be different from the device units. The part of the world coordinate system that can be seen on our actual device is called the window. Obviously, the world coordinates must be converted into device coordinates by software or firmware.

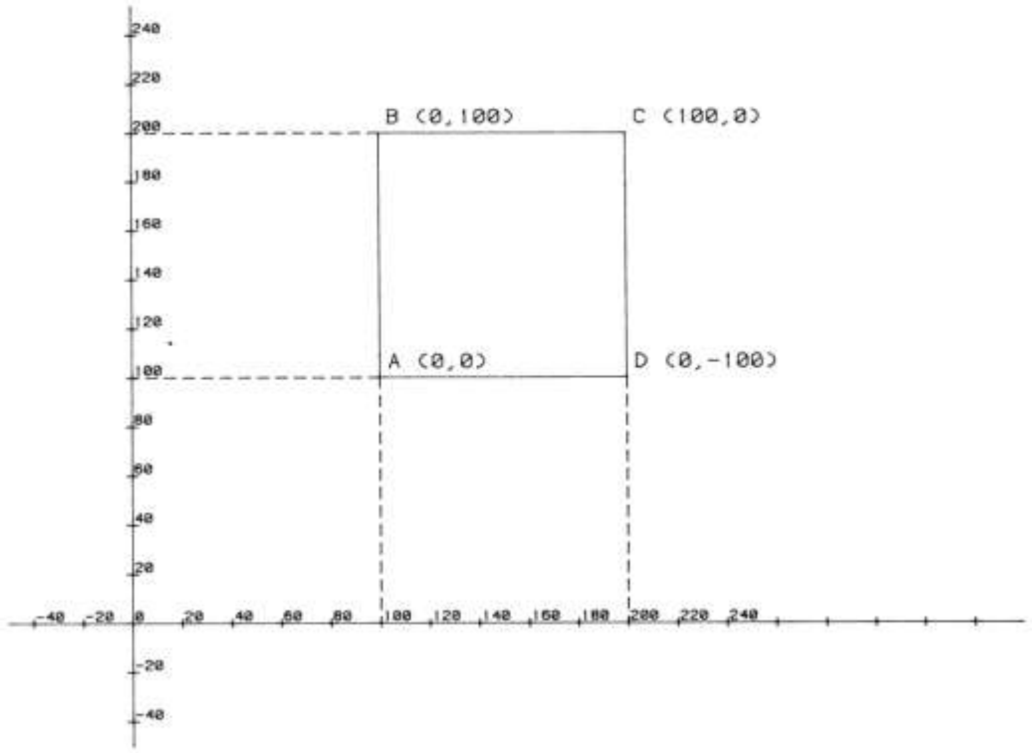


FIG. 6. Relative Coordinates

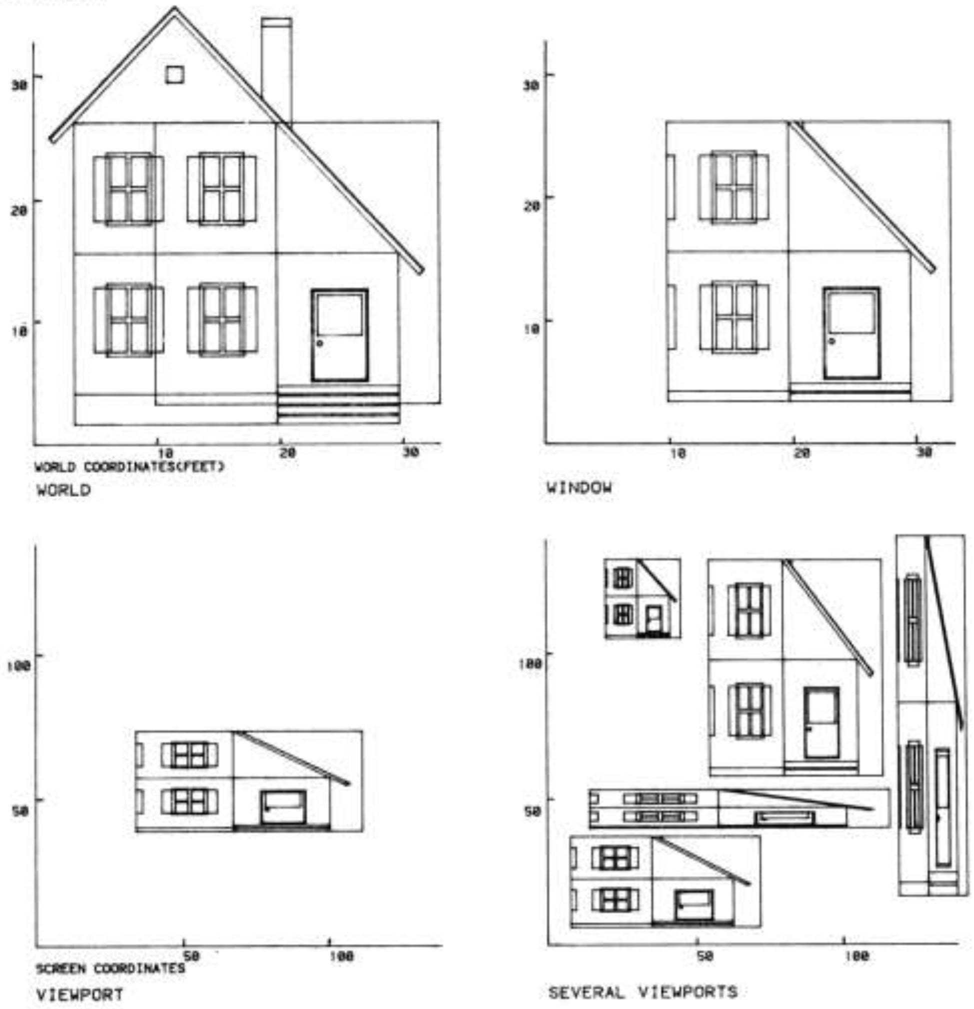


FIG. 7. World Units, Device Units, Windows, Viewports

An extension of the idea of windows on the world coordinates is the viewport. The viewport permits only a section of the window to actually be displayed on our device. The segments of the lines or alphanumeric characters are not displayed outside of the viewport. They are said to be clipped.

VECTOR GENERATION: DRAWING LINES

EVEN VERY SMALL CHILDREN can easily connect two dots together with a line. It is a different matter with a computer. To draw a line on a screen, or draw a line with a plotter, is not a trivial action. In most cases, the end user of computer graphics equipment does not have to worry about the drawing of these lines, or vectors. The hardware or firmware in the display device will transparently construct the line. Because this fundamental procedure is at the heart of any graphics operation, it seems appropriate that we momentarily consider what is occurring.

We select a starting point of X1 and Y1, and an ending point of X2 and Y2. The BASIC listing below shows these inputs to the Vector Generation program in lines 40 through 70. Line 90 then finds the ratio of sides of a triangle formed by the vector—the hypotenuse of the triangle—and two sides of the right triangle—the distance of the X and Y legs of the right triangle. This ratio is called the Y increment, and the variable name is YINCR. A FOR . . . NEXT loop then plots a point on every pixel along the dimension between the old X and

new X position. As the loop moves along, the Y distance is being incremented with the value of the YINCR. While this listing is not necessarily the most efficient way of drawing a line, it does illustrate, in a simple way, the issues involved in drawing vectors. Happily, almost all computer graphics devices have such vector capabilities, so when we want to draw a line, the machinery does the calculating for us.

```
10 ' Vector generator
20 SCREEN 2
30 CLS
40 INPUT " X1 coordinate";X1
50 INPUT " Y1 coordinate";Y1
60 INPUT " X2 coordinate";X2
70 INPUT " Y2 coordinate";Y2
80 CLS
90 YINCR=(Y2-Y1)/(X2-X1)
100 FOR I=X1 TO X2
110       YYINCR=YYINCR+YINCR
120       PSET(I,Y1+YYINCR)
130 NEXT I
```

STANDARD GRAPHICS COMMANDS

THE STANDARD COMMANDS used to draw lines are all similar but they all have slight variations in their syntax. Essentially they all boil down to two simple ideas: move, with the pen up, to a specified place; and move, with the pen down, to a specified place. Frequently, video displays will use software commands that will draw a line from one point to another in a single command. But most plotters use some sort of pen up, pen down command structure. Thus a typical BASIC software command to a plotter would consist of:

```
PRINT #1,MOVE$;OLD X COORDINATE VARIABLE;OLD Y COORDINATE VARIABLE
PRINT #1,DRAW$;NEW X COORDINATE VARIABLE;NEW Y COORDINATE VARIABLE
```

Naturally, instead of variables we could have constants, but for most purposes variables will be used. Indeed, good programming practice makes the use of variables mandatory. In the above example, the device would move, with the pen up, to the starting position denoted with the old *x* and *y* coordinate variables, and then move to the finishing position denoted with the new *x* and *y* coordinate variables.

TURNING A GRAPHICS DISPLAY DEVICE ON AND OFF

Many graphics devices require a software command to turn them on or enable them to react to commands sent by the host computer. Often this will be several ASCII characters that must be sent to the device. A similar situation exists with the graphics displays of many microcomputers. Often it is necessary to send a software command that puts the machine in a graphics mode. If the device has a command to turn it on, there will usually be a corresponding command to turn it off. While remembering to use this software command to enable a device to react is a trivial matter, forgetting it can nonetheless cause a novice much frustration when attempting to write software.

OTHER GRAPHICS COMMANDS

Depending on the amount of "intelligence" built into your display device, you may be able to send commands that will greatly expedite drawing of forms and symbols. One of the most common examples is drawing circles. Writing software necessary to draw a circle, although not difficult, can be cumbersome in terms of computation time on a microcomputer. Thus, if your plotter has a command to draw a circle, it will greatly speed up the drawing process. Modern low-cost plotters frequently have an impressive set of commands available. These would include drawing circles, arcs, and symbols such as boxes, triangles, and the like. Most plotters also have a ASCII character set that will allow them to print out characters. Usually, the size of these characters can be set from software commands. Instead of generating a pattern of pixels on the screen as does a video character generator, a ROM in the plotter actually contains a series of vectors that describe a letter. The plotter then plots those vectors to draw the ASCII character. Some of the other commands available may give the plotter the ability to draw dotted and dashed lines, to select viewports, and to slant and rotate the ASCII character set.

Needless to say, the types of graphics manipulations on a video monitor are somewhat different, but a "smart" display will have a variety of methods to enhance the creation of pictures. Probably the most obvious difference would be in a color display, where a number of different colors are available. Unlike plotters, a video color display may have a fill command that fills an area with a specified color. This is extremely useful. Circle and arc commands are sometimes available, and of course, an ASCII character set is ălmost universally standard in any type of video display.

PLOTTER LANGUAGE FORMATS

As an example of the format of commands to plotters, the listing below shows how a BASIC language program would send the plotter a command to move, with the pen up, to an *x* and *y* coordinate of 100 units, and then move, with the pen down, to an *x* and *y* coordinate of 200 units. This would be the command sequence to draw a diagonal line. This listing is then repeated incorporating the commands used by several of the major plotter manufacturers, to give the reader an idea of the variations in command structure.

Our example will assume that we have two sets of variables, the STARTX and STARTY coordinates, and the ENDX and ENDY coordinates.

```
10 STARTX=100
20 STARTY=100
30 ENDX=200
40 ENDY=200
```

Hewlett-Packard Graphics Language (HP-GL):

```
100 PRINT #1,"PUPA";STARTX;";;STARTY
110 PRINT #1,"PDPA";ENDX;";";ENDY
```

Houston Instruments (DM/PL):

```
100 PRINT #1,"U";STARTX;STARTY
110 PRINT #1,"D";ENDX;ENDY
```

Tektronix 4662 and 4663 Plotter Language:

```
100 PRINT #1,"!AX"+STR$(STARTX)+STR$(STARTY);
110 PRINT #1,"!AY"+STR$(ENDX)+STR$(ENDY);
```

IBM XY749 and XY750 Plotters:

```
100 PRINT #1,STR$(STARTX)+STR$(STARTY)+"HK"
110 PRINT #1,STR$(ENDX)+STR$(ENDY)+"IK"
```

These examples reveal the essential similarity, but, at the same time, the slight divergence of each command from the others. Once you become familiar with whatever software system you might be using, the command format becomes a trivial problem. But for the first-time user, the command format can be frustrating. In most cases, the actual command codes sent to the plotter are ASCII characters, but the numeric or variable values can be represented by ASCII characters or by actual numeric or variable values. Further complicating the issue is the format of the numbers. Some machines will accept only integers, in which case coordinate variables must be defined as integers in the program. Various dialects of BASIC also use different formats and delimiters in their respective PRINT statements. A delimiter is a character that is used to separate different values. Commonly, the space character (ASCII 32), commas, and semicolons are used as delimiters. In some cases, it may be necessary to send a carriage return control character (ASCII 13) at the end of a line.

COMMUNICATIONS: THE MACHINES SPEAK TO ONE ANOTHER

COMPUTER COMMUNICATIONS is a very large, complex, and important subject. In order to have a microcomputer transfer information to a peripheral device, such as a printer or plotter, you must establish a line of communication. Usually this is not an insurmountable problem, but at the same time, it can be frustrating and perplexing.

There are two major and several minor types of interfaces available for microcomputers. The most common are the serial and parallel interfaces. The serial interface is a standard that is defined by the Electronic Industries Association (EIA) and is officially known as the EIA RS-232C interface, or more simply RS-232. Occasionally, the parallel interface will be described as a "Centronics" printer port. Centronics is a printer manufacturer whose interface became very widely used for dot matrix printers. Both interfaces use a similar connector, known as a 25-pin DB-25. This connector is supplied in a positive and negative form, known unabashedly as male and female. The parallel interface is very widely available for printers, while the serial interface is more commonly used with letter quality printers, plotters, terminals, and modems.

A sophisticated variation of the parallel interface sometimes, although infrequently, seen on microcomputers is the General Purpose Interface Bus, or GPIB for short. It is also known as the IEEE-488 interface. This communications interface was originally developed by the Hewlett-Packard company for use with test instruments and is currently referred to as the Hewlett-Packard Interface Bus, or HPIB, by that firm.

Parallel interfaces are generally very easy to use, and in many cases, the operation of the interface is transparent to the user. In other words, the user does not have to set any parameters or worry about overflows of data at either end of the communication link. Unfortunately, the serial interface tends to be somewhat more complex. It is also more widespread. It would be useful, therefore, to consider some of the details necessary for the proper operation of serial communications.

EIA RS-232C

The serial interface, EIA RS-232C, has a variety of parameters that can be set through software and, sometimes, hardware. These parameters include the speed of transmission, or baud rate, and the digital nature of the characters that are interchanged between devices.

The character can be specified as to its length, whether or not parity will be checked, and the number of stop bits that are sent. The length of the character is generally seven or eight bits. Parity checking is a means of verifying the binary message that is transmitted. The stop bits signal the end of the transmission of one character. These settings must match both the computer and the peripheral device.

The baud rate is important because it represents the number of bits per second that are transmitted in the communication. A baud rate of 300, or 300 bits per second, is the standard rate of exchange for personal computer modems. If ASCII characters are being sent in the format of seven data bits and one stop bit, then approximately thirty-seven characters can be transmitted every second $(300/(7 + 1) = 37.5)$. Obviously, higher baud rates are desirable, but, for a variety of reasons, they are not always practical. High-speed modems are more expensive, for instance; and many types of peripheral devices are simply not capable of processing the data that is received. This is particularly true of plotters and printers, which are physically slow machines that cannot accommodate the data as fast as it is sent.

THE PLODDING PLOTTER

As an example of the communications problems a plotter will face, let us do a simple calculation. Suppose we want to make a solid square of color. We can do this by making a series of parallel lines. We will hatch the square with a pen that has a line width of about 1/64 inch (about 0.015 inch). To make the lines just touch one another and fill the area, we would need sixty-four parallel lines to make a 1-inch square. The computer must calculate the coordinate pairs for the start and end points. Thus, we have 64 lines with a pair of coordinates at the beginning and end, making a total of 128 calculations—64 × (1 starting coordinate + 1 end coordinate) = 128. If we make horizontal lines it is not necessary to recalculate the starting and ending *x* coordinates. If we perform the calculations with addition or multiplication, which are faster than division, using a BASIC interpreter, we get the following results: a Texas Instruments 99/4—a slow machine—will perform the necessary incrementing of the *y* axis points in roughly 2 to 4 seconds, depending on the algorithm used. Plotter pens can move anywhere from 2 inches per second to more than 20 inches per second. If we use a mean speed of 10 inches per second for the pen speed, it would take at a minimum of 6.4 seconds to fill the area. In practice, it will take longer because the pen may have to lift and then lower again to plot each new *y* axis step. Accelerating and decelerating the pen also adds to the time. Thus, in this example, even at the worst case with a slow computer, the data will arrive at the plotter about 50 percent faster than it can be plotted on paper.

Printers and plotters usually have some internal memory allotted to solve this problem of data reception. It is called the buffer. The buffer stores data as it is received and then relays it in the order in which it was received. This is known as "first in, first out," or FIFO. Sizes can vary greatly, but even a large buffer will eventually be filled. Unless the machine and buffer have some means to signal when the buffer is about to be filled, incoming data will overflow the buffer and be lost. Because of this, it is generally necessary to have some type of protocol, or "handshaking," that will prevent buffer overrun.

BUFFER OVERRUN

There are several methods of dealing with the problems of buffer overrun. Parallel interfaces and some serial interfaces will use an internal method called hardware flagging. Hardware flagging means that when the device's buffer begins to fill, say, half full, the device will send a signal to the host computer that will stop the transmission of data.

While the RS-232 interface was conceived as an industry-wide standard so that machines would be pin-by-pin compatible in their wiring, in fact, the machinery is widely divergent from this ideal. Theoretically, all of the pins—or signals—should be directly connected to their corresponding mates. Unfortunately, this is rarely the case. Because there are so many different wiring schemes, it is impossible to generalize. Someone hoping to hook up a computer and peripheral device with the RS-232 interface will, in many cases, be forced to experiment with different wiring schemes. Most commonly, this hardware flagging will potentially involve signals on pin 20 (Data Terminal Ready—DTR), pin 5 (Clear

To Send—CTS), pin 6 (Data Set Ready—DSR), and pin 8 (Data Carrier Detect—DCD). Instead of being wired pin-for-pin, these pins may have to be interconnected in some way. Frequently pins 6 and 20 are interchanged. In other words, the signal from pin 6 will be routed to pin 20, and vice versa.

The second strategy is simply to slow the transmission of data in some manner. There are two easy ways to do this. The first method lowers the baud rate so that the peripheral device consumes data faster than it is sent from the host computer. Another similar technique would be to use a software delay. As each command is sent to the peripheral, the program could jump to a subroutine that would contain a timing loop. This loop could even base its length on the distance between points being plotted, for instance. While such delay techniques are easy to implement, they have two important disadvantages. First, the peripheral may waste a great deal of time waiting for commands to be received from the host. Second, peculiar circumstances may arise in which there is a data overrun. In that case, data will be lost.

Many serial devices use a third method called software flagging or handshaking. This technique causes the peripheral device to send to the host an ASCII control character—ASCII 19—that says, in effect, "my buffer is almost full, stop sending data." Naturally, the host computer must be able to read this character and act upon it, for the desired effect. When the buffer has been sufficiently depleted so that the device is capable of receiving more data, an ASCII 17 is sent to the host, saying, "you may now safely send more data." This protocol is sometimes called XON/XOFF or DC1/DC3 handshaking. The following listing will show a rudimentary form of XON/XOFF handshaking that might be implemented in a BASIC subroutine. The program would jump to this subroutine each time data was sent to a peripheral device.

```
1000 'XON/XOFF Subroutine
1010 IF EOF(1) THEN RETURN
1020 A$=INPUT$(1,1)
1030 IF RIGHT$(A$,1)=CHR$(19) THEN 1050
1040 IF RIGHT$(A$,1)=CHR$(17) THEN RETURN
1050 A$=INPUT$(1,1)
1060 GOTO 1030
```

This XON/XOFF listing was written for an IBM PC and may require some modification to work with other microcomputers. It is also important to remember that many, but not all, devices are capable of sending the XON/XOFF control characters. If you are in doubt, check the specifications of the particular machines, or with the manufacturer.

4. FIRST PRINCIPLES: HOW TO MAKE YOUR PLOTTER DO SOMETHING/ANYTHING

NOW LET'S MOVE from the theoretical to actual practice. Much of the instruction to follow will be specifically in reference to operating a pen plotter, but in general terms it will apply to almost any type of display device. Our example will be the ultimate in simplicity—drawing a single line. In the case of the video display with a machine like an IBM PC, which has a large set of graphics primitives included in the BASIC language, we can draw such a line thus:

```
100 SCREEN 1                           'Initialize graphic mode
110 LINE(STARTX,STARTY)-(ENDX,ENDY)    'Draw a line
```

Such a program will obviously only operate on a machine with appropriate graphics capabilities. A display with only alphanumeric capabilities would not produce a line.

In the case of a peripheral device—our example will be a pen plotter with a serial interface—we must do three things. First, open a line of communications between the microcomputer and the plotter; second, initialize the plotter; and, finally, command the plotter to draw a line to the specified coordinates:

```
100 OPEN "COM1,1200,0,7,1" AS #1     'Serial communications opened
110 PRINT #1,"!AE"                   'Initialize plotter
120 STARTX=100
130 STARTY=100
140 ENDX=200
150 ENDY=200
160 PRINT #1,"!AX"+STR$(STARTX)+STR$(STARTY);     'Move
170 PRINT #1,"!AY"+STR$(ENDX)+STR$(ENDY);         'Draw
```

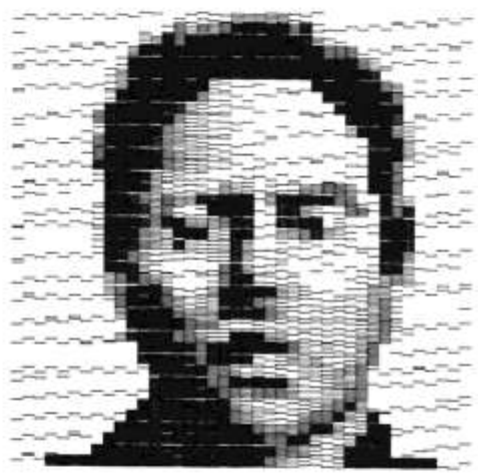
While this listing provides the essential details of drawing a line with a plotter, there are many things that may intervene to prevent the line from being drawn. Indeed, it is unusual when everything works on the first try. It is always frustrating, and sometimes maddening, to try to diagnose where the error lies. Often, the first thought is that the hardware is at fault. Somehow, one of the chips must be broken. In fact, some kind of hardware failure, though not impossible, is probably least likely.

The most probable source of difficulties is in the software. Check both the BASIC manual and the manual for your printer or plotter. The format of PRINT statements can be critical. Check the format of the numbers that are being sent to the peripheral device. Actually printing them out on the screen may be a helpful debugging procedure. The numbers or variables may have to be converted into integers or converted into strings of ASCII characters.

Another important source of potential problems is the cables and associated connectors. It may be necessary to rewire the RS-232 connection and experiment with various wiring arrangements. If you have made up your own cable and connectors, carefully check your soldering. Be certain that electrical continuity exists from one end of the cable to the other. The continuity can be easily checked with a voltmeter. If you do not have a voltmeter, a continuity checker can be improvised with nothing more than a 1.5-volt battery, a flashlight bulb, and two pieces of wire.

Most devices will have some sort of self-testing routine available. This self-test should be used if there is a failure to establish communications between the host computer and peripheral device. Finally, after testing for all of these possible flaws, it is possible that some sort of hardware problem may exist. Again, in most cases, hardware failure is least likely. Check with your dealer or the manufacturer for the procedure to follow in attempting to isolate the problem.

When the computer and peripherals fail to perform, the experience can be exceedingly irritating. Relax! Stop for a while and do something else, then go back to your labors. In most cases, the problem will be a simple one, but not necessarily an obvious one.



5. ONE HUNDRED SQUARES: BEGINNING GRAPHICS PROGRAMMING

NOW THAT we have learned to communicate with a plotter and have gained some familiarity with coordinate systems, let's try to draw a picture. Our first picture may not be an enduring masterpiece, but at least it will be fun. It will also give us a good starting place from which we can grow.

First, we will draw a square. We will assume that the plotter has been initialized and that MOVE and DRW (DRAW) strings have been defined previously. The input command of lines 70 and 80 of listing DRAWSQ asks us to give the program a starting value for the *x* and *y* coordinates. Line 110 commands the plotter to move to XO (*x* origin) and YO (*y* origin). We will complete the drawing of the square by moving the pen around in clockwise fashion. Our first DRW, therefore, does nothing to the *y* coordinate but adds the SIDE dimension to the *x* coordinate. In the second DRW, the *x* and *y* coordinates are incremented with the SIDE dimension. The third DRW places the pen at the *y* dimension plus the SIDE value, but uses the original *x* dimension. And finally, we head for home with the final DRW, which, of course, has the same coordinates as our initial MOVE.

```
10 DEFINT A-Z'     DRAWSQ
20 CLS
30 MOVE$="!AX"
40 DRW$="!AY"
50 OPEN "COM1:1200,0,7,1" AS #1
60 PRINT #1,"!AE"
70 INPUT "X origin";XO
80 INPUT "Y origin";YO
90 INPUT "Side of square";SIDE
100                GOSUB 260
110                PRINT #1,MOVE$+STR$(XO)+STR$(YO)
120                GOSUB 260
130                PRINT #1,DRW$+STR$(XO+SIDE)+STR$(YO)
140                GOSUB 260
150                PRINT #1,DRW$+STR$(XO+SIDE)+STR$(YO+SIDE)
160                GOSUB 260
170                PRINT #1,DRW$+STR$(XO)+STR$(YO+SIDE)
```

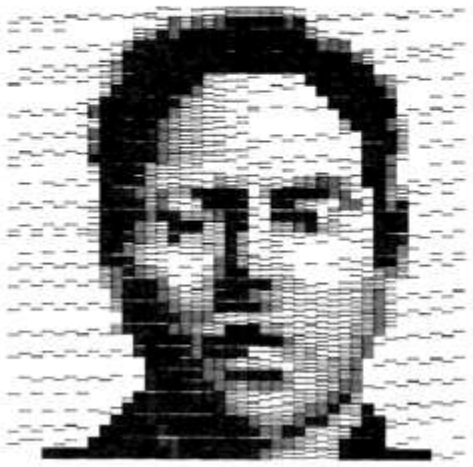
(continued on next page)

```
180                 GOSUB 260
190                 PRINT #1,DRW$+STR$(XO)+STR$(YO)+";"
200 PRINT
210 GOTO 70
220 '
230 '
240 '
250 '
260 'XON/XOFF subroutine
```

Once we have conquered the square, why not proceed to let our machine do one of the things that computers do best. Instead of one square, we will make many squares.

Listing DRAW100SQ has two FOR ... NEXT loops. These loops create a series of rows and columns. The square-drawing sequence will be repeated with each pass through the loops. We add several new inputs in line 90 and line 100. These inputs ask for the SIDE dimension (the dimension of the square) and the SIZE dimension (the distance between squares). The outer loop, ROW, controls the *x* coordinates. The inner loop, CLM, controls the *y* coordinates. We will use a STEP of 70. The program multiplies this STEP by nine. There will be ten STEPs along the ROWs and ten STEPs along the columns (CLMs). Thus, we will draw one hundred squares. The GOSUB 310 uses the XON/XOFF communications program to check the data flow between computer and plotter.

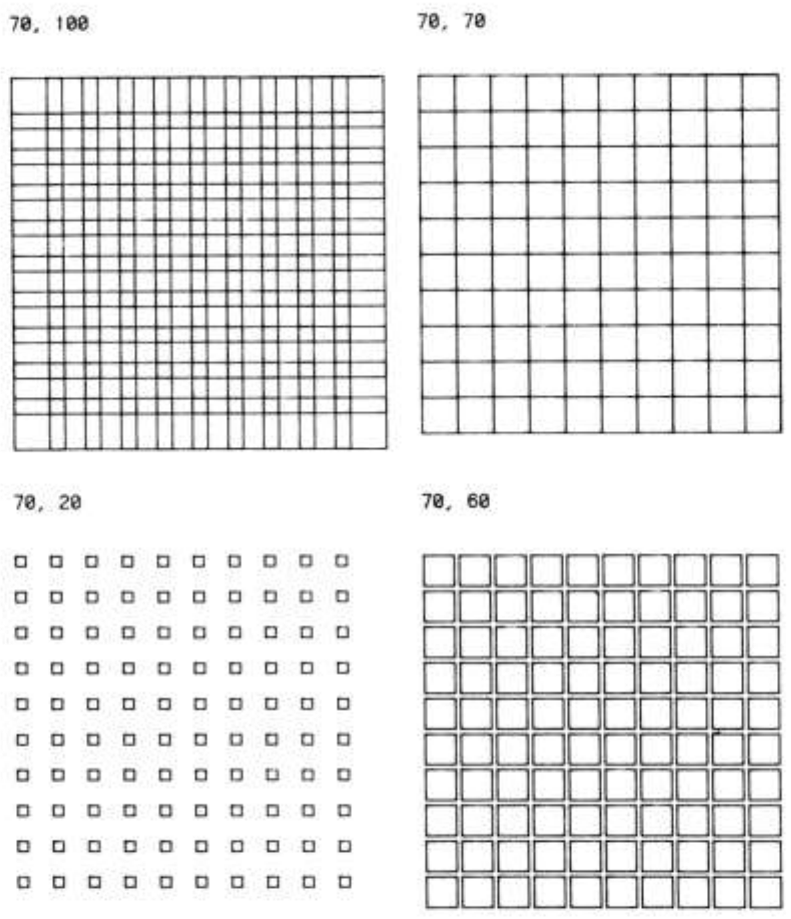
```
10 DEFINT A-Z'    DRAW100SQ
20 CLS
30 MOVE$="!AX"
40 DRW$="!AY"
50 OPEN "COM1:1200,O,7,1" AS #1
60 PRINT #1,"!AE";
70 INPUT "X origin";XO
80 INPUT "Y origin";YO
90 INPUT "Step size";SIZE
100 INPUT "Side of square";SIDE
110 FOR ROW=XO TO XO+SIZE*9 STEP SIZE
120       FOR CLM=YO TO YO+SIZE*9 STEP SIZE
130                 GOSUB 310
140                 PRINT #1,MOVE$+STR$(ROW)+STR$(CLM)
150                 GOSUB 310
160                 PRINT #1,DRW$+STR$(ROW+SIDE)+STR$(CLM)
170                 GOSUB 310
180                 PRINT #1,DRW$+STR$(ROW+SIDE)+STR$(CLM+SIDE)
190                 GOSUB 310
200                 PRINT #1,DRW$+STR$(ROW)+STR$(CLM+SIDE)
210                 GOSUB 310
220                 PRINT #1,DRW$+STR$(ROW)+STR$(CLM)+";"
230       NEXT CLM
240 NEXT ROW
250 PRINT
260 GOTO 70
270 '
280 '
290 '
300 '
310 'XON/XOFF subroutine
```



Each time the program runs through the five lines of MOVE and DRWs, it will draw a square, using the *x* and *y* coordinates provided by ROW and CLM. The variable that controls the dimension, SIDE, remains the same and is simply added, when needed, to the *x* and *y* coordinates.

In Fig. 1 we have four drawings made using different values. The STEP dimension is listed at the upper left corner of each drawing. After the STEP dimension, the SIDE is printed out. In the lower right drawing, using a slightly smaller value for SIDE—60—the grid takes on a different visual quality. In the lower left drawing, the SIDE is 20. Finally, in

FIG. 1.



the upper left drawing, we use a large SIDE value, 100. Once again, the grid takes on a new appearance.

Although these drawings could hardly be described as momentous, they reveal how rapidly drawings can be made and how simple changes in only a single numeric value can create images that are very distinct from one another.

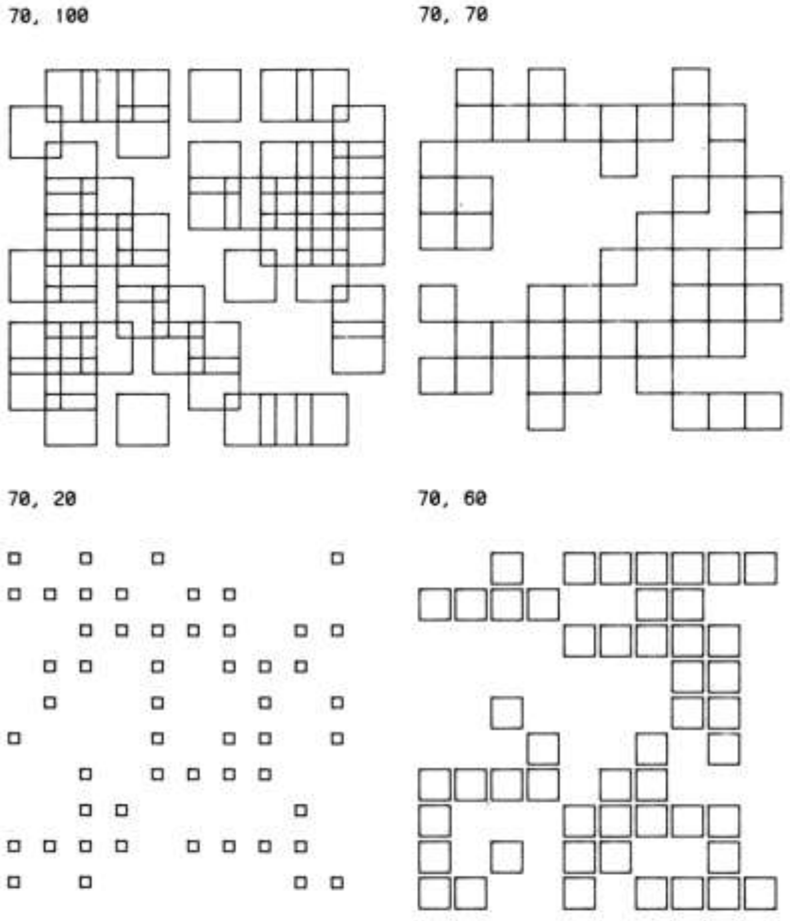
Using these same principles and the same values for SIDE, we will now add a variation to our theme. We modify our program in listing DRAW100SQ2, by adding line 130, which tests whether a randomly generated number is greater than a quantity we place in our program. Each time we invoke the random-number generator, a value is produced between 0 and 1, such as 0.3748938. We will use 0.5 for the value to test the RND number. In about half the cases, our program will execute the commands to draw a square, and the rest of the time it will jump down to the NEXT statement in line 240. The process will then repeat. We will use the same SIDE dimensions as before—20, 60, 70, 100.

```
10 DEFINT A-Z'     DRAW100SQ2
20 CLS
30 MOVE$="!AX"
40 DRW$="!AY"
50 OPEN "COM1:1200,0,7,1" AS #1
60 PRINT #1,"!AE";
70 INPUT "X origin";XO
80 INPUT "Y origin";YO
90 INPUT "Step size";SIZE
100 INPUT "Side of square";SIDE
110 FOR ROW=XO TO XO+SIZE*9 STEP SIZE
120       FOR CLM=YO TO YO+SIZE*9 STEP SIZE
130                IF RND>.5 THEN 240
140                GOSUB 350
150                PRINT #1,MOVE$+STR$(ROW)+STR$(CLM)
160                GOSUB 350
170                PRINT #1,DRW$+STR$(ROW+SIDE)+STR$(CLM)
180                GOSUB 350
190                PRINT #1,DRW$+STR$(ROW+SIDE)+STR$(CLM+SIDE)
200                GOSUB 350
210                PRINT #1,DRW$+STR$(ROW)+STR$(CLM+SIDE)
220                GOSUB 350
230                PRINT #1,DRW$+STR$(ROW)+STR$(CLM)+";"
240       NEXT CLM
250 NEXT ROW
290 PRINT
300 GOTO 70
310 '
320 '
330 '
340 '
350 'XON/XOFF subroutine
```

These simple mathematical manipulations produce distinctive visual results. Not only does Fig. 2 give us an immediate graphic depiction of what a random-number generator does, but we also have images that strongly evoke the abstract art of the early twentieth century. The pattern on the upper left seems to be of particular interest. Using a SIDE value of 100 and STEP of 70 creates random squares, but there are also several side effects. Other, smaller, squares and rectangular figures emerge from the network of lines. The un-

expected creation of these smaller squares and rectangles is a perfect illustration of the visual excitement computer graphics provide. If we had carefully and systematically planned to make this drawing by hand, we might have been aware of the results we would obtain. But art making is often a spontaneous activity. These unplanned effects encourage our experimentation. Creative impulses are stimulated by the playfulness of this process. All that was necessary in this case, was a rudimentary modification of the program. The machinery does the work for us—quickly, precisely, and patiently.

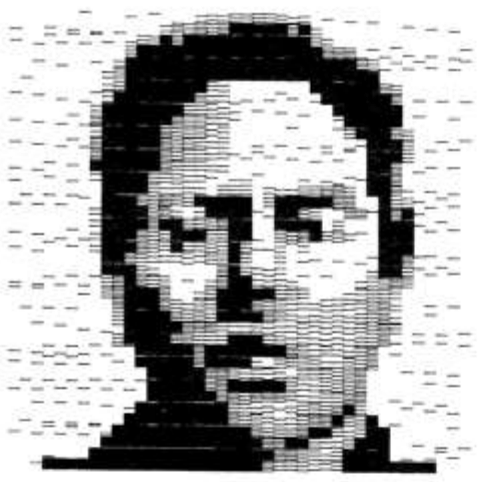
FIG. 2.



```
10 DEFINT A-Z'     DRAW100SQ3
20 CLS
30 MOVE$="!AX"
40 DRW$="!AY"
50 OPEN "COM1:1200,0,7,1" AS #1
60 PRINT #1,"!AE";
70 INPUT "X origin";XO
80 INPUT "Y origin";YO
90 INPUT "Step size";SIZE
100 INPUT "Side of square";SIDE
110 INPUT "Random side value";RVAL
120 DEF FNRVAL=RND*RVAL
130 FOR ROW=XO TO XO+SIZE*9 STEP SIZE
140      FOR CLM=YO TO YO+SIZE*9 STEP SIZE
150                GOSUB 360
160                PRINT #1,MOVE$+STR$(ROW)+STR$(CLM)
170                GOSUB 360
180                PRINT #1,DRW$+STR$(ROW+SIDE+FNRVAL)+STR$(CLM+FNRVAL)
190                GOSUB 360
200                PRINT #1,DRW$+STR$(ROW+SIDE+FNRVAL)+STR$(CLM+SIDE-FNRVAL)
210                GOSUB 360
220                PRINT #1,DRW$+STR$(ROW+FNRVAL)+STR$(CLM+SIDE-FNRVAL)
230                GOSUB 360
240                PRINT #1,DRW$+STR$(ROW)+STR$(CLM)+";"
250      NEXT CLM
260 NEXT ROW
300 PRINT
310 GOTO 70
320 '
330 '
340 '
350 '
360 'XON/XOFF subroutine
```

Investigating a similar use of random numbers, the computer and plotter will now be instructed to add a random value to each of the *x* and *y* coordinates at the corner of our square. To allow for the closure of the square, our last DRW must return to the starting point. Listing DRAW100SQ3 shows the modification to the input statements as well as the addition of FNRVAL to the DRW statements. Let us run the same set of numbers through the machine again—20, 60, 70, and 100. With each run, we will use a different value for FNRVAL. The FNRVAL value is the third number printed out at the upper left of each drawing.

Fig. 3 (p. 64) shows our familiar collection of one hundred squares. But this time they are distorted and twisted. The small boxes are not boxes. Perhaps they could be stones or snowflakes. All of the patterns in this illustration begin to look as if they were made by the artist's hand, not the computer. Adding random values creates images with a wholly different visual feeling from the rigid geometric figures.



```
10 DEFINT A-Z'    DRAW100SQ4
20 CLS
30 MOVE$="!AX"
40 DRW$="!AY"
50 OPEN "COM1:1200,O,7,1" AS #1
60 PRINT #1,"!AE";
70 INPUT "X origin";XO
80 INPUT "Y origin";YO
90 INPUT "Step size";SIZE
100 INPUT "Side of square";SIDE
110 INPUT "Random side value";RVAL
120 DEF FNRVAL=RND*RVAL
130 FOR ROW=XO TO XO+SIZE*9 STEP SIZE
140      FOR CLM=YO TO YO+SIZE*9 STEP SIZE
150              IF RND>.5 THEN 260
160              GOSUB 340
170              PRINT #1,MOVE$+STR$(ROW)+STR$(CLM)
180              GOSUB 340
190              PRINT #1,DRW$+STR$(ROW+SIDE+FNRVAL)+STR$(CLM+FNRVAL)
200              GOSUB 340
210              PRINT #1,DRW$+STR$(ROW+SIDE+FNRVAL)+STR$(CLM+SIDE-FNRVAL)
220              GOSUB 340
230              PRINT #1,DRW$+STR$(ROW+FNRVAL)+STR$(CLM+SIDE-FNRVAL)
240              GOSUB 340
250              PRINT #1,DRW$+STR$(ROW)+STR$(CLM)+";"
260      NEXT CLM
270 NEXT ROW
280 PRINT
290 GOTO 70
300 '
310 '
320 '
330 '
340 'XON/XOFF subroutine
```

FIG. 3.

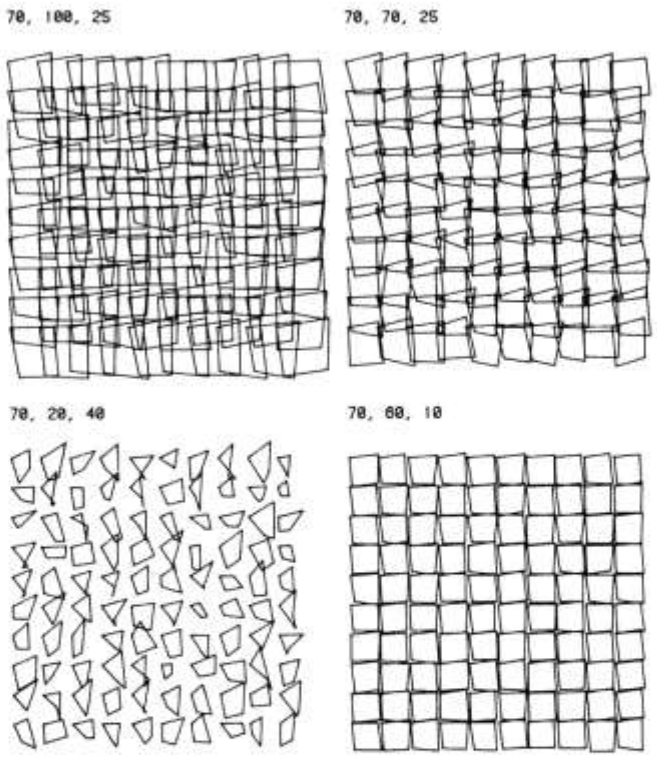
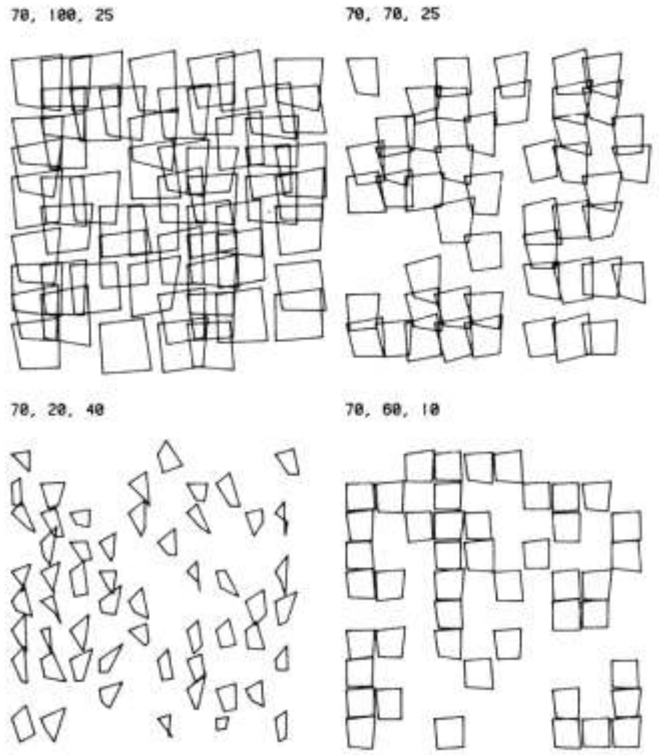
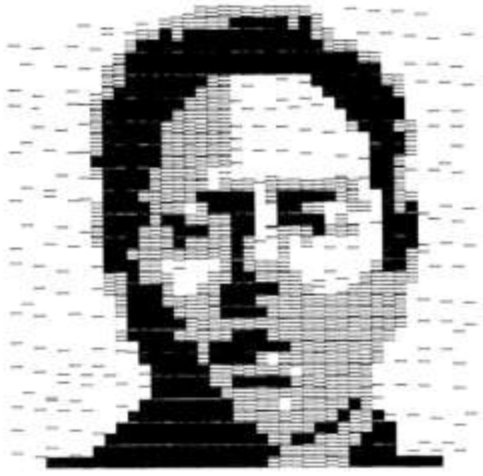


FIG. 4.



Continuing to cut and paste our BASIC program, we can now combine all of our tricks. We add the random number test, line 150. Fig. 4 demonstrates the ultimate results.

It should now be obvious that even a very simple program can provide a large amount of visual variety. You can glimpse the power and versatility of computer-generated images, using even a minimal program like One Hundred Squares.



HATCHPLOT

USING A SIMPLE modification to the existing square program, we will now create tonal values of gray—or whatever color ink you might be using. Creating tonal values with some type of linear medium, such as pencil or pen, is frequently done using series of parallel lines. This is called hatching. Printmakers and engravers brought the technique of hatching to a very high level. A good example is the engraving on bank notes or paper currency. We will now add instructions for hatching to our program.

```
10 DEFINT A-Z'    HATCHPLOT1
20 CLS
30 MOVE$="!AX"
40 DRW$="!AY"
50 OPEN "COM1:1200,O,7,1" AS #1
60 PRINT #1,"!AE";
70 INPUT "X origin";XO
80 INPUT "Y origin";YO
90 INPUT "Step size";SIZE
100 INPUT "Side of square";SIDE
110 INPUT "Hatching interval";HTCH
120 FOR ROW=XO TO XO+SIZE*9 STEP SIZE
130     FOR CLM=YO TO YO+SIZE*9 STEP SIZE
140             FOR INCR=0 TO SIDE STEP HTCH
150             GOSUB 280
160             PRINT #1,MOVE$+STR$(ROW)+STR$(CLM+INCR)
170             GOSUB 280
180             PRINT #1,DRW$+STR$(ROW+SIDE)+STR$(CLM+INCR)
190             NEXT INCR
200     NEXT CLM
210 NEXT ROW
220 PRINT
230 GOTO 70
240 '
250 '
260 '
270 '
280 'XON/XOFF subroutine
```

FIG. 5.

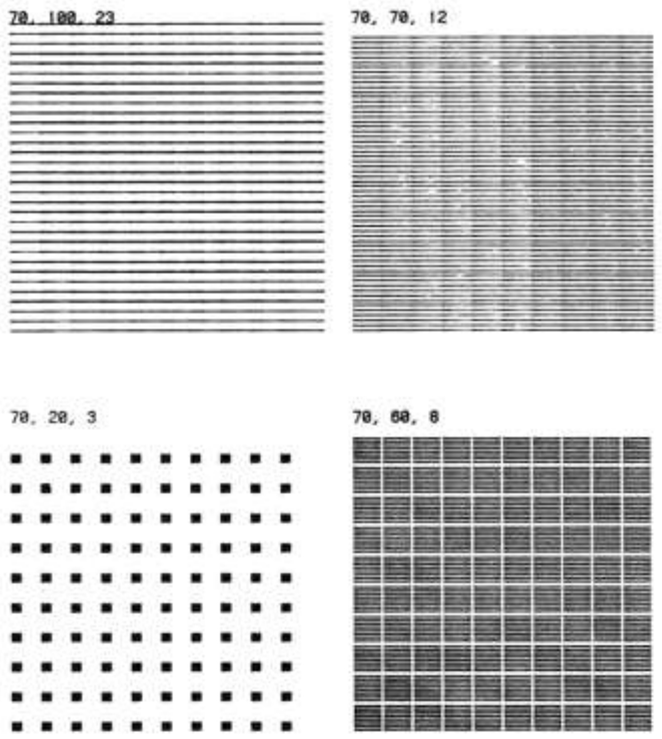
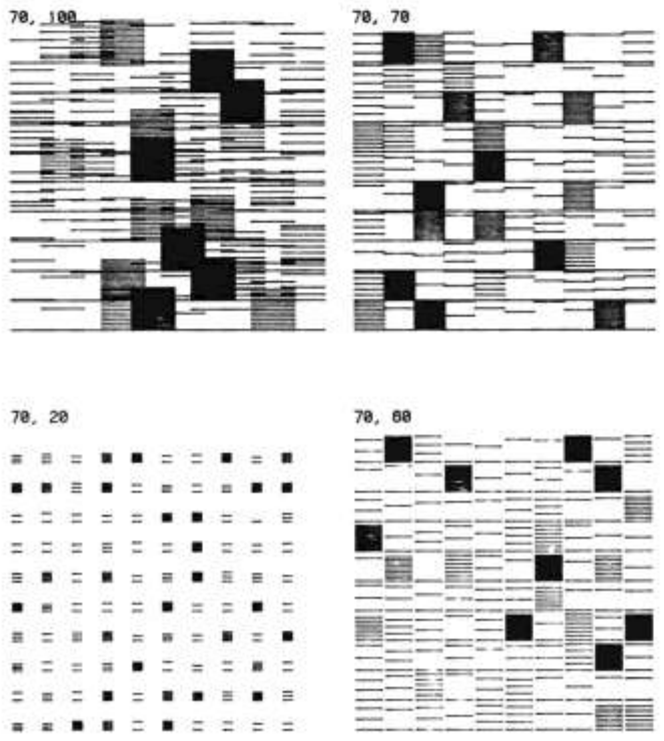
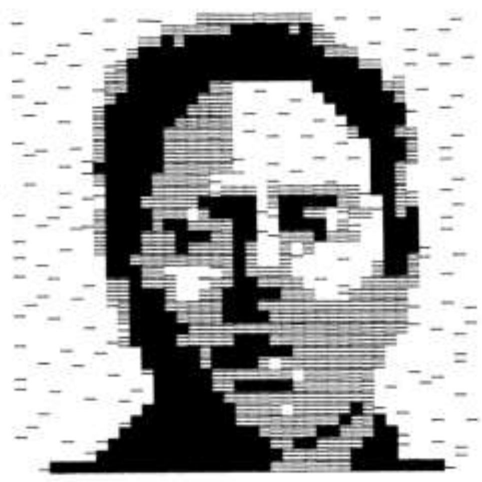


FIG. 6.



Listing HATCHPLOT 1 adds several new lines to the procedure. We ask for INPUT of the distance between the parallel lines of the hatching. This interval gets the variable name HTCH. A new FOR . . . NEXT loop is added inside the other two loops. This loop adds an increment to the vertical, or *y*, axis locations. This increment is called INCR and is simply the SIDE divided by the HTCH variable. The MOVE in line 160 moves the pen carriage, with



the pen up, to the lower left position. The DRW (in line 180) then causes a line to be traced to the lower right corner. This pattern is repeated until the loop reaches the side value, thus completing a hatched square. The program will then proceed to make ninety-nine additional hatched squares. In Fig. 5 the third number at the upper left of each drawing is the hatching interval, or INCR.

As in preceding cases, we can modify this program very easily in a variety of ways. We can add a random number test that will plot a random number of squares. Listing HATCHPLOT 2 multiplies the HTCH value by a random value that is multiplied by the SIDE. Fig. 6 reveals that each square has a different tonal value.

```
10 DEFINT A-Z'    HATCHPLOT2
20 CLS
30 MOVE$="!AX"
40 DRW$="!AY"
50 OPEN "COM1:1200,0,7,1" AS #1
60 PRINT #1,"!AE";
70 INPUT "X origin";XO
80 INPUT "Y origin";YO
90 INPUT "Step size";SIZE
100 INPUT "Side of square";SIDE
110 INPUT "Hatching interval";HTCH
120 FOR ROW=XO TO XO+SIZE*9 STEP SIZE
130     FOR CLM=YO TO YO+SIZE*9 STEP SIZE
140             FOR INCR=0 TO SIDE STEP SIDE/HTCH*(RND*SIDE)
150             GOSUB 280
160             PRINT #1,MOVE$+STR$(ROW)+STR$(CLM+INCR)
170             GOSUB 280
180             PRINT #1,DRW$+STR$(ROW+SIDE)+STR$(CLM+INCR)
190             NEXT INCR
200     NEXT CLM
210 NEXT ROW
220 PRINT
230 GOTO 70
240 '
250 '
260 '
270 '
280 'XON/XOFF subroutine
```

We can continue to expand upon the theme and variations. We have just made a set of squares with different tonal values, and with a little tinkering with several lines of the BASIC program, it is simplicity itself to alter the visual qualities of the image. Our next two variations are produced in the same way as those in the One Hundred Squares program—by modifying the SIDE dimension and supplying a random value to vary the SIDE dimension. We multiply the SIDE value by 2 and then multiply that value by a random number. Each square is, therefore, of a randomly determined size. Line 130 produces this new randomly determined value for SIDE each time the program passes through the loop.

```
10 DEFINT A-Z'    HATCHPLOT3
20 CLS
30 MOVE$="!AX"
40 DRW$="!AY"
50 OPEN "COM1:1200,O,7,1" AS #1
60 PRINT #1,"!AE";
70 INPUT "X origin";XO
80 INPUT "Y origin";YO
90 INPUT "Step size";SIZE
100 INPUT "Side of square";SIDE
110 FOR ROW=XO TO XO+SIZE*9 STEP SIZE
120     FOR CLM=YO TO YO+SIZE*9 STEP SIZE
130     RNDSIDE=SIDE*RND*2
140             FOR INCR=0 TO RNDSIDE STEP (RND*RNDSIDE)+1
150             GOSUB 290
160             PRINT #1,MOVE$+STR$(ROW)+STR$(CLM+INCR)
170             GOSUB 290
180             PRINT #1,DRW$+STR$(ROW+RNDSIDE)+STR$(CLM+INCR)
190             NEXT INCR
200     NEXT CLM
210 NEXT ROW
220 PRINT
230 PRINT
240 GOTO 70
250 '
260 '
270 '
280 '
290 'XON/XOFF subroutine
```

Fig. 7 reveals the results of these simple manipulations. Again, the images we have created are modest and somewhat less than astonishing, but nonetheless we have an indication of the ease involved in transforming the image.

Let us now run the program again with a different set of values. Using a much larger value for the SIDE dimension, multiplied by 6, and a small random value—0.3—to determine whether to plot the boxes, we again obtain another visual variant of our original theme. Fig. 8 (p. 70) shows the results. Because of the large random values generated for the SIDE, some of the boxes overlap. Since our image is composed of a series of parallel lines with different intervals between the hatchings in each box, we begin to see another new effect. The pattern formed by the two overlapping hatchings is called the moire effect. A moire pattern is a third pattern created when two other patterns are superimposed upon one another. Originally the term moire referred to wavy, watery patterns in certain fabrics, but something similar can be seen in a highly linear drawing such as we are making.

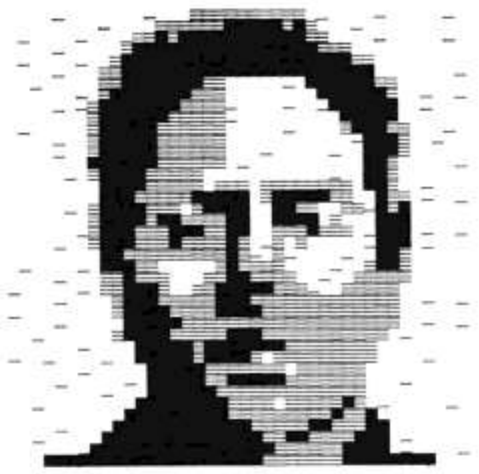
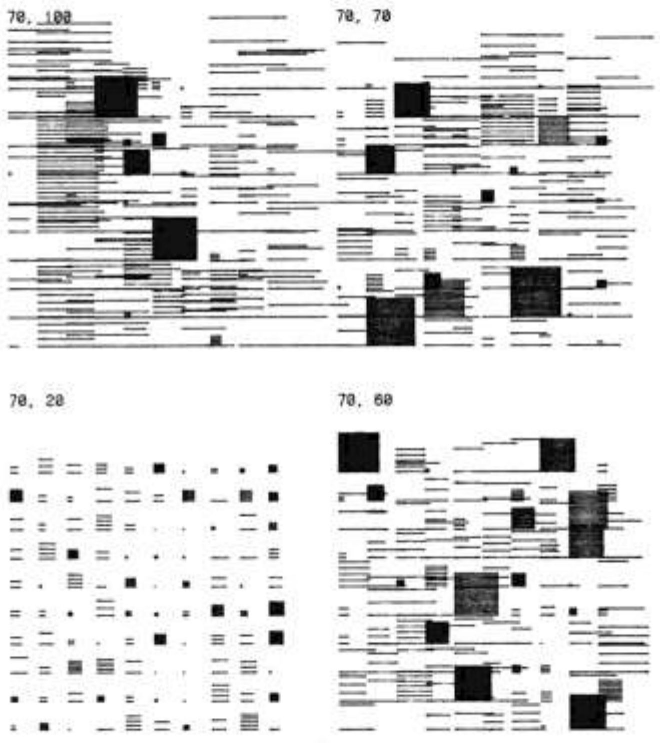


FIG. 7.

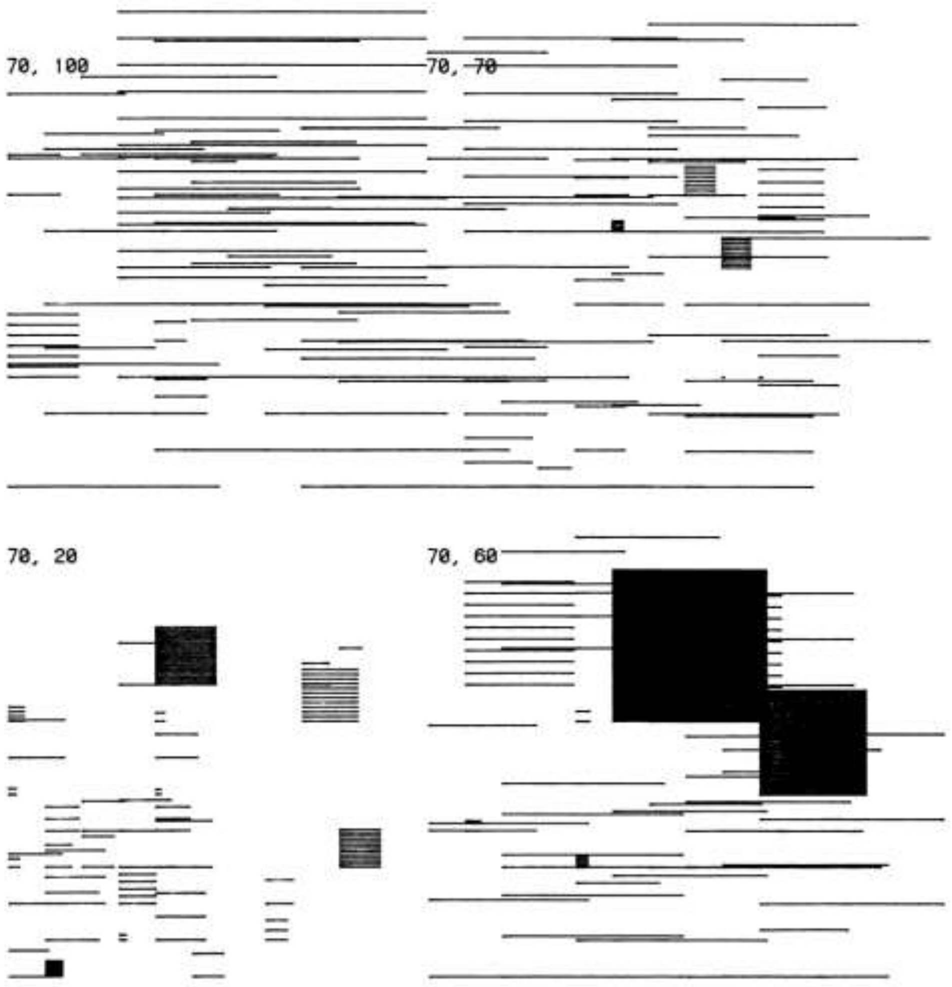


```
10 DEFINT A-Z'    HATCHPLOT4
20 CLS
30 MOVE$="!AX"
40 DRW$="!AY"
50 OPEN "COM1:1200,0,7,1" AS #1
60 PRINT #1,"!AE";
70 INPUT "X origin";XO
80 INPUT "Y origin";YO
90 INPUT "Step size";SIZE
100 INPUT "Side of square";SIDE
110 FOR ROW=XO TO XO+SIZE*9 STEP SIZE
```

(continued on next page)

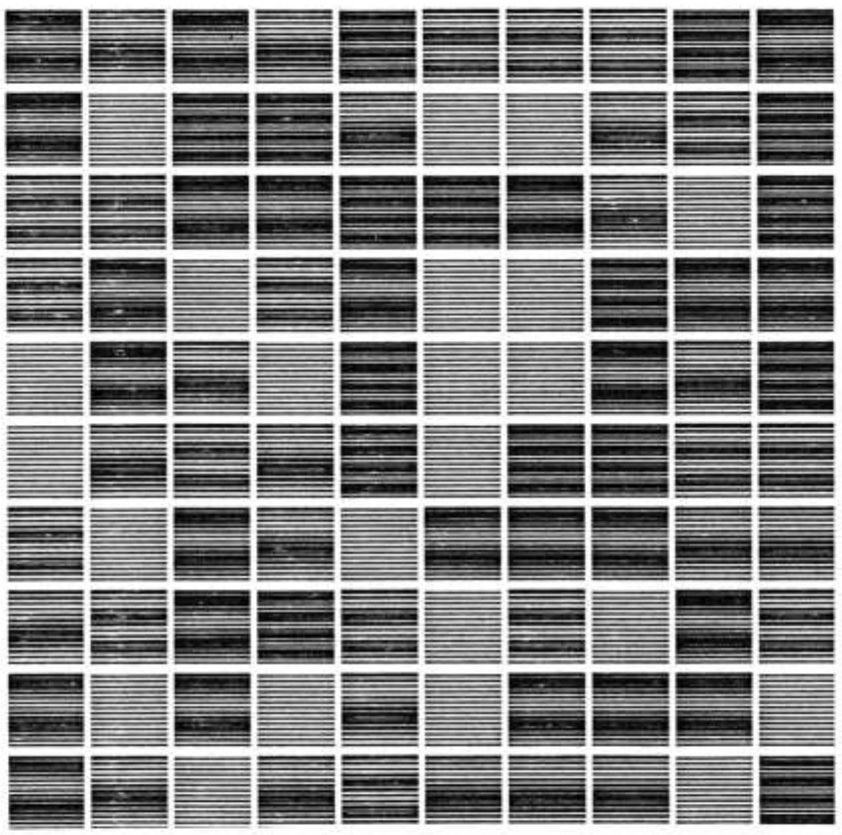
```
120      FOR CLM=YO TO YO+SIZE*9 STEP SIZE
130      RNDSIDE=SIDE*RND*6
140      IF RND>.3 THEN 210
150              FOR INCR=0 TO RNDSIDE STEP (RND*RNDSIDE)+3
160              GOSUB 290
170              PRINT #1,MOVE$+STR$(ROW)+STR$(CLM+INCR)
180              GOSUB 290
190              PRINT #1,DRW$+STR$(ROW+RNDSIDE)+STR$(CLM+INCR)
200              NEXT INCR
210      NEXT CLM
220 NEXT ROW
230 PRINT
240 GOTO 70
250 '
260 '
270 '
280 '
290 'XON/XOFF subroutine
```

FIG. 8.



As an example of exploiting whatever creative effects we may discover in these drawing experiments, we can attempt to use moire patterns. We will go back to our original hatching program. First, we will plot our series of squares, using the same hatching value for each square and the same SIDE dimension. We obtain a very regular matrix of hatched squares. Next, we will plot over these squares another series of smaller squares with regular SIDE values but using randomly chosen values for the hatching. In Fig. 9, the vivid moire effect is seen. Each of the smaller squares with its slightly different hatching value produces, in turn, another moire pattern with the underlying square.

FIG. 9. 140, 125, 9



At this point, you already may have invented your own variations on these themes. Both the One Hundred Squares and Hatchplot programs are very simple. The variations we have used involve elementary alterations of the basic concepts of the program. There are many other possibilities. Instead of using only horizontal lines in the Hatchplot program, we could use vertical hatching or a combination of horizontal and vertical lines. If we were using a plotter with multiple pens, it would be an easy matter to shift between the different pens, using a random-number test in the same way we used a test to plot only part of the time. Or we could alternate between hatched squares and boxes, and so forth.

The possibilities boggle the mind. Contrary to any notion of the computer stifling the creative and aesthetic impulses of the artist, this simple exercise of the squares demonstrates that the possibilities are great. Indeed, the possibilities are vast, and, as when the artist sits staring at the blank canvas or sheet of paper, using the computer is only limited by your imagination. The vast possibilities can even be a liability. Indeed, as the artist Harry Holland has said, "the machine's very flexibility makes it essential to be strictly decisive or else wallow in a sea of alternatives."

INTERACTIVE HATCHPLOT

UP TO THIS POINT we have always made a grid of ten by ten squares, creating one hundred figures in the image. Once again, using a few elementary modifications to the program, the graphic nature of the results can be changed. Instead of being confined to making a ten-by-ten matrix, two INPUT statements will be added that will allow the use of any number of rows or columns.

Our two loops that generate the rows and columns are multiplied by nine, or in our particular case, ten minus one. This is necessary to satisfy the manner in which the loops generate our numbers. We will specify our two variables as NUMROW and NUMCLM, for the number of boxes in each row and the number of boxes in each column. When we substitute these values in the loops, we obtain NUMROW−1 and NUMCLM−1. When we actually plot our hatched square, we can plot rectangular matrices of any dimension at any location on our plotting surface.

```
10 DEFINT A-Z'     HATCHPLOT5
20 CLS
30 MOVE$="!AX"
40 DRW$="!AY"
50 OPEN "COM1:1200,0,7,1" AS #1
60 PRINT #1,"!AE";
70 INPUT "X origin";XO
80 INPUT "Y origin";YO
90 INPUT "Step size";SIZE
100 INPUT "Side of square";SIDE
110 INPUT "Hatching interval";HTCH
120 INPUT "Length of row";NUMROW
130 INPUT "Length of clm";NUMCLM
140 FOR ROW=XO TO XO+SIZE*(NUMROW-1) STEP SIZE
150      FOR CLM=YO TO YO+SIZE*(NUMCLM-1) STEP SIZE
160              FOR INCR=0 TO SIDE STEP HTCH
170              GOSUB 300
180              PRINT #1,MOVE$+STR$(ROW)+STR$(CLM+INCR)
190              GOSUB 300
200              PRINT #1,DRW$+STR$(ROW+SIDE)+STR$(CLM+INCR)
210              NEXT INCR
220      NEXT CLM
```

```
230 NEXT ROW
240 PRINT
250 GOTO 70
260 '
270 '
280 '
290 '
300 'XON/XOFF subroutine
```

If we should want to plot a single hatched square we input values of 1 for both the row and column values. If we wanted to plot a rectangle of 37 rows by 119 columns, that would be fine as well. With each run of the program we must specify the *x* and *y* origins of the rectangle.

In Fig. 10 (p. 74) we see the results of such a series of runs. Eleven rectangular matrices have been plotted on the paper. As the program ran, I chose values for the rectangles and entered them into the machine, and the plotter proceeded to plot them. The following is a sample output from this series of runs of the Interactive Hatchplot program.

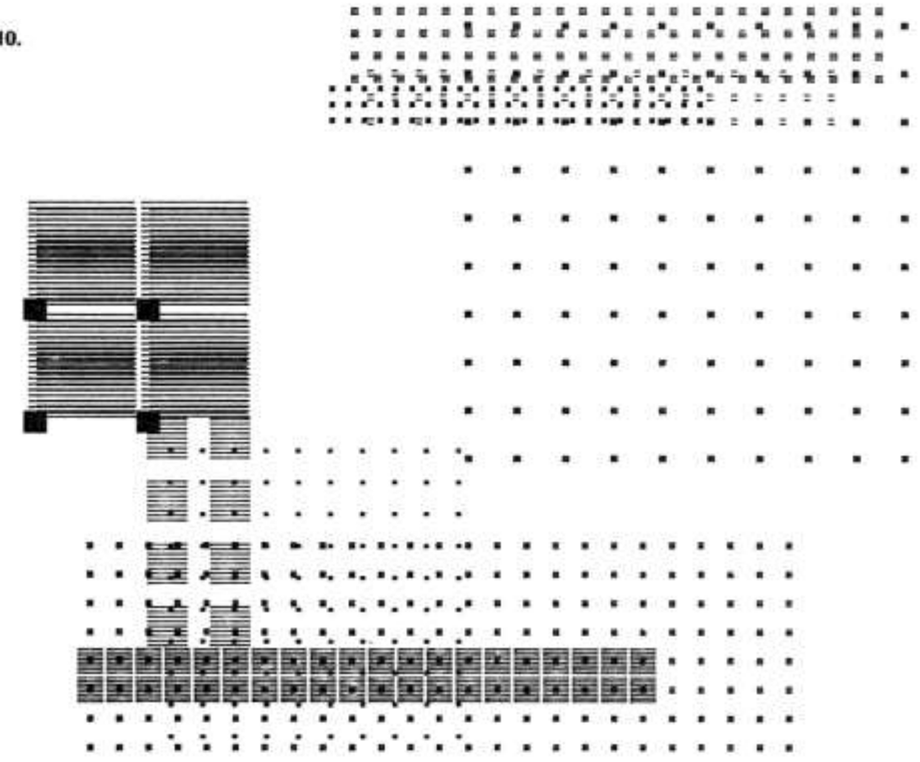
```
X origin? 200              X origin? 387              X origin? 1000
Y origin? 200              Y origin? 125              Y origin? 700
Step size? 60              Step size? 66              Step size? 100
Side of square? 50         Side of square? 8          Side of square? 12
Hatching interval? 7       Hatching interval? 5       Hatching interval? 5
Length of row? 20          Length of row? 10          Length of row? 10
Length of clm? 2           Length of clm? 10          Length of clm? 10

X origin? 220              X origin? 344              X origin? 733
Y origin? 100              Y origin? 317              Y origin? 1400
Step size? 60              Step size? 130             Step size? 50
Side of square? 10         Side of square? 80         Side of square? 12
Hatching interval? 5       Hatching interval? 10      Hatching interval? 12
Length of row? 25          Length of row? 2           Length of row? 20
Length of clm? 8           Length of clm? 4           Length of clm? 3
```

(continued on next page)

```
X origin? 766
Y origin? 1488
Step size? 47
Side of square? 13
Hatching interval? 6
Length of row? 24
Length of clm? 4

X origin? 722
Y origin? 1403
Step size? 33
Side of square? 8
Hatching interval? 4
Length of row? 24
Length of clm? 3

X origin? 100
Y origin? 800
Step size? 233
Side of square? 215
Hatching interval? 12
Length of row? 2
Length of clm? 2
```

```
X origin? 117
Y origin? 792
Step size? 233
Side of square? 203
Hatching interval? 11
Length of row? 2
Length of clm? 2

X origin? 90
Y origin? 760
Step size? 233
Side of square? 44
Hatching interval? 3
Length of row? 2
Length of clm? 2
```

FIG. 10.



What this program and drawing procedure really represent is a crude computer-aided-design (CAD) system. It is totally interactive. This means that it really is capable of doing nothing unless the artist or designer gives the system a series of numbers upon which to act as the program runs.

This drawing program is also a good illustration of the sort of methodology with which the artist and the computer operate. In the previous example of the One Hundred Squares, we set up a series of conditions to which the computer would respond. These conditions are embodied in the program. Once the program is written and run, we can only add a few inputs to the system, such as specifying the size of the square and the spacing between squares. When we have done that the machinery does its work.

The Hatchplot program is somewhat more interactive in its final form. It produces a very simple geometric figure, but we can specify all of the qualities that it will have. It is very important to note several other aspects of this drawing procedure. Each time we wish to plot, we must specify numeric coordinates. This can be a very frustrating experience for the artist. Working with conventional media, you simply draw a box wherever you please. But if we want to use a plotter to put a box at a certain point in the lower left corner of our plotting surface, we must precisely specify the numeric coordinates of that point. This fact has both virtues and vices. In this instance, we are making a very precise geometric picture. Being able to position the lines very accurately is a great advantage. In fact, in the case of the drawing with moire patterns, it would be very difficult to draw the lines accurately by hand. But the other side of the coin is that we may not be able to put our rectangle exactly where we want it. Unless we adopt a tedious method of actually measuring with a ruler, we may not be able to place our rectangle where our artistic sensibilities would have us place it.

A simple solution that partially solves this program is frequently available. Many plotters have the capability of digitizing. This means that the plotter can report the position of its pen. When queried by the computer, the plotter will reply with the *x* and *y* coordinates. Thus, it would be a simple matter to move the pen carriage to the precise point where the image is to be plotted, query the plotter about pen position, and then input those *x* and *y* values to the computer.

6. PAINT SYSTEMS: DRAWING AND PAINTING ON YOUR VIDEO SCREEN

COMPUTER GRAPHICS SYSTEMS, like computers themselves, are a universal image-making tool. They are capable of being used in a great variety of ways. For an artist using such a system, there are many different methodological and aesthetic approaches. In our previous examples we used a mixture of interactive and noninteractive methods to produce an image. We can tell the computer to make One Hundred Squares and then introduce a random quality that will modify the image in a somewhat predictable fashion. But when the program runs we can never precisely predict which boxes will be plotted. Indeed, some artists, such as Harold Cohen, have structured their programs so that the machinery needs absolutely no input or intervention.

At the opposite pole are a whole class of computer programs that have received a great deal of interest and attention. Paint systems or programs, as they have come to be known, are under the total control of the artist. Most paint systems use the video screen as the primary display device. They use a variety of methods to input graphic information to the computer, ranging from the keyboard to digitizer pads, mice, joysticks, or light pens. In all cases, the concept is to smoothly and effectively translate our natural hand-eye-brain reflex into digital information for the display device.

Paint systems are literally electronic canvases. They rely completely on the conventional input of the artist's hand to create an image. Such a description may not sound like much of an improvement over our existing drawing and painting tools. In some sense, they are not an improvement, because, for instance, they cannot take a poor likeness of a face and somehow magically transform the image into a good likeness. Nor will they somehow magically change a dull composition into a lively composition. What paint systems do offer, however, are many techniques that do border on magic.

Consider the question of color. In most cases, when the artist paints or draws using colors, the results will tend to be quite permanent. We can repaint an area or erase a line—correcting or altering a color is not impossible. But in many cases, it is cumbersome and difficult. Almost all paint systems allow an almost immediate change of color of whatever lines or forms exist on the video screen. Not only is it possible to alter the colors immediately, but, depending on the sophistication and amount of memory available in the system, the actual choice of colors can be huge.

Similarly, most paint systems will allow the user to almost instantaneously flood or fill a polygon or area with a color or textured pattern. This ability eliminates the tedium of filling an area by brush.

Another common feature of paint programs is the use of "brushes." Just as the artist can choose from a very large range of conventional physical devices to transfer color to a surface, such as paper or canvas, most paint systems utilize different brushes to impart color to a video screen. The brush and how it leaves a trace on paper or canvas is in some sense, a trivial, mechanical subject. Yet in the history of painting, the manipulation of paint by brush is absolutely central to the visual effect of a picture. Ingres strove for an absolutely smooth polished surface, obliterating any trace of the brush stroke. Frans Hals lustily created a heavy impasto surface with oil paint.

A modern computer paint system will offer a variety of brush effects. Some, like an airbrush, mimic conventional art tools. Some systems allow the user to define his or her own brush. Brushes have been designed that simulate three-dimensional forms. These brushes produce images that appear to cast shadows and possess highlights.

Paint systems are capable of encompassing a vast number of features, but there are several qualities they must have to make them efficient tools. They must have a tactile responsiveness. This means that they must give a very quick response to movements from the input device. Another important issue is the ease of use. We can all pick up a pen and begin drawing, but a paint system will often require us to have some initial training in its use. Naturally, as with most computer activities, this initial period of training and learning is characterized by clumsiness and frustration. To improve the interaction with the system, many paint programs will offer a menu that is displayed on the screen.

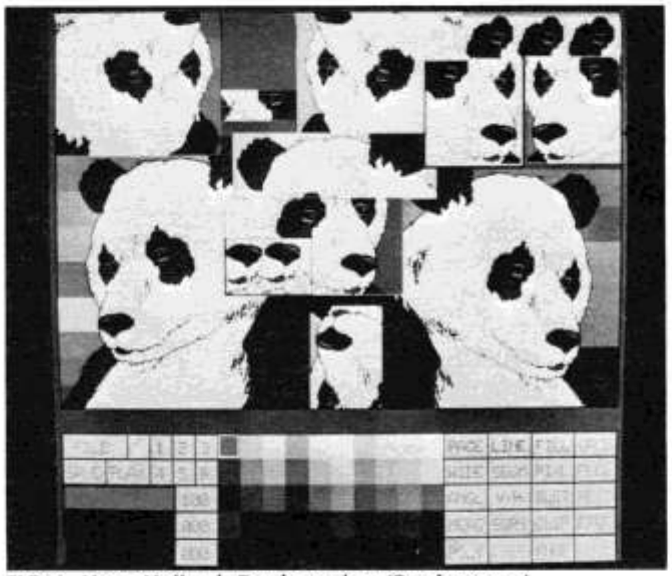


FIG. 1. Harry Holland. *Pandamonium* (Panda screen), 1983 screen photograph. DEC LSI-11 and an AED 512 color display controller. © 1983 Harry Holland.

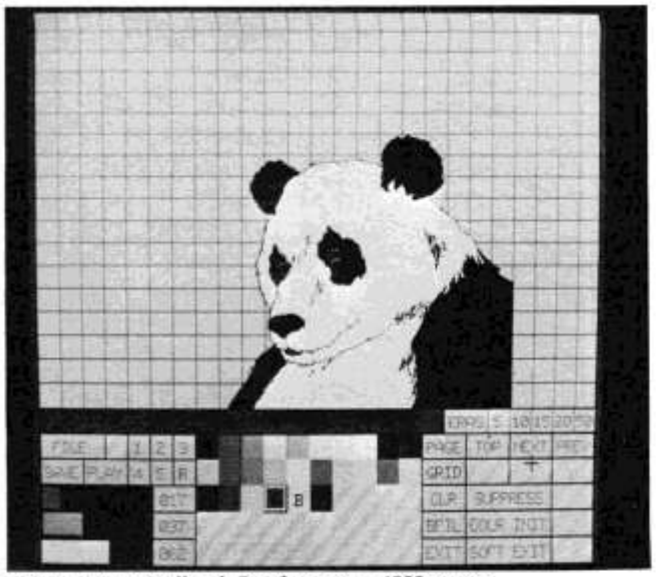


FIG. 2. Harry Holland. Panda screen, 1983 screen photograph. DEC LSI-11 and an AED 512 color display controller. © 1983 Harry Holland.

Paint programs are available for almost all computers that have a graphics capability. From the Atari 400 upward, there is a wide range of choices. The Apple Macintosh computer is notable for including a sophisticated paint package as part of its standard software offering. The MacPaint program uses a mouse as its input device and has many desirable features. Color is not one of them.

A PAINT SAMPLE

PAINT PROGRAMS CAN be very large and complex, but they can also be very simple. At heart they consist of a fundamental procedure:

- Establish cursor coordinates
- Input new cursor coordinates
- Update cursor coordinates
- Display "brush" at current coordinates
- Loop

As a simple demonstration, we can use the keyboard as our input device. Keyboards are certainly not the most desirable or natural input devices for graphics but they do have the virtue of being universally available. Listing Pen1 is written for the IBM Personal Computer, using the standard numeric keypad as the origin of cursor control. The "4" and "6" move the cursor left and right; "8" and "2" move the cursor up and down; and "7", "9", "1", and "3" control diagonal motion. An INKEY statement reads the keyboard and a series of IF . . . THEN statements update the cursor location. The IF . . . THEN statements then branch to an ON BRUSH GOSUB that directs the program to the proper display routine.

```
10 CLS          'PEN1
20 KEY OFF
30 DEFINT A-Z
40 RANDOMIZE(TIMER)
50 X=160:Y=100:C=3
60 Y=100
70 C=3
80 PRINT "Brush 1=point 2=elliptical"
90 PRINT "      3=airbrush 4=rnd lines"
100 INPUT BRUSH
110 CLS:SCREEN 1:COLOR 0,1
120 X$=INKEY$:IF X$="" THEN 120
130 LOCATE 1,1:PRINT ASC(X$)
140 IF X$="4" THEN X=X-1:GOSUB 230
150 IF X$="6" THEN X=X+1:GOSUB 230
160 IF X$="8" THEN Y=Y-1:GOSUB 230
170 IF X$="2" THEN Y=Y+1:GOSUB 230
180 IF X$="7" THEN Y=Y-1:X=X-1:GOSUB 230
190 IF X$="9" THEN Y=Y-1:X=X+1:GOSUB 230
200 IF X$="1" THEN Y=Y+1:X=X-1:GOSUB 230
210 IF X$="3" THEN Y=Y+1:X=X+1:GOSUB 230
220 GOTO 120
230 ON BRUSH GOSUB 250,280,370,420
240 RETURN
250 'point brush
260 PSET(X,Y),C
```

```
270 RETURN
280 'elliptical brush
290 FOR EB=1 TO 6
300 PSET(X+EB,Y)
310 NEXT EB
320 FOR EB=1 TO 4
330 PSET(X+EB+1,Y+1)
340 PSET(X+EB+1,Y-1)
350 NEXT EB
360 RETURN
370 'airbrush
380 FOR AB=1 TO 15
390 PSET(X+(RND*15)+RND*5,Y+(RND*15)+RND*5),C
400 NEXT
410 RETURN
420 'rnd lines
430 LINE(X,Y)-(X+RND*10,Y+RND*10),C
440 RETURN
```

Pen1 has four branches for the brush subroutine. The simplest, POINT BRUSH, places a single pixel at the current location. The ELLIPTICAL BRUSH places a crude ellipse using individual PSET statements. The AIRBRUSH subroutine is somewhat more involved. It has a coarse approximation of an airbrush spray pattern. In this case, our airbrush sprays a somewhat unusual square pattern. The airbrush concept should let you see how the variables involved could be manipulated and expanded upon to produce different graphic patterns. Finally, the last brush, RND LINES, uses the random-length line.

These various sample brushes were used to produce the images that are pictured in Fig. 3. (p. 80). These illustrations are on a large scale, so each pixel has been enlarged. In each case, the path of the brush stroke was identical.

Another important element of a paint program, and, indeed, of any sort of graphics display system, is the ability to create a file from the information on the screen. Almost any type of video display system with graphics capability will have a command that allows the user to query the system to determine the numeric value of a pixel on the screen. For instance, in the IBM Personal Computer, BASIC returns the numeric value associated with the color on the screen, at the *x* and *y* location specified.

```
XX=POINT(X,Y)
```

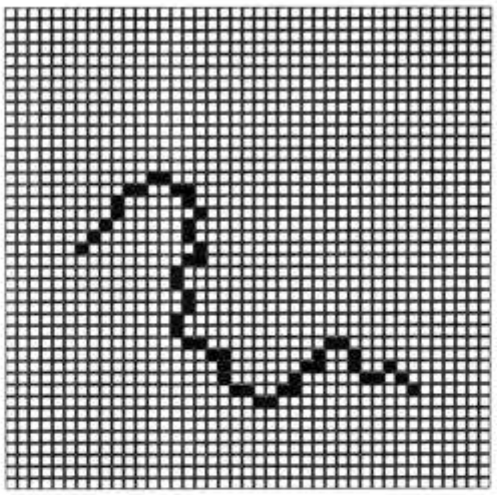


FIG. 3a.

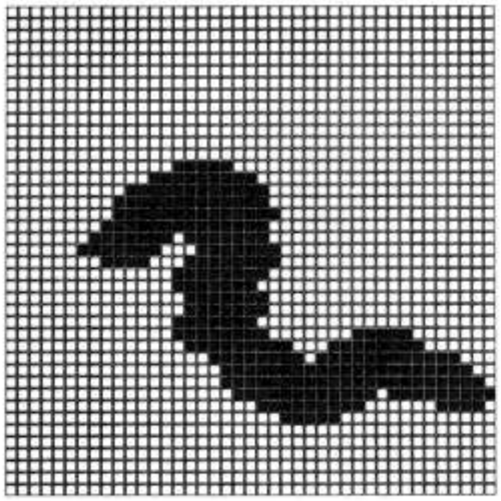


FIG. 3b.

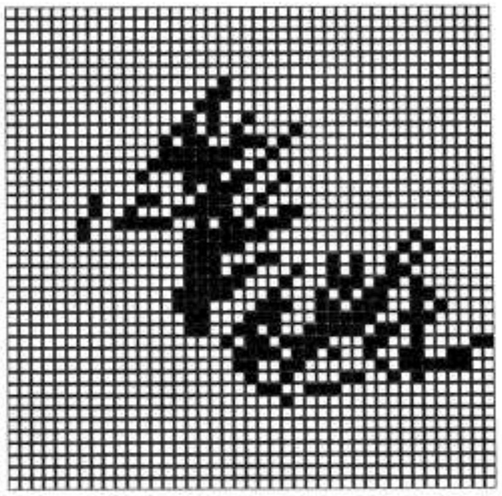


FIG. 3c.

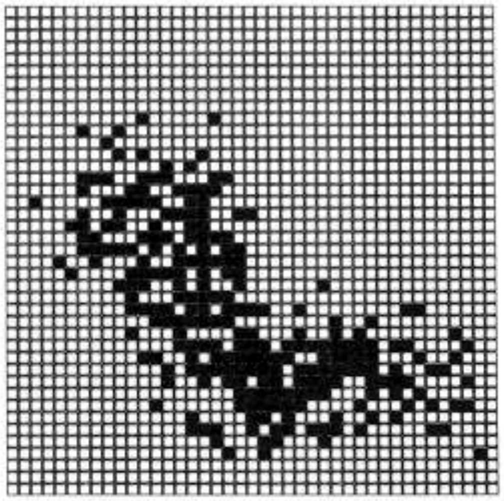


FIG. 3d.

A formal—although not the easiest—way to preserve the graphic information on the screen would be to create a sequential file and read the information with a series of row and column loops. Many BASICs offer a more elegant method. The segment of video memory is saved in a file with a BSAVE statement. Thus,

```
10 DEF SEG=&HB800:BSAVE "PRETTY.PIX",0,&H4000
```

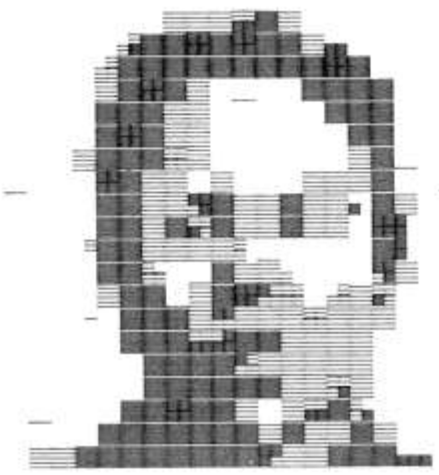
will save 16,000 bytes of video memory in a file. The counterpart of this procedure would use a BLOAD statement to read that file into the frame buffer:

```
10 DEF SEG=&HB800:BLOAD "PRETTY.PIX",0
```

These complementary operations run on the IBM Personal Computer with a 16,000-byte color graphics frame buffer.

DIGITIZERS, MICE, JOYSTICKS, LIGHT PENS

HOLDING A PEN, PENCIL, OR BRUSH in our hands is an old and familiar experience. But making a drawing using a set of keyboard cursor commands is not. The computer graphics industry has created many different hardware options so that computer graphics can be made more comfortably.



The graphics pad or tablet is the oldest attempt to solve the problem of graphic input. It is also the most directly analogous to drawing with pen or pencil on flat surfaces. The pad or tablet—sometimes also known as a digitizer or digitizing pad—is a bit like a plotter in reverse. Instead of moving a pen about a flat surface in response to a stream of coordinates from the host computer, the user moves a stylus over the flat digitizer surface, and the tablet returns a series of coordinate pairs to the computer.

These digitizing pads come in a great variety of sizes and resolutions. Recently very small and inexpensive pads have become available for simple home computing systems. Some of these low-resolution pads respond to the user's finger touch to return the coordinate pairs. At the opposite end of the scale are very large and extremely accurate tablets. Such large tablets or tables, which may be larger than 40" × 60" and have a resolution of 0.001", are used in cartography and industrial CAD applications.

The most common method of transmitting the location of the stylus to the computer utilizes a signal generated by the stylus that is in turn picked up by a finely spaced grid of wires under the tablet. With the proper decoding circuitry, the pad can sense the *x* and *y* coordinates of the stylus and send them to the computer. These coordinate pairs can be transmitted continuously, in what is sometimes called stream mode, or can be communicated upon command. With the proper software in the computer, these coordinates can be stored in a file, displayed on the screen and connected with lines, or manipulated in

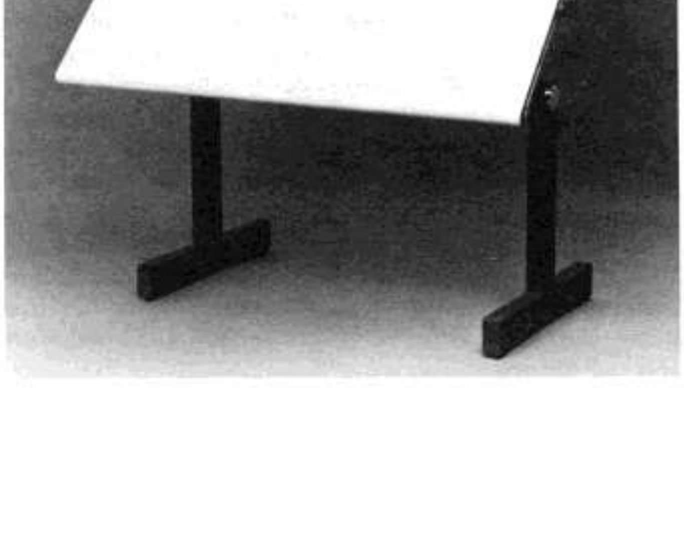


FIG. 4. A 42" x 60" high-resolution digitizing pad. Photograph courtesy of Summagraphics Corp.

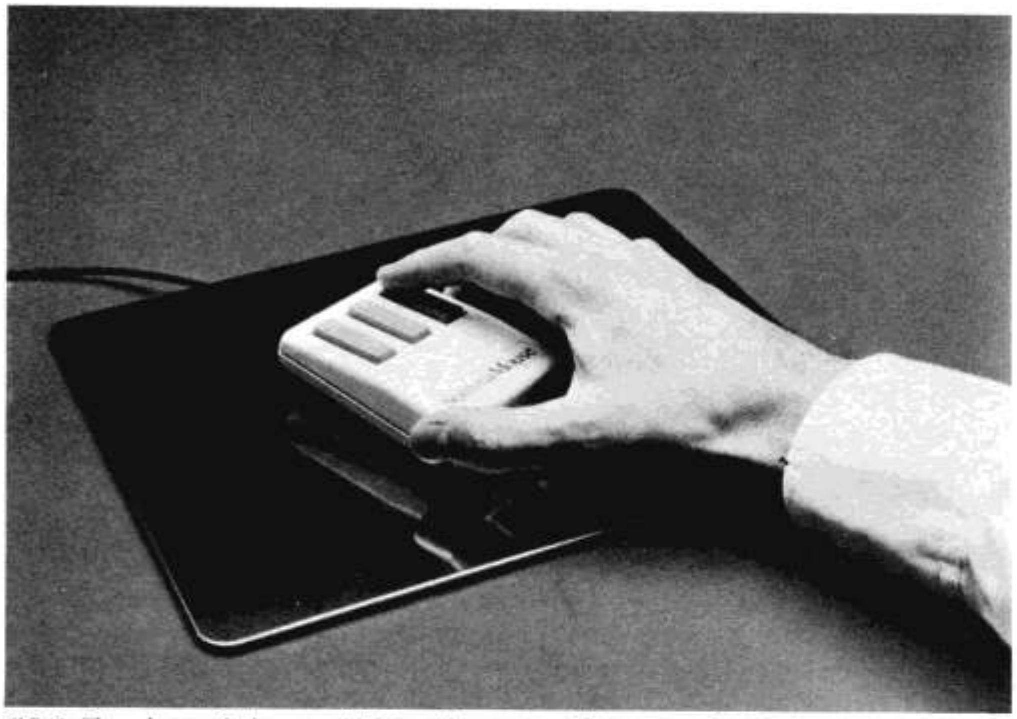


FIG. 5. Three-key optical mouse. Photograph courtesy of Summagraphics Corp.

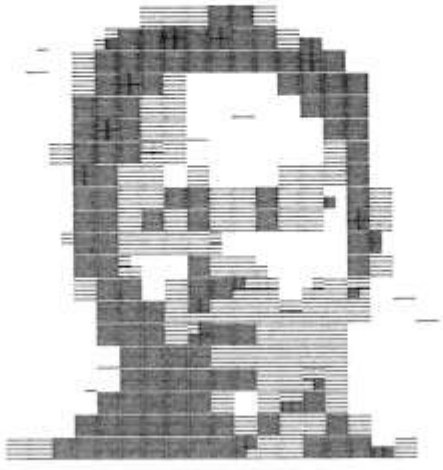
other ways. Some plotters have the ability to act as digitizers and can provide data in the same ways as pads, but require the user to move the cursor—the pen carriage—using keyboard directional controls on the plotter.

Another common, but somewhat less sophisticated, input device is the joystick. Joysticks have been heavily used with computer games to provide cursor control. They can be useful for moving a cursor about the video display. Joysticks contain potentiometers that sense *x* and *y* motion. The potentiometers translate the two analog electrical signals into digital values that are transmitted to the computer. The joystick is thus only capable of signaling indications of relative motion to the computer, unlike the graphics tablet, which can indicate absolute position. Game paddles are another variation on the joystick and have been mostly used with video and computer games.

Although the digital mouse has been used for a number of years with computer systems, recently it has become very popular. Currently mice are used in graphics systems, but they are perhaps even more widely used in business applications, such as word processing and spreadsheet programs. A mouse is a bit like a joystick turned upside down. The mouse consists of a small hand-held box that must be moved about a flat surface. The mouse rides on wheels that produce indications of *x* and *y* motion to the computer. Like the joystick, mice can only indicate relative motion. In business applications they are used to point to menu selections. Typically, a mouse will have a number of buttons that enable the user to input menu selections to the computer.

Joysticks and mice usually indicate motion and position through the use of a cursor on the video screen. Because their relative position is translated into a cursor position, they have lower resolution than a graphics tablet. Another device occasionally used in graphics systems is a light pen. A light pen is pointed at the screen display. The tip of the pen contains a photosensitive element that indicates to the computer whether the screen element pointed to is on or off. Light pens can be used to draw on the screen but more commonly are used for pointing to menu selections. Software is necessary to decode changes in the light pen's position.

Other factors to consider about the use of these types of input devices might include whether software is available for their use, what type of communications are used between the device and computer, and, last and probably most important, whether the device will meet your artistic and stylistic needs.



7. SCREENDUMP: TRANSFERRING A SCREEN IMAGE TO A PLOTTER OR A PRINTER

THE SOMEWHAT INELEGANT term "screendump" simply means to transfer the information on the video screen to some other device. Typically this would mean to a printer or plotter. A screendump of alphanumeric characters that was being transferred to a printer would have to be formated into strings. This is easily accomplished with a series of loops:

```
10 FOR ROW=1 TO 24
20       FOR CLM=1 TO 80
30                S$=CHR$(SCREEN(ROW,CLM))+S$
40       NEXT CLM
50       LPRINT S$
60       S$=""
70 NEXT ROW
```

A screendump to a plotter is a useful procedure that can be performed easily. The main complication is converting the screen coordinates into plotter coordinates.

If the screen has its origin at the lower left corner and the plotter also has its origin at lower left, we can convert the entire screen image into plotter coordinates by multiplying the screen coordinates by a scaling factor. If we were not concerned with the strict proportions of the screen image, we could simply multiply by the maximum dimension of the plotter.

If the width along the *x* axis of our screen was 320 and the dimension, in plotter units, across the *x* axis was 3000, we would divide 3000 / 320 = 9.375. Thus, the scaling factor would be 9.375. We would multiply each coordinate on the screen by this scaling factor. Performing the same operation of the *y* axis coordinates would produce an image that corresponded to the screen image. Similarly, if we multiplied the screen image by 1, we would reproduce the screen image at the same scale as the screen coordinates; which is to say, a very small image would be produced, because the plotter coordinates would typically be in the neighborhood of 200 per inch. On the other hand, if we wanted to magnify our image by a factor of 2, we would multiply the scaling factor by 2.

```
(DEVICECOORD/SCREENCOORD)*SCALEFACTOR
```

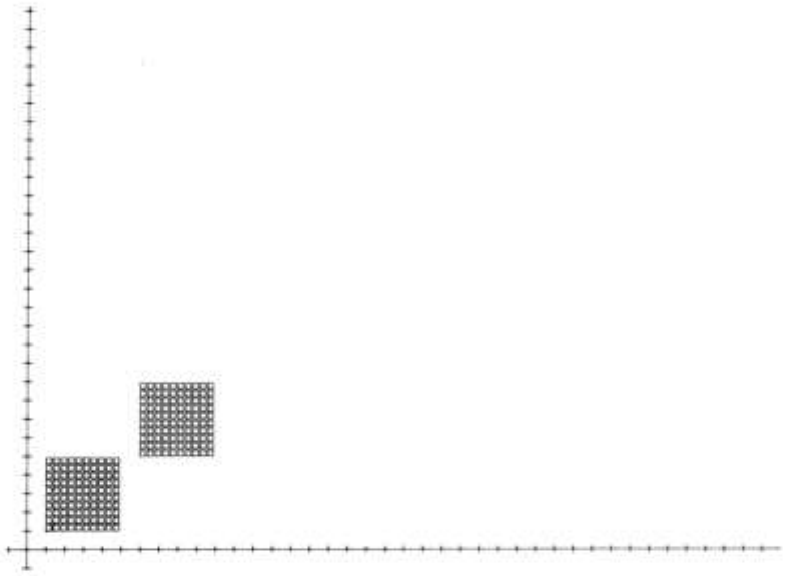


FIG. 1a.

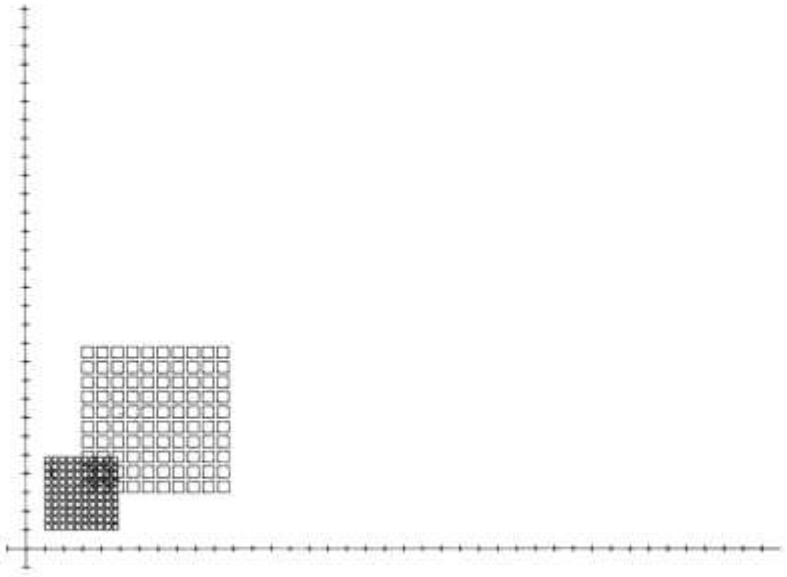


FIG. 1b.

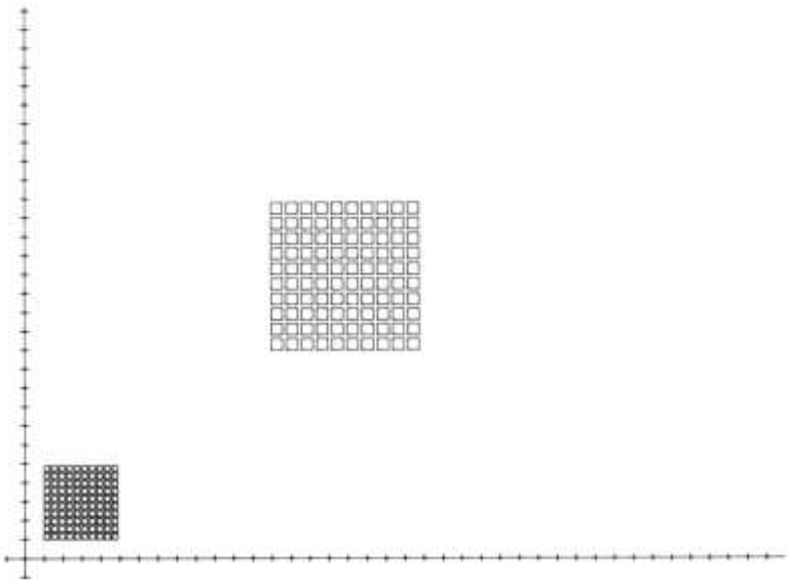
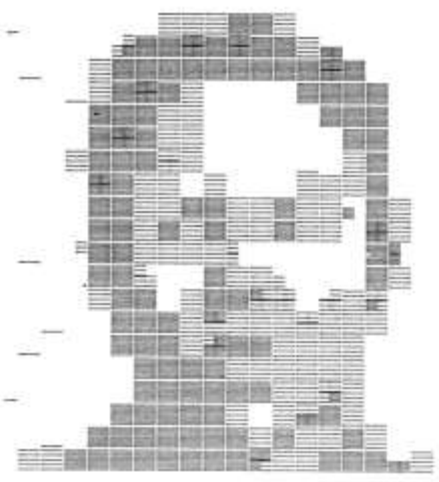


FIG. 1c.



If the screen image represented a figure composed of a series of points connected with lines, it would be a simple matter to command the plotter to connect those coordinate pairs with lines, using MOVE and DRW commands. Also note that when we scale this image, it is always in reference to the origin. In other words, the two origins will always have the same relative location.

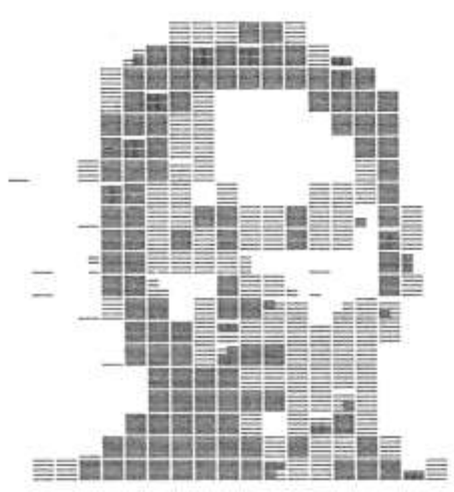
If the origin is to be moved, an appropriate value must be added to—or subtracted from—the *x* and *y* coordinates. This is called translation. Translation moves a figure by adding, or subtracting, a value. Scaling makes a figure larger or smaller by multiplying the coordinates by some value. Translation and scaling are both very important concepts but both are essentially very simple. Fig. 1 shows these operations independently and then combined together.

Using such a screendump program with a plotter permits us to employ an interesting graphics strategy. If we have a color screen display, we can color our drawing by using different color pens to correspond to the screen colors. Of course, the selected pen colors could be whatever we choose and would not have to match the screen colors. Another possibility would be to use different shapes or forms for each pixel to be plotted. A program could offer a menu of different shapes to be plotted. These shapes could be as simple or as complex as you desire. Such a collection of forms is sometimes called a shape table. When the shape table is invoked, the program proceeds to draw some predefined form.

```
10 DEFINT A-Z'    SCREENDUMP
20 CLS
30 MOVE$="!AX"
40 DRW$="!AY"
50 OPEN "COM1:1200,0,7,1" AS #1
60 PRINT #1,"!AE";
70 SCREEN 1
80 INPUT "Picture file name";FILNAM$
90 DEF SEG=&HB800:BLOAD FILNAM$,0
100 GOSUB 630
110 FOR PIXCLR=0 TO 3
120      FOR ROW=1*SCALE TO ((XRIGHT1+1)-XLEFT1)*SCALE STEP SCALE
130           FOR CLM=1*SCALE TO ((YBOT1+1)-YTOP)*SCALE STEP SCALE
140                IF POINT(XLEFT,YBOT)<>PIXCLR THEN 420
150                ON PIXSELC GOTO 160,280,360,360
160                'Outline box
170                GOSUB 520
180                PRINT #1,MOVE$+STR$(ROW+XO)+STR$(CLM+YO)
190                GOSUB 520
200                PRINT #1,DRW$+STR$(ROW+SIDE+XO)+STR$(CLM+YO)
```

(continued on next page)

```
210                GOSUB 520
220                PRINT #1,DRW$+STR$(ROW+SIDE+XO)+STR$(CLM+SIDE+YO)
230                GOSUB 520
240                PRINT #1,DRW$+STR$(ROW+XO)+STR$(CLM+SIDE+YO)
250                GOSUB 520
260                PRINT #1,DRW$+STR$(ROW+XO)+STR$(CLM+YO)+";"
270                GOTO 420
280                'Hatch box
290                FOR INCR=0 TO SIDE STEP 3
300                GOSUB 520
310                PRINT #1,MOVE$+STR$(ROW+XO)+STR$(CLM+YO+INCR)
320                GOSUB 520
330                PRINT #1,DRW$+STR$(ROW+XO+SIDE)+STR$(CLM+YO+INCR)
340                NEXT INCR
350                GOTO 420
360                'Circle
370                GOSUB 520
380                FOR CINCR=CIRTYPE TO SIDE/2 STEP 3
390                PRINT #1,MOVE$+STR$(ROW+XO+SIDE/2)+STR$(CLM+YO+SIDE/2)
400                PRINT #1,"!AAC"+STR$(CINCR)
410                NEXT CINCR
420                YBOT=YBOT-1
430            NEXT CLM
440            XLEFT=XLEFT+1:YBOT=YBOT1
450       NEXT ROW
460 XLEFT=XLEFT1:YBOT=YBOT1
470 GOSUB 710
480 NEXT PIXCLR
490 CLOSE
500 END
510 '
520 'XON/XOFF subroutine
.
.
.
.
.
630 LOCATE 1,1:INPUT "Xleft";XLEFT1:XLEFT=XLEFT1
640 LOCATE 1,1:INPUT "Xright";XRIGHT1:XRIGHT=XRIGHT1
650 LOCATE 1,1:INPUT "Ytop";YTOP1:YTOP=YTOP1
660 LOCATE 1,1:INPUT "Ybot";YBOT1:YBOT=YBOT1
670 LOCATE 1,1:INPUT "X origin";XO
680 LOCATE 1,1:INPUT "Y origin";YO
690 LOCATE 1,1:INPUT "Scaling factor";SCALE:SCALE=(4096/320)*SCALE
700 LOCATE 1,1:INPUT "Side";SIDE
710 LOCATE 1,1:INPUT "Shape 1=Outline box 2=Hatch box 3=Circle 4=Dot";PIXSELC
720 IF PIXSELC=3 THEN CIRTYPE=SIDE/2-3
730 IF PIXSELC=4 THEN CIRTYPE=3
740 RETURN
```



As an example, listing SCREENDUMP utilizes such a shape table for plotting. It will take an image that has been generated on an IBM PC with the color graphics adapter. The image is selected from the screen by defining the left and right x axis coordinates, and the top and bottom y axis coordinates. An outer loop sets up four passes through the matrix of the selected screen area. This corresponds to the four colors on the screen. Whenever the program finds the pixel value that equals the current value in the loop, it jumps to the shape-table section.

Returning to our previous examples, the shape table offers a square that is either outlined or filled in, using hatching. The menu selection also offers a circle that is generated by the plotter hardware but whose starting radius and ending radius can be specified.

Such a screendump program is capable of a great deal of visual variety. For instance, several images can be plotted in layers, one over the other. Different size parameters can be specified for the shapes.

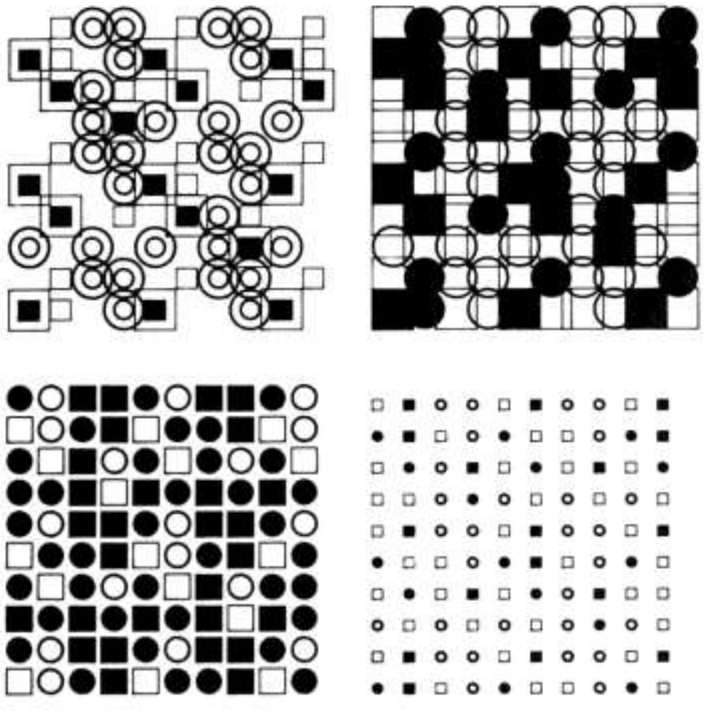


FIG. 2.

8. CIRCLES AND POLYGONS

COMPUTERS can and cannot draw circles. Probably the same can be said for people. We can all draw an approximation of a circle, but it is difficult to draw a good circle without using a mechanical aid. The computer has a similar problem in the sense that it can draw an approximation of a circle, but cannot ultimately draw a true circle. A circle is a polygon with an infinite number of sides. Because the computer is a digital device, it can only approximate a circle by drawing a polygon with a large number of sides.

The proper way to draw a circle involves some simple trigonometry. If we consider the radius of the circle to be swung through an angle of rotation, or *theta*, we can calculate the coordinates of the new point with the following formula:

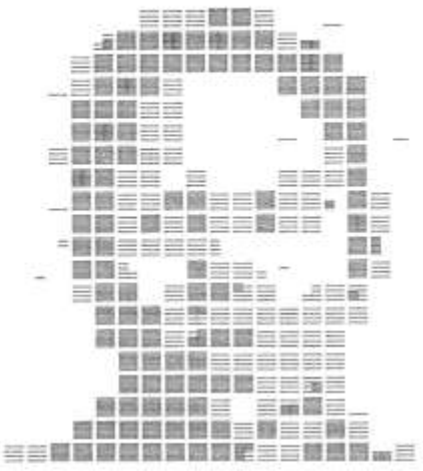
$$x = radius \cdot sin\ (theta)$$
$$y = radius \cdot cos\ (theta)$$

To use a computer to draw a complete circle, we make a loop and determine the number of sides of the polygon with 360/n where *n* is the number of sides. This formula will rotate a point around the origin, but to actually place the circle somewhere other than the origin we must translate it by supplying *x* and *y* coordinates and adding them to the equation.

Listing CIRCLE is a sample program that produces a ten-by-ten array of polygons, using this formula. The program begins with a three-sided polygon—a triangle—and continues to increase the number of polygon sides by one after plotting each column. As you can see, the approximation of a circle becomes much better as the number of sides increases. Roughly speaking, the larger the relative size of the circle, the greater the number of polygon sides necessary to produce a good illusion of a circle.

There are several other points to be made about circle drawing. Instead of drawing a circle, an arc could be made using this formula by simply drawing from the starting angle of the arc to the end angle of the arc instead of looping through the entire 360 degrees. Most microcomputer BASIC languages require the trigonometric functions to be calculated in radians, rather than degrees. Thus, it is necessary to make a conversion by multiplying *theta* (the rotation angle) by *pi* divided by 180 degrees.

```
10 '    CIRCLE
20 CLS
30 PI=3.141593/180
40 P=3
50 MOVE$="!AX"
60 DRW$="!AY"
70 OPEN "COM1:1200,0,7,1" AS #1
```



```
80 PRINT #1,"!AE";
90 INPUT "X origin";XO
100 INPUT "Y origin";YO
110 INPUT "Step size";SIZE
120 INPUT "Radius of polygon";RD
130 FOR ROW=XO TO XO+SIZE*9 STEP SIZE
140       FOR CLM=YO TO YO+SIZE*9 STEP SIZE
150               X=RD*SIN(0*PI)
160               Y=RD*COS(0*PI)
170               PRINT #1,MOVE$+STR$(ROW+X)+STR$(CLM+Y);
180               FOR T=0 TO 360 STEP 360/P
190               GOSUB 340
200               X=RD*SIN(T*PI)
210               Y=RD*COS(T*PI)
220               PRINT #1,DRW$+STR$(ROW+X)+STR$(CLM+Y);
230               NEXT T
240       NEXT CLM
250       P=P+1
260 NEXT ROW
270 P=3
280 PRINT
290 GOTO 90
300 '
310 '
320 '
330 '
340 'XON/XOFF subroutine
```

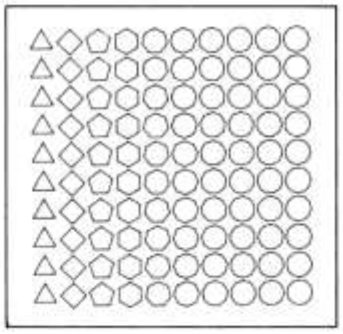


FIG. 1.

Because the computation of trigonometric functions is costly in terms of time, especially with microcomputer BASIC interpreters, the generation of circles can be slow. Since circles are symmetrical, it is possible to speed up calculations by only computing part of the circle and then "mirroring" the coordinates for the other sections. From the user's point of view, the best technique of circle generation is to be able to have the circle generated by either hardware or firmware. Often, circles are available as a graphics primitive in various types of systems.

POLAR COORDINATES

A USEFUL AND INTERESTING variation on the idea of circle drawing is the polar coordinate system, an alternative to the Cartesian coordinate system. Instead of the rectilinear organization of Cartesian coordinates, polar coordinates have one dimension as a function of angular rotation about a point. The other dimension is the distance from that center point. The coordinates of a point would be specified by *theta*—or the angular rotation from 0 degrees—and the number of units from the origin. The direction of rotation can be clockwise or counterclockwise.

As an example of the creative possibilities of using the polar coordinate system, we will combine the Screendump program with a system of polar coordinates. There are a number of things to note in the program. As in the previous Screendump program, there is a shape table that allows a pixel to be drawn as a box, a filled box, or a circle. The dimension of the step between the plotted pixel and the dimension of the shape can be specified. The origin of the plot is to be specified. Likewise, the point at which angular rotation is begun must be indicated. The angular rotation, or the sweep through which the plot will operate, is also included. The width of the boxes and the radii of the circles that are plotted are dependent on the distance from the origin. In other words, small circles will be plotted near the center of the circle, while larger circles would be plotted near the periphery of the polar plot. However, this is a purely arbitrary scheme. It could be just the reverse.

```
10 'SCREENDUMP POLAR
20 CLS:PRINT "SCREENDUMP POLAR"
30 BNK$="                                               "
40 FOR D=1 TO 1000:NEXT D
50 CLS
60 DEFINT A-Y
70 ZPI=3.141593/180
80 OPEN "COM1:1200,O,7,1" AS #1:PRINT #1,"!AE":M$="!AX":D$="!AY"
90 INPUT "Plot step";ST:INPUT "Plot square";ST1
100 PRINT "1=Circle":PRINT "2=Box fill":PRINT "3=Outline box"
110 INPUT BXX1
120 IF BXX1>3 OR BXX1<1 THEN 100
130 IF BXX1=2 OR BXX1=3 THEN 220
140 GOSUB 160
150 GOTO 220
160 LOCATE 1,1:INPUT "Start radius";RA
170 LOCATE 1,1:INPUT "Final radius";SC
180 LOCATE 1,1:INPUT "Step";VC
190 LOCATE 1,1:INPUT "Arc smoothness(1.0=course)";ARSM$
200 IF ARSM$="" THEN PRINT #1,"!ABA"+".5" ELSE PRINT #1,"!ABA"+ARSM$
210 RETURN
220 '
230 GOSUB 970
240 LOCATE 1,1:INPUT "Screen X left";XL:LOCATE 1,1:INPUT "Screen X right";XR
250 LOCATE 1,1:INPUT "Y screen top";YT:LOCATE 1,1:INPUT "Y screen bottom";YB
260 LOCATE 1,1:PRINT BNK$:LOCATE 1,1:INPUT "Radius";RD:RD2=RD
270 LOCATE 1,1:PRINT BNK$:LOCATE 1,1:INPUT "Center plot(x axis)";CPX
280 LOCATE 1,1:PRINT BNK$:LOCATE 1,1:INPUT "Center plot(y axis)";CPY
290 LOCATE 1,1:PRINT BNK$:LOCATE 1,1:INPUT "Color start(0=black)";COLSTRT
300 LOCATE 1,1:PRINT BNK$:LOCATE 1,1:INPUT "Theta start(0 to 360)";THOC
310 LOCATE 1,1:PRINT BNK$:LOCATE 1,1:INPUT "Angular rotation";ART
320 ZTHR1=ART/(XR-XL)
330 ZTHR2=ZTHR1*(ST1/ST)
340 LINE(XL-1,YT-1)-(XR+1,YB+1),3,B
350 LOCATE 1,1:PRINT "Enter CONT to plot":STOP:FOR K1=COLSTRT TO 3:THR=THOC
360 FOR J=XL TO XR
370 RD1=RD:DY=1
380 FOR K=YT TO YB
```

```
390 Z=POINT(J,K)
400 LOCATE 1,22:PRINT Z:LOCATE 1,30:PRINT J;K
410 IF Z=K1 THEN 420 ELSE 800
420 ON BXX1 GOTO 690,440,560
430 '
440 'Box fill
450 '
460 GOSUB 860
470 PRINT #1,M$+STR$(CPX+RD1*SIN(THR*ZPI))+STR$(CPY+RD1*COS(THR*ZPI));
480 FOR BX1=RD1 TO RD1-ST1 STEP -2.5
490 GOSUB 860
500 PRINT #1,D$+STR$(CPX+BX1*SIN(THR*ZPI))+STR$(CPY+BX1*COS(THR*ZPI))
510 GOSUB 860
520 PRINT #1,D$+STR$(CPX+BX1*SIN((THR+ZTHR2)*ZPI))+STR$(CPY+BX1*COS((THR+Z
THR2)*ZPI))+";";
530 NEXT BX1
540 GOTO 780
550 '
560 'Outline Box
570 '
580 GOSUB 860
590 PRINT #1,M$+STR$(CPX+RD1*SIN(THR*ZPI))+STR$(CPY+RD1*COS(THR*ZPI));
600 GOSUB 860
610 PRINT #1,D$+STR$(CPX+(RD1-ST1)*SIN(THR*ZPI))+STR$(CPY+(RD1-ST1)*COS(TH
R*ZPI));
620 GOSUB 860
630 PRINT #1,D$+STR$(CPX+(RD1-ST1)*SIN((THR+ZTHR2)*ZPI))+STR$(CPY+(RD1-ST1
)*COS((THR+ZTHR2)*ZPI));
640 GOSUB 860
650 PRINT #1,D$+STR$(CPX+RD1*SIN((THR+ZTHR2)*ZPI))+STR$(CPY+RD1*COS((THR+Z
THR2)*ZPI));
660 GOSUB 860
670 PRINT #1,D$+STR$(CPX+RD1*SIN(THR*ZPI))+STR$(CPY+RD1*COS(THR*ZPI))+";";
680 GOTO 780
690 '
700 'Circle
710 '
720 FOR I=RA TO SC STEP VC
730 GOSUB 860
740 PRINT #1,M$+STR$(CPX+(RD1-(ST1/2))*SIN((THR+ZTHR2/2)*ZPI))+STR$(CPY+(R
D1-(ST1/2))*COS((THR+ZTHR2/2)*ZPI));
```

(continued on next page)

```
750 GOSUB 860
760 PRINT #1,"!AAC"+STR$(I*(RD1/RD2))+";";
770 NEXT:GOTO 780
780 IF DY=1 THEN 790 ELSE 800
790 BEEP:DY=0
800 RD1=RD1-ST
810 NEXT:THR=THR+ZTHR1:NEXT:LOCATE 1,1:INPUT "1=Circle 2=Box fill 3=Outlin
e box";BXX1:IF BXX1=1 THEN GOSUB 160
820 STOP:NEXT
830 GOTO 90
840 CLOSE #1'close #1 file
850 END
860 'XON/XOFF subroutine
```

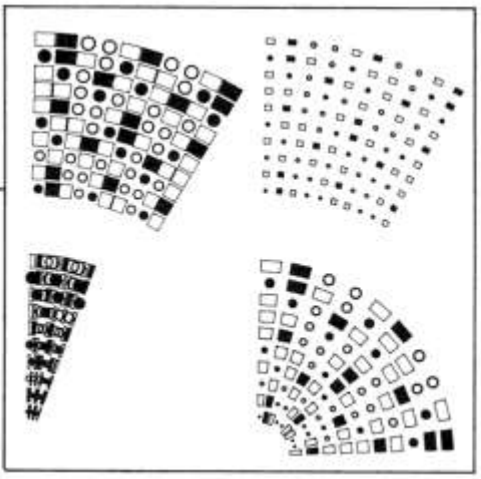
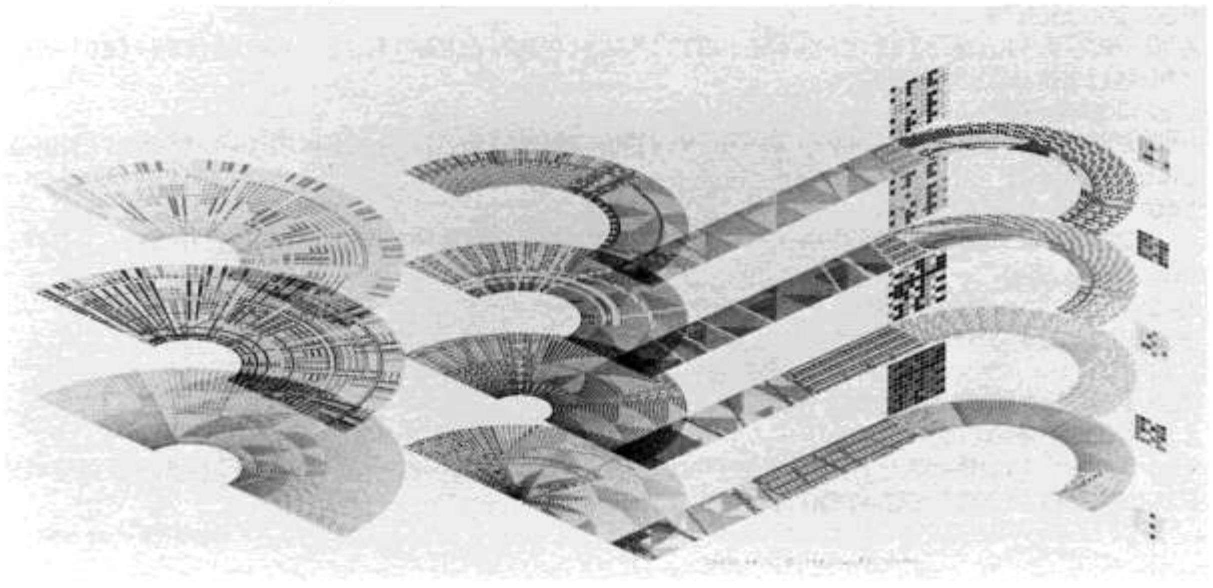


FIG. 2.

Using exactly the same ten-by-ten array of pixels as was used in the Screendump program, we now proceed to plot them with a polar coordinate system. Fig. 2 shows four such plots in which different values were used for the step size and the dimension of the shape. The two designs on the left both have the same origin at the lower left, but the top pattern is rotated through an angle of 30 degrees while the lower drawing is rotated through 15 degrees. On the right, the origins are also the same, but different dimensions are used as well as large amounts of rotation.

While the conversion of conventional Cartesian coordinates into polar coordinates requires extra effort, the results produce very curious effects. It is also possible to apply additional graphic transformations to the polar coordinates. For instance, the coordinates can be scaled along either the *x* or *y* axis, creating an elliptical figure. D'Arcy Thompson, a British biologist, wrote the classic *On Growth and Form* in 1917. In that work, Thompson discusses the relationship of the morphology of biological form to mathematics. Many of his examples describe the transformations of a single archetypal form into a variety of dif-

FIG. 3. Mark Wilson. *Skew AB10,* 1984 plotter drawing on paper, 20" x 38". IBM PC and a Tektronix 4663 plotter. © 1984 Mark Wilson.



fering forms, using a series of different coordinate systems, including several types of polar coordinates.

SKEW AND ROTATE

AS AN EXAMPLE of some of the types of transformations that are easily available by using trigonometric functions, we will consider the skewing and rotating of an image. The term transformation simply means the mathematical manipulation of a graphic image in some manner, such as scaling, translating, rotating, and so forth.

To skew an image means to add some angular value to the coordinates along the *x* or *y* axis, or both. When added to a single axis, the coordinates along the other axis are unaltered. The equation for skewing is:

$$x = y \cdot \tan(t)$$
$$y = x \cdot \tan(t)$$

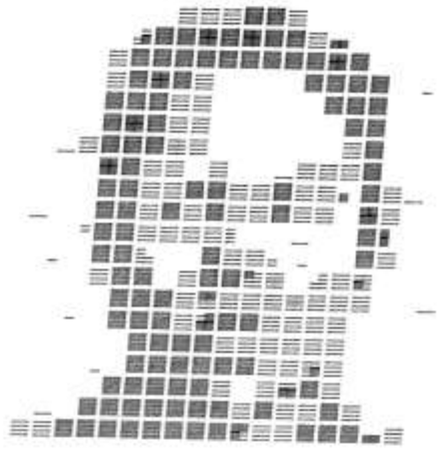
The listings SKEWX and SKEWY perform skewing on the One Hundred Squares program. The program has been modified somewhat from its original state. In this case the skew is performed on the ROW and CLM values and is then added to the *x* and *y* origin value. If this had not been done, the angular skew value would have been added to the figure plus the distance from the origin.

```
10            'SKEWX
20 CLS
30 MOVE$="!AX"
40 DRW$="!AY"
50 OPEN "COM1:1200,0,7,1" AS #1
60 PRINT #1,"!AE";
70 INPUT "X origin";XO
80 INPUT "Y origin";YO
90 INPUT "Step size";SIZE
100 INPUT "Side of square";SIDE
110 INPUT "Rotation";T
120 T=T*(3.141593/180)
130 FOR ROW=.1 TO SIZE*10 STEP SIZE
140       FOR CLM=.1 TO SIZE*10 STEP SIZE
150             GOSUB 360
160             PRINT #1,MOVE$+STR$((CLM*TAN(T))+XO+ROW)+STR$(CLM+YO)
170             GOSUB 360
180             PRINT #1,DRW$+STR$((CLM*TAN(T))+XO+SIDE+ROW)+STR$(CLM+YO)
190             GOSUB 360
200             PRINT #1,DRW$+STR$(((CLM+SIDE)*TAN(T))+XO+SIDE+ROW)+STR$(CL
M+SIDE+YO)
210             GOSUB 360
```

(continued on next page)

```
220             PRINT #1,DRW$+STR$(((CLM+SIDE)*TAN(T))+XO+ROW)+STR$(CLM+SID
E+YO)
230             GOSUB 360
240             PRINT #1,DRW$+STR$((CLM*TAN(T))+XO+ROW)+STR$(CLM+YO)+";"
250       NEXT CLM
260 NEXT ROW
300 PRINT
310 GOTO 70
320 '
330 '
340 '
350 '
360 'XON/XOFF subroutine
```

```
10            'SKEWY
20 CLS
30 MOVE$="!AX"
40 DRW$="!AY"
50 OPEN "COM1:1200,O,7,1" AS #1
60 PRINT #1,"!AE";
70 INPUT "X origin";XO
80 INPUT "Y origin";YO
90 INPUT "Step size";SIZE
100 INPUT "Side of square";SIDE
110 INPUT "Rotation";T
120 T=T*(3.141593/180)
130 FOR ROW=.1 TO SIZE*10 STEP SIZE
140      FOR CLM=.1 TO SIZE*10 STEP SIZE
150             GOSUB 360
160             PRINT #1,MOVE$+STR$(XO+ROW)+STR$((ROW*TAN(T))+CLM+YO)
170             GOSUB 360
180             PRINT #1,DRW$+STR$(XO+SIDE+ROW)+STR$(((ROW+SIDE)*TAN(T))+C
LM+YO)
190             GOSUB 360
200             PRINT #1,DRW$+STR$(XO+SIDE+ROW)+STR$(((ROW+SIDE)*TAN(T))+C
LM+SIDE+YO)
210             GOSUB 360
220             PRINT #1,DRW$+STR$(XO+ROW)+STR$((ROW*TAN(T))+CLM+SIDE+YO)
230             GOSUB 360
240             PRINT #1,DRW$+STR$(XO+ROW)+STR$((ROW*TAN(T))+CLM+YO)+";"
250      NEXT CLM
260 NEXT ROW
300 PRINT
310 GOTO 70
320 '
330 '
340 '
350 '
360 'XON/XOFF subroutine
```



Figs. 4 and 5 are plotted with two images. The unaltered squares image is plotted from the same origin along with the same image skewed by an angle of 30 degrees. Note that the coordinates of the unskewed axis are unaltered.

Finally, both the *x* and *y* axes can be skewed simultaneously. Listing SKEWXY—Fig. 6—has both the normal image and the *x* and *y* axes skewed at 30 degrees.

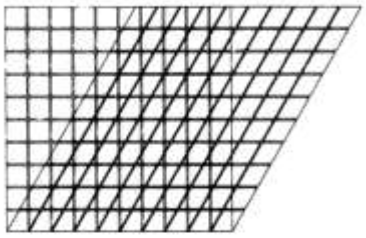


FIG. 4.

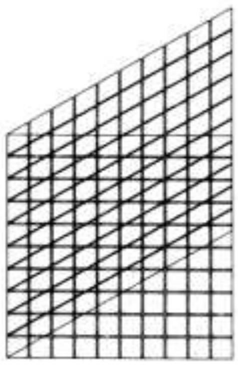


FIG. 5.

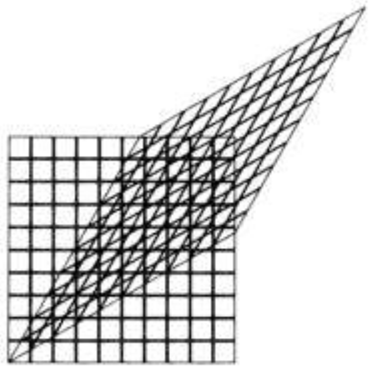


FIG. 6.

```
10          'SKEWXY
20 CLS
30 MOVE$="!AX"
40 DRW$="!AY"
50 OPEN "COM1:1200,O,7,1" AS #1
60 PRINT #1,"!AE";
70 INPUT "X origin";XO
80 INPUT "Y origin";YO
90 INPUT "Step size";SIZE
100 INPUT "Side of square";SIDE
110 INPUT "X skew";XT
111 INPUT "Y skew";T
120 T=T*(3.141593/180)
121 XT=XT*(3.141593/180)
130 FOR ROW=.1 TO SIZE*10 STEP SIZE
140      FOR CLM=.1 TO SIZE*10 STEP SIZE
150           GOSUB 360
160           PRINT #1,MOVE$+STR$((CLM*TAN(XT))+XO+ROW)+STR$((ROW*TAN(T)
)+CLM+YO)
```

(continued on next page)

```
170              GOSUB 360
180              PRINT #1,DRW$+STR$((CLM*TAN(XT))+XO+SIDE+ROW)+STR$(((ROW+S
IDE)*TAN(T))+CLM+YO)
190              GOSUB 360
200              PRINT #1,DRW$+STR$(((CLM+SIDE)*TAN(XT))+XO+SIDE+ROW)+STR$(
((ROW+SIDE)*TAN(T))+CLM+SIDE+YO)
210              GOSUB 360
220              PRINT #1,DRW$+STR$(((CLM+SIDE)*TAN(XT))+XO+ROW)+STR$((ROW*
TAN(T))+CLM+SIDE+YO)
230              GOSUB 360
240              PRINT #1,DRW$+STR$((CLM*TAN(XT))+XO+ROW)+STR$((ROW*TAN(T))
+CLM+YO)+";"
250       NEXT CLM
260 NEXT ROW
300 PRINT
310 GOTO 70
320 '
330 '
340 '
350 '
360 'XON/XOFF subroutine
```

ROTATION

ROTATION is a more commonly used procedure than skewing and involves similar principles. The trigonometry is somewhat more complex, but essentially simple. The equations are:

$$x' = x \cdot \cos(t) - y \cdot \sin(t)$$
$$y' = x \cdot \sin(t) + y \cdot \cos(t)$$

These equations will rotate points about the origin. As in all previous examples, it is necessary to add the translation value to this equation. If this is not done, the rotation will be made about a radius equal to the translation dimension. As usual, our illustration, Fig. 7, is of the hundred squares. The whole image is rotated about the origin. The origin is the lower left corner. The ROW and CLM values are, in essence, the radius arms. The figure has been rotated 30 degrees.

```
10             'ROT100SQ
20 CLS
30 MOVE$="!AX"
40 DRW$="!AY"
50 OPEN "COM1:1200,0,7,1" AS #1
60 PRINT #1,"!AE";
70 INPUT "X origin";XO
80 INPUT "Y origin";YO
90 INPUT "Step size";SIZE
100 INPUT "Side of square";SIDE
110 INPUT "Rotation";T
120 T=T*(3.141593/180)
130 FOR ROW=.1 TO SIZE*10 STEP SIZE
140       FOR CLM=.1 TO SIZE*10 STEP SIZE
```

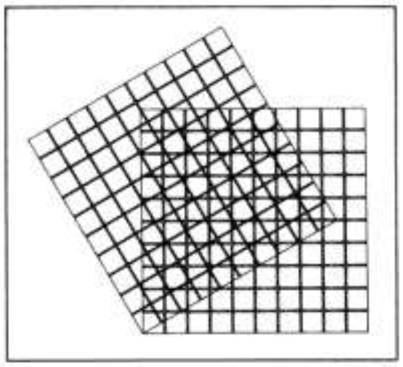


FIG. 7.

```
150           GOSUB 360
160           PRINT #1,MOVE$+STR$((ROW*COS(T)-CLM*SIN(T))+XO)+STR$((ROW*SI
N(T)+CLM*COS(T))+YO)
170           GOSUB 360
180           PRINT #1,DRW$+STR$(((ROW+SIDE)*COS(T)-CLM*SIN(T))+XO)+STR$((
(ROW+SIDE)*SIN(T)+CLM*COS(T))+YO)
190           GOSUB 360
200           PRINT #1,DRW$+STR$(((ROW+SIDE)*COS(T)-(CLM+SIDE)*SIN(T))+XO)
+STR$(((ROW+SIDE)*SIN(T)+(CLM+SIDE)*COS(T))+YO)
210           GOSUB 360
220           PRINT #1,DRW$+STR$((ROW*COS(T)-(CLM+SIDE)*SIN(T))+XO)+STR$((
ROW*SIN(T)+(CLM+SIDE)*COS(T))+YO)
230           GOSUB 360
240           PRINT #1,DRW$+STR$((ROW*COS(T)-CLM*SIN(T))+XO)+STR$((ROW*SIN
(T)+CLM*COS(T))+YO)+";"
250       NEXT CLM
260 NEXT ROW
300 PRINT
310 GOTO 70
320 '
330 '
340 '
350 '
360 'XON/XOFF subroutine
```

The listing is fairly straightforward, except that the lines that do the actual work of rotation become fairly long. Note that the dimensions XO and YO are the translation values. These values are added to the rotation values after the rotation has been performed.

THE TWO-DIMENSIONAL TRANSFORMATION

THE VARIOUS MANIPULATIONS we have performed comprise the two-dimensional transformation. Scaling, translating, rotating, and sometimes skewing enable you to take any point, or collection of points, and move them about the two-dimensional surface of the screen or paper. All of these graphics actions can be performed in various combinations. It is very important to note the sequence in which these actions are performed. As an

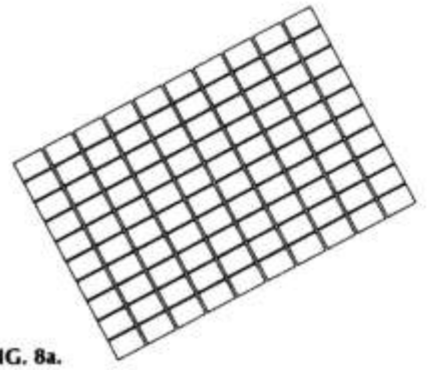


FIG. 8a.

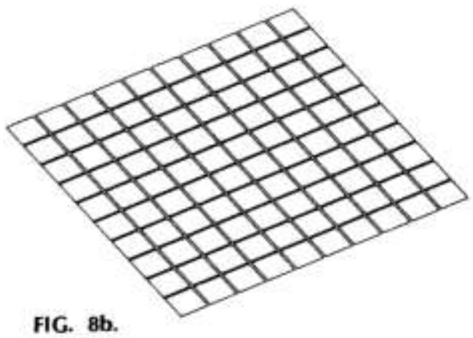


FIG. 8b.

illustration we will take our hundred squares and first do a scaling operation and then rotate the resulting figure. Then we will reverse the order. The rotation listing is modified by simply adding a line that scales the ROW, CLM, and SIDE along the *x* axis. The *y* axis values are not scaled.

Changes in the transformation sequence can produce very divergent results. It is always important to carefully consider the order of any transformation, because the resulting image can be very different in its visual appearance.

THE DEGENERATE LINE

MOST PAINT PROGRAMS feature some type of variable brush. As the cursor is moved across the screen, the software will respond with a variety of different effects. As we have seen, these graphic traces can be very elementary, such as a trail of pixels that correspond to the simple trace we leave when a line is drawn with a pencil or pen. At the other extreme, paint programs can leave all manner of elaborate traces—airbrushings, or whatever the artist or programmer might choose to supply to the computer.

A similar strategy can be employed in drawing with the computer when using a program that generates its own figure. In the case of the One Hundred Squares program, we gave the computer a series of parameters, and the machine then proceeded to produce a figure. However, we might wish to modify the program with some sort of line-drawing procedure. This is easily done. A single square consists of four lines, each line moving in a different direction. The first line moves in a positive *x* axis direction with no change in the *y* coordinate, the second moves in a positive *y* axis direction, and so forth. Therefore we need to provide four different subroutines to accommodate these relative motions.

Each of these subroutines will then subdivide the line segment into a number of intermediate line segments. When this is done, we can then apply some value to the line. As a foil to the precise, geometric lines that have predominated in these figures, we will add a random value. This random value is added with a BASIC function. At each subsegment a new randomly generated value will be added to the *x* or *y* coordinate value—dependent, of course, on the relative motion of the line.

```
10 DEFINT A-Z'      TEXSQ
20 CLS
30 MOVE$="!AX"
40 DRW$="!AY"
50 OPEN "COM1:1200,O,7,1" AS #1
60 PRINT #1,"!AE";
70 INPUT "X origin";XO
80 INPUT "Y origin";YO
```

```
90 INPUT "Step size";SIZE
100 INPUT "Side of square";SIDE
110 INPUT "Line increment";INCR
120 SI=SIDE/INCR
130 INPUT "Random value";RV
140 DEF FNYINCR=RND*RV+1
150 FOR ROW=XO TO XO+SIZE*9 STEP SIZE
160      FOR CLM=YO TO YO+SIZE*9 STEP SIZE
170                GOSUB 360
180                PRINT #1,MOVE$+STR$(ROW)+STR$(CLM)
190                GOSUB 360
200                GOSUB 470
210                GOSUB 360
220                GOSUB 550
230                GOSUB 360
240                GOSUB 630
250                GOSUB 360
260                GOSUB 710
270                PRINT #1,MOVE$+STR$(ROW)+STR$(CLM)+";"
280      NEXT CLM
290 NEXT ROW
300 PRINT
310 GOTO 70
320 '
330 '
340 '
350 '
360 'XON/XOFF subroutine
.
.
.
.
470 'X +
480 '
490 '
500 FOR II=ROW TO ROW+SIDE-SI STEP SI
510      GOSUB 360
520      PRINT #1,DRW$+STR$(II)+STR$(CLM+FNYINCR)
530 NEXT II
540 RETURN
```

(continued on next page)

```
550 'Y +
560 '
570 '
580 FOR II=CLM TO CLM+SIDE-SI STEP SI
590      GOSUB 360
600      PRINT #1,DRW$+STR$(ROW-FNYINCR+SIDE)+STR$(II)
610 NEXT II
620 RETURN
630 'X -
640 '
650 '
660 FOR II=ROW+SIDE TO ROW+SI STEP -SI
670      GOSUB 360
680      PRINT #1,DRW$+STR$(II)+STR$(CLM-FNYINCR+SIDE)
690 NEXT II
700 RETURN
710 'Y -
720 '
730 '
740 FOR II=CLM+SIDE TO CLM STEP -SI
750      GOSUB 360
760      PRINT #1,DRW$+STR$(ROW+FNYINCR)+STR$(II)
770 NEXT II
780 RETURN
```

Fig. 9 uses the same dimensions for the square in each case, but uses different random values. When small values are used, the line successfully mimics the motion of the hand. When the values become larger, the hand becomes somewhat shaky. And when very large values are used the graphic effect becomes quite different again. It should be noted that in all these cases the random value is positive, thus tending to move the line toward the center of the square. But it could just as easily be negative, which would result in the line swelling the square.

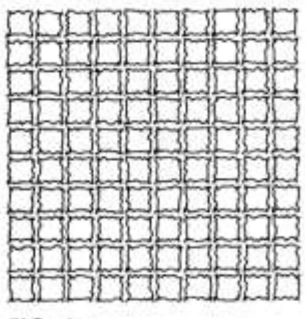


FIG. 9a.

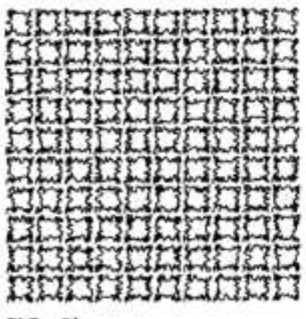


FIG. 9b.

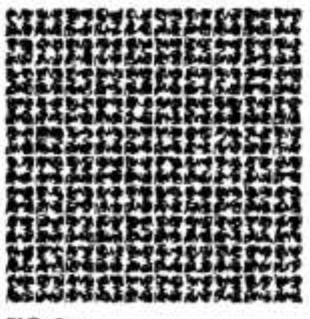


FIG. 9c.

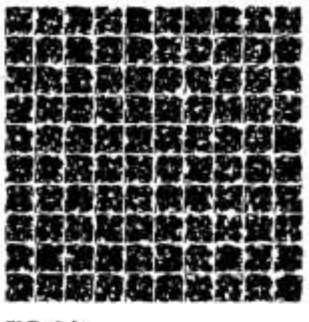


FIG. 9d.

In the same way, this figure can be drawn with a regular pattern. We will use the identical procedure of subdividing the line segment, but the perpendicular increment will be a constant value rather than a random value. After each excursion to this value, the line will return to the original position on the line. Thus, the line will take on a zigzag appearance.

```
.
.
.
450 'X +
460 '
470 '
480 FOR II=ROW TO ROW+SIDE-SI STEP SI
490       GOSUB 340
500       PRINT #1,DRW$+STR$(II)+STR$(CLM+ZV)
510       PRINT #1,DRW$+STR$(II)+STR$(CLM)
520 NEXT II
530 RETURN
540 'Y +
550 '
560 '
570 FOR II=CLM TO CLM+SIDE-SI STEP SI
580       GOSUB 340
590       PRINT #1,DRW$+STR$(ROW-ZV+SIDE)+STR$(II)
600       PRINT #1,DRW$+STR$(ROW+SIDE)+STR$(II)
610 NEXT II
620 RETURN
630 'X -
640 '
650 '
660 FOR II=ROW+SIDE TO ROW+SI STEP -SI
670       GOSUB 340
680       PRINT #1,DRW$+STR$(II)+STR$(CLM-ZV+SIDE)
690       PRINT #1,DRW$+STR$(II)+STR$(CLM+SIDE)
700 NEXT II
710 RETURN
720 'Y -
730 '
740 '
750 FOR II=CLM+SIDE TO CLM+SI STEP -SI
760       GOSUB 340
770       PRINT #1,DRW$+STR$(ROW+ZV)+STR$(II)
780       PRINT #1,DRW$+STR$(ROW)+STR$(II)
790 NEXT II
800 PRINT #1, DRW$+STR$(ROW)+STR$(CLM)+";"
810 RETURN
```

To prevent the zigzag pattern from overlapping at the end of the line segment, the line increments end at the line dimension minus the line increment. This was purely an aesthetic decision rather than a mathematical choice. The line could just as easily have continued zigging and zagging to the end of the segment. Similarly, the line segment terminates at the actual point of the ROW and CLM dimensions, but need not have. In the case of the random value, the line could have terminated when it reached a value equal to or greater than the ROW and CLM dimension. Often, such algorithmic decisions must be made on a purely visual basis.

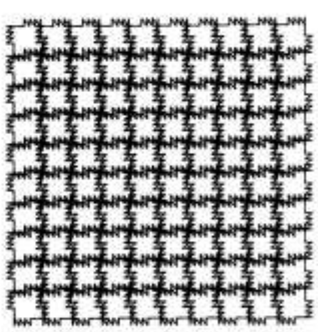


FIG. 10a.

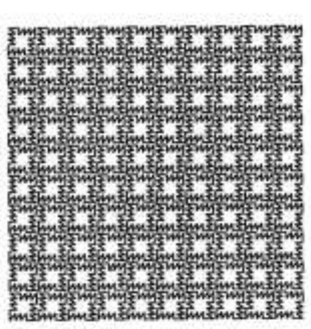


FIG. 10b.

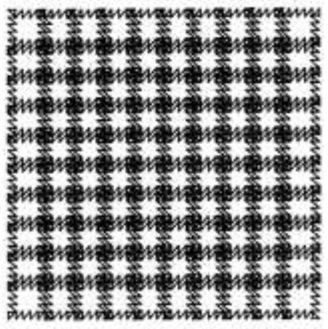


FIG. 10c.

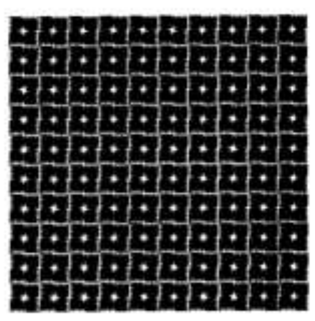


FIG. 10d.

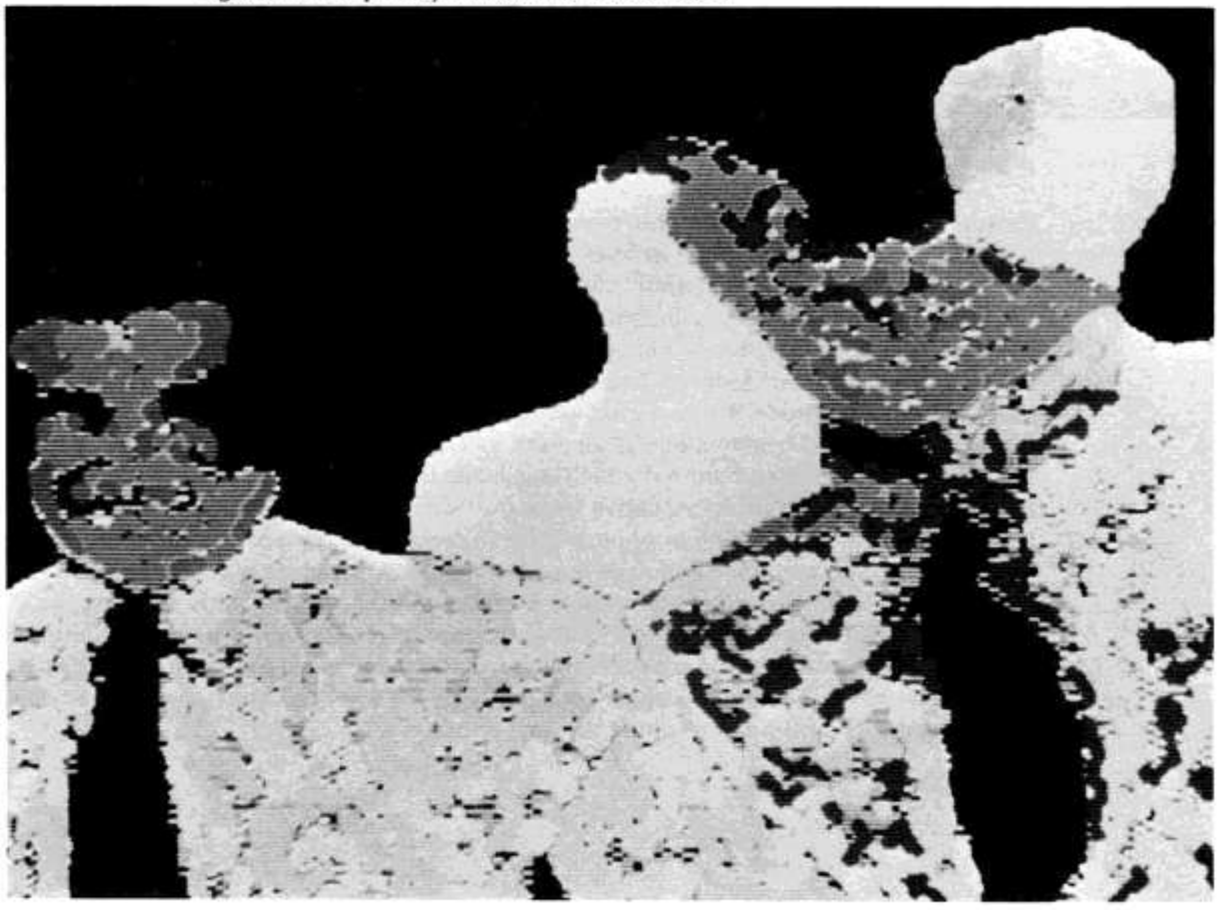
These subroutines only provide for horizontal and vertical motion. If we wanted to include line vectors at oblique angles it would be necessary to include some type of vector generator. Naturally, this would complicate the program, but it is certainly achievable. Some types of display devices include line types that have been implemented in the hardware or firmware. Most commonly, they include dotted and dashed lines, or, in some cases, you can define your own line types.

Creating different line types can provide a useful and interesting tool. While such line types are in some sense a purely decorative device, they can be very important in the creation of an overall aesthetic effect in a picture. Variations in line widths are usually very simple to create, but it is not always simple to join different line segments in a consistent fashion. In the case of the squares, the program could be modified to draw a series of lines parallel to the original box figure.

9. PHOTOGRAPHY

CURRENTLY there is relatively little low-cost equipment available for converting photographic images into digital images. A photographic image must be converted into a series of digital values. In the case of a black-and-white photograph, the tonal values must be converted into a gray scale. A gray scale represents the different tonal values, from light to

FIG. 1. Thomas Porett. 1983 screen image photographed with a Lang camera. Apple with video digitization and paint system. © 1983 Thomas Porett.



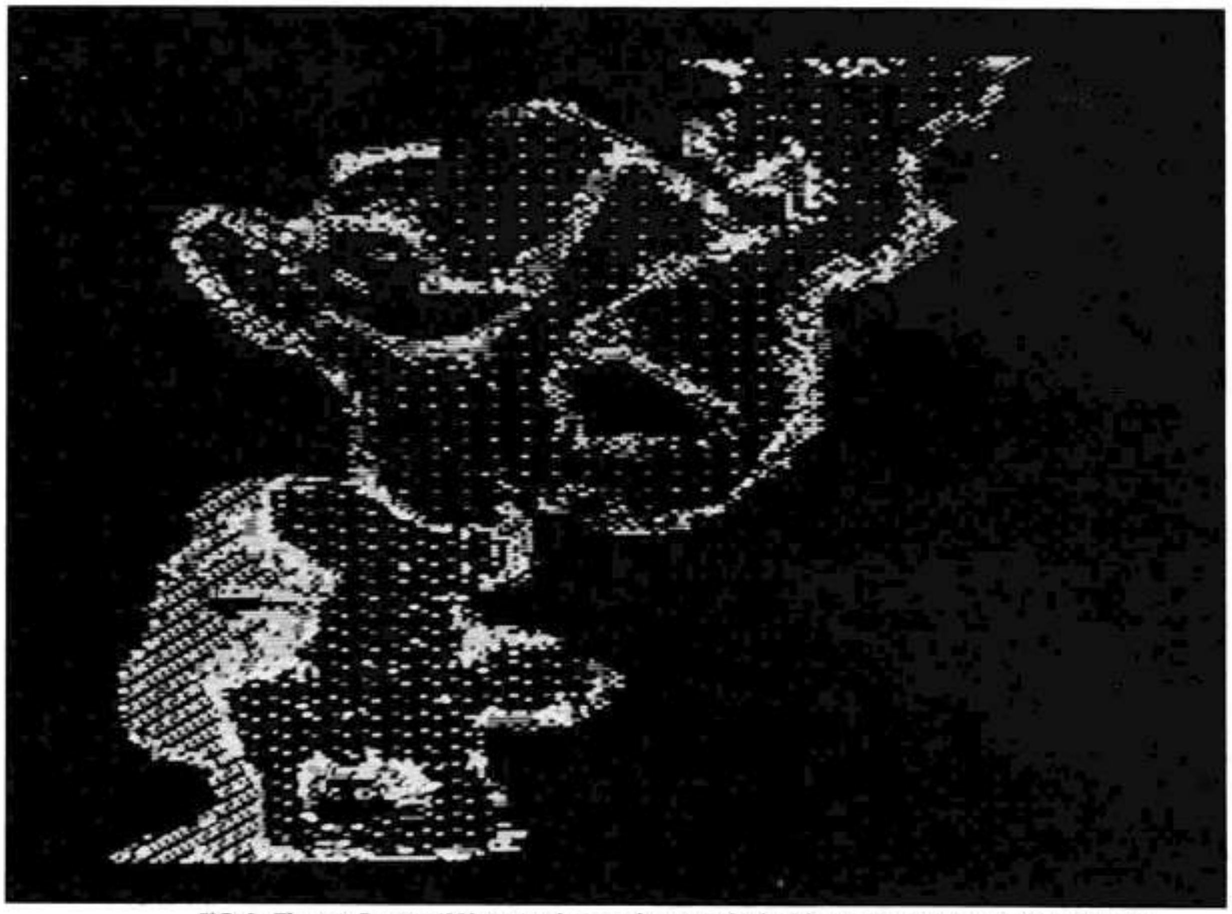


FIG. 2. Thomas Porett. 1983 screen image photographed with a Lang camera. Apple with video digitization and paint system. © 1983 Thomas Porett.

dark, that exist in the photographic or video image. A color image would have to be converted into tonal values as well as hues.

A conventional photographic image, such as a color slide or photographic print, can be scanned by specialized equipment known as a densitometer. Such equipment is expensive. Video images, by virtue of being already encoded electronically, are more potentially useful to the computer. A device known as a frame grabber, or video digitizer, can take a video signal and encode it into digital values. It is also possible to mix the video signals from a computer with the video signals from a camera or a video recording. While such devices are not cheap and are not widely available, the great proliferation of home video recorders may well create an incentive for manufacturers to produce inexpensive devices that will allow the interfacing of photographic material with computers.

Needless to say, once an image has been digitized, it can be manipulated in a vast variety of ways with the computer. Nancy Burson, an artist working with photographs of famous people, has produced a highly amusing series of photographs called *The Composite News.* In one example, she digitized a series of photographs of famous autocrats, Hitler, Mao, and so forth. The digital values from the faces are averaged and a composite facial image is produced from the resulting values.

10. PERSPECTIVE: THREE-DIMENSIONAL GRAPHICS

UP TO THIS POINT we have concentrated on two-dimensional images. Two-dimensional images lack the third dimension of depth. Of course, we are discussing display devices that have only the capability of presenting two dimensions, such as a video screen or printer or plotter. But the illusion of depth can be produced within the confines of a two-dimensional display.

One of the central triumphs of early Western art was the accurate analysis of perspective. Painstaking methodology contributed to an understanding of how to accurately

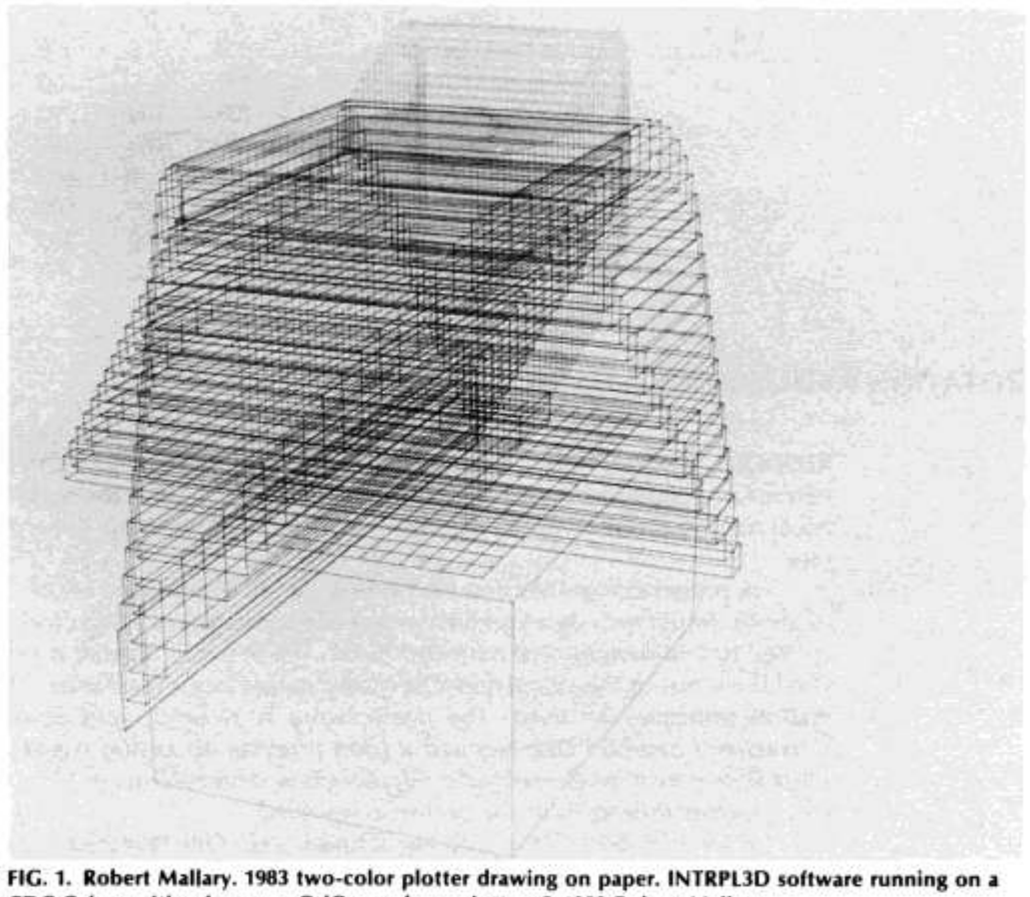


FIG. 1. Robert Mallary. 1983 two-color plotter drawing on paper. INTRPL3D software running on a CDC Cyber, with a four-pen CalComp drum plotter. © 1983 Robert Mallary.

render objects in an illusionistic three-dimensional world. The picture became regarded as a window that the viewer would look through. Today, art students sometimes study these methods using vanishing-point perspective. We are all familiar with the illusion of two parallel lines, such as railroad tracks or roadways, converging at the distant horizon. While vanishing-point perspective is adequate for rendering many objects by hand, more rigorous techniques are necessary when drawing with the computer.

The *x* and *y* axes of the two-dimensional plane are well known to us. To fully describe an object or point in real space, or in three-dimensional space, we must specify both the *x* and *y* coordinates, and a third coordinate of depth. By convention, this depth coordinate is called the *z* coordinate, which lies along the *z* axis. The *z* axis runs perpendicular to the plane formed by the *x* and *y* axes. A point would be specified by three coordinates, *x*, *y*, and *z*. Our square, converted into a three-dimensional solid, a cube, could be described by listing the coordinates of each of the corners, or vertices:

Vertex	X	Y	Z
Lower left corner	0	0	0
Lower right corner	100	0	0
Upper right corner	100	100	0
Upper left corner	0	100	0
Lower rear left corner	0	0	100
Lower rear right corner	100	0	100
Upper rear right corner	100	100	100
Upper rear left corner	0	100	100

ROTATION ABOUT THREE AXES

REDUCED TO THE ESSENTIALS, producing three-dimensional graphic rotations involves two major operations: rotating the point coordinates around three axes and creating the necessary projection of the three-dimensional depth onto a two-dimensional viewing surface.

A proper comprehension of the mathematical problems of the three-dimensional rotations should include an understanding of matrix algebra. Matrix techniques can also be applied to two-dimensional transformations. While matrix algebra is not difficult, it is beyond the scope of this discussion. The reader should consult a standard text on the mathematical principles involved. The classic work is Newman and Sproull's *Principles of Interactive Computer Graphics* and a good practical discussion is given in Artwick's *Applied Concepts in Microcomputer Graphics* (see bibliography, p. 127). We can, nonetheless, proceed with some of the problems involved.

If we think back to the problem of rotating our One Hundred Squares, we remember

that two trigonometric equations were used that calculated the new x and y coordinates as a point was rotated around an axis. If we were to rotate that flat planar figure around the z axis, all of the x and y values would be altered by the rotation but none of the z values would be changed. Such a rotation around the z axis does not change any depth values.

Likewise, if we were to rotate the flat figure of the squares around the x axis, the z depth values and the y axis values would be altered by the x axis values will all remain the same.

If the figure were rotated around the y axis, the x values and z values would change, but the y axis values would remain unchanged.

The three-dimensional rotation employs all three of these rotations sequentially. Each rotation is performed on the results of the previous rotation. As in the two-dimensional transformation, the order of the rotations is important in terms of the final results.

The direction of rotations is also important. The direction is indicated by the sign of the rotation angle. A positive or negative value will rotate the figure in opposite directions about the particular axis.

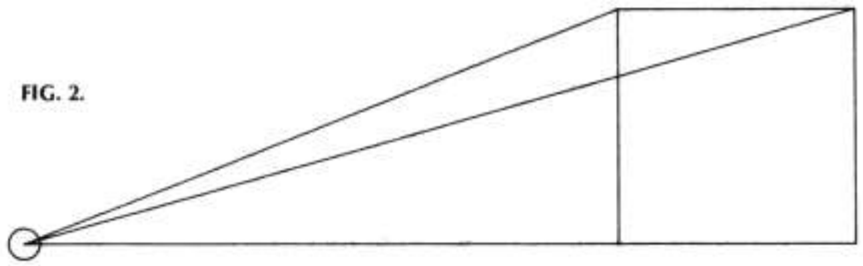
THE PERSPECTIVE PROJECTION

THE THREE AXIS ROTATIONS will produce a geometrically proper rotation of a point in three-dimensional space. If we take this collection of coordinate information and project these points onto a two-dimensional surface—the video screen or plotter paper—the image will correctly show the rotated points but it will not reveal any of the familiar intuitive depth information that we obtain from the real world. In other words, in the real world the tree that is close to our eye spans a large segment of our vision, while the tree on the distant hillside spans only a small segment of our vision. The numeric information that our system has provided about the depth and distance of points is contained in the z axis data. A distant point will have a large z coordinate, while the nearby point will have a small z value.

If we imagine two triangles formed by the eye, with sides being formed by the perpendicular faces of a cube (see Fig. 2 on p. 108), we can see that to correctly project the face of the cube on a display surface requires scaling the x or y coordinates by the ratio formed by the viewing distance divided by the viewing distance plus the z depth information. While this is a somewhat simplified solution to the problems of projection, it nonetheless represents the essential geometry of the problem.

The listing 3D is a subroutine that provides the necessary equations to rotate the flat figure about the three axes. The rotations are arbitrarily in an x, y, and z sequence. Frequently the three axes are referred to by another set of names that correspond to the motions of an aircraft. Pitch corresponds to x axis rotation, heading corresponds to y axis rotation, and bank corresponds to z axis rotation. As in our previous two-dimensional ro-

FIG. 2.



tation program, we must convert angular values into a form the computer can use. This is done early in the program to cut down on the amount of computation. These angular values are:

$$Rx' = Rx \cdot (3.141593/180)$$
$$Ry' = Ry \cdot (3.141593/180)$$
$$Rz' = Rz \cdot (3.141593/180)$$

The actual coordinate values are generated by the One Hundred Squares program and are simply stored in a file. To simplify matters the figure is assumed to be a flat plane without any depth. Thus, the *z* value—as stored in the file—is 1. The coordinates are generated from the lower left corner, the origin, which has the value 0,0,0. When the figure is rotated about any of the three axes, the rotation is about this origin. We could alter this point of rotation simply by adding translation values to the coordinates, thereby shifting the rotational origin to another point. This translation should be performed before rotation.

```
10          '3D
20 DIM XX(501),YY(501),ZZ(501)
30 OPEN "b:hunsq" FOR INPUT AS 1
40 IF EOF(1) THEN CLOSE:GOTO 80
50 INPUT #1,XX(I),YY(I),ZZ(I)
60 I=I+1
70 GOTO 40
80 CLS
90 I=0
100 MOVE$="!AX"
110 DRW$="!AY"
120 OPEN "COM1:1200,O,7,1" AS #1
130 PRINT #1,"!AE";
140 INPUT "X origin";XO
150 INPUT "Y origin";YO
160 INPUT "Rotate y";RY
170 INPUT "Rotate x";RX
180 INPUT "Rotate z";RZ
190 INPUT "Z scale";ZSC
200 RZ=RZ*(3.141593/180)
210 RY=RY*(3.141593/180)
220 RX=RX*(3.141593/180)
230 ZO=1
240 FOR ROW=1 TO 10
```

```
250       FOR CLM=1 TO 10
260                 GOSUB 440
270                 PRINT #1,MOVE$+STR$(X)+STR$(Y)
280                 GOSUB 440
290                 PRINT #1,DRW$+STR$(X)+STR$(Y)
300                 GOSUB 440
310                 PRINT #1,DRW$+STR$(X)+STR$(Y)
320                 GOSUB 440
330                 PRINT #1,DRW$+STR$(X)+STR$(Y)
340                 GOSUB 440
350                 PRINT #1,DRW$+STR$(X)+STR$(Y)+";"
360       NEXT CLM
370 NEXT ROW
380 PRINT
390 END
.
.
.
.
440 'Y axis rotation
450 '
460 LOCATE 8,1:PRINT XX(I);YY(I);ZZ(I)
470 '
480 X1=XX(I)*COS(RY)+ZZ(I)*SIN(RY)
490 Y1=YY(I)
500 Z1=-XX(I)*SIN(RY)+ZZ(I)*COS(RY)
510 '
520 LOCATE 9,1:PRINT X1;Y1;Z1
530 '
540 'X axis rotation
550 '
560 '
570 X2=X1
580 Y2=Y1*COS(RX)-Z1*SIN(RX)
590 Z2=Y1*SIN(RX)+Z1*COS(RX)
600 '
610 LOCATE 10,1:PRINT X2;Y2;Z2
620 '
630 'Z axis rotation
640 '
```

(continued on next page)

```
650 '
660 X3=X2*COS(RZ)-Y2*SIN(RZ)
670 Y3=X2*SIN(RZ)+Y2*COS(RZ)
680 Z3=Z2
690 '
700 '
710 LOCATE 11,1:PRINT X3;Y3;Z3
720 '
730 '
740 '
750 X=(X3/(ZO+(Z3*ZSC)))+XO
760 Y=(Y3/(ZO+(Z3*ZSC)))+YO
770 LOCATE 12,1:PRINT X;Y
780 I=I+1
790 RETURN
```

We can now execute several rotations of our familiar hundred squares. Fig. 3 shows the result of rotating the flat figure about the *z* axis by 30 degrees but does not rotate it about either the *x* or *y* axis. An important practical matter to be noted here is the fact that a very small, non-zero, value must be used in the program. This value—for example 0.001—must be used to prevent a "division-by-zero" error when the program executes. Fig. 4 shows the image rotated about the *z* and *y* axes by 30 degrees. Finally, in Fig. 5 it is rotated

FIG. 3.

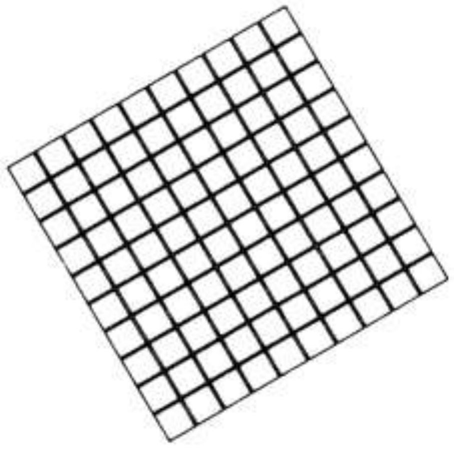
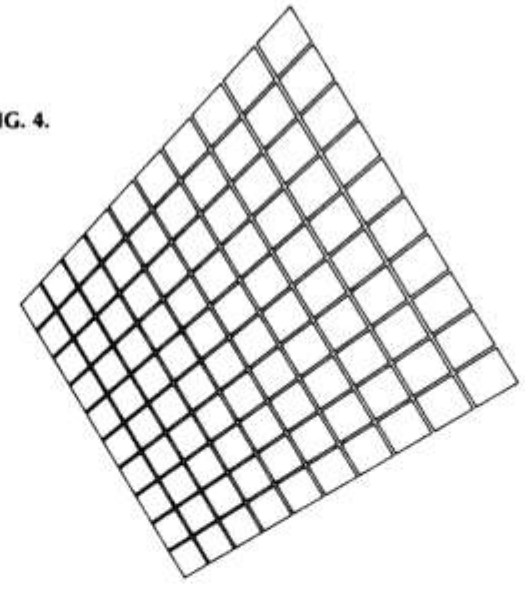


FIG. 4.



about all three axes by 30 degrees. Because the observer's viewpoint is relatively close to the figure, a certain amount of distortion is seen. This is similar to the distortion in a photograph taken with a wide-angle lens. Figs. 6 and 7 show the same image as if from different viewpoints. Fig. 6 shows distortion equivalent to that of a photo taken with a very wide angle lens, while Fig. 7 would be similar to one taken using a telephoto lens. In these pictures the *z* depth values are, respectively, quite large and quite small.

FIG. 5.

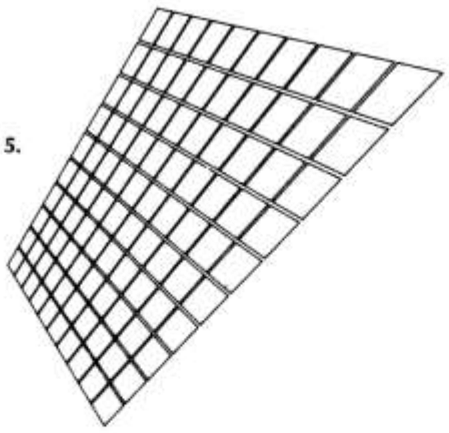


FIG. 6.

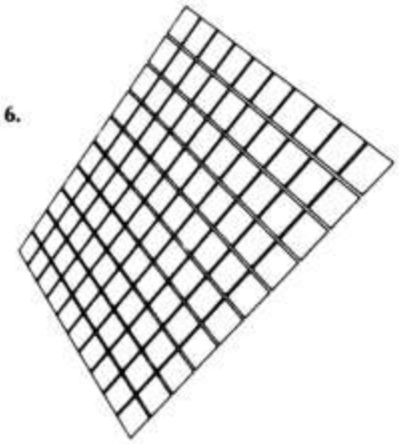
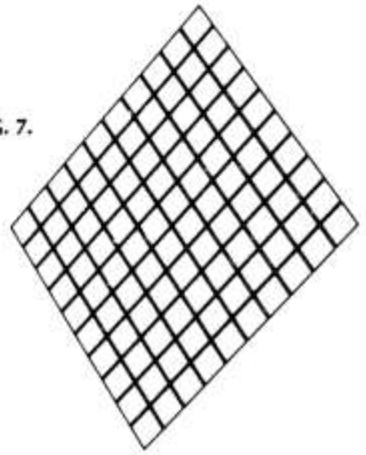


FIG. 7.



By a simple modification to the program we can also draw a more complex image. We have the data for a flat planar figure of One Hundred Squares. But, by using the raw *x* and *y* values and then creating another figure with the same *x* and *y* values but with a different *z* value, it is possible to construct a wireframe model of a rectangular solid. The flat planar figure becomes a true representation of a three-dimensional solid. Figs. 8, 9, and 10 reveal this solid object from our three different viewpoints.

FIG. 8.

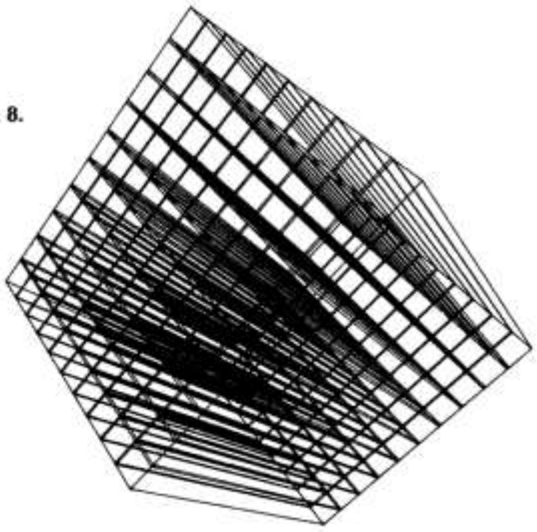


FIG. 9.

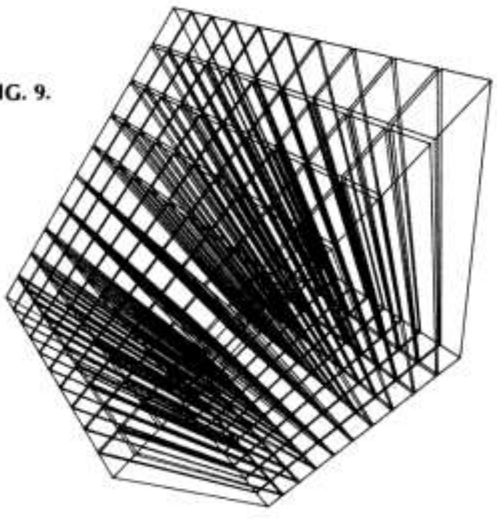
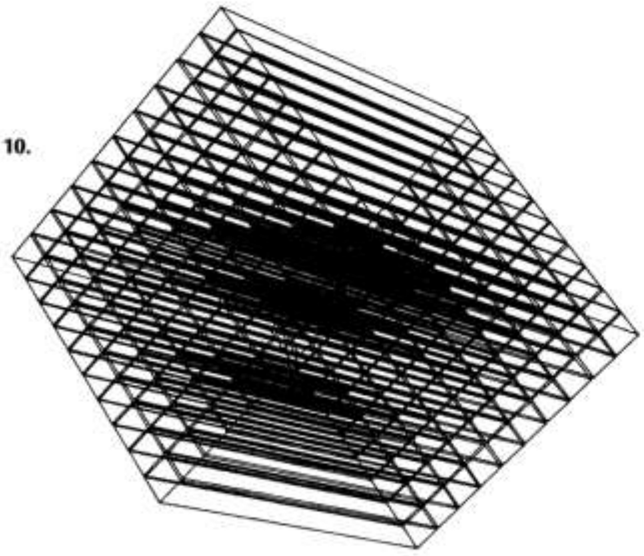
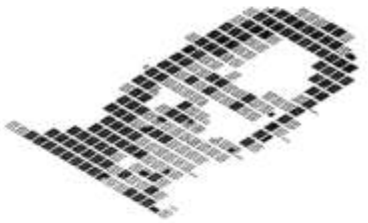


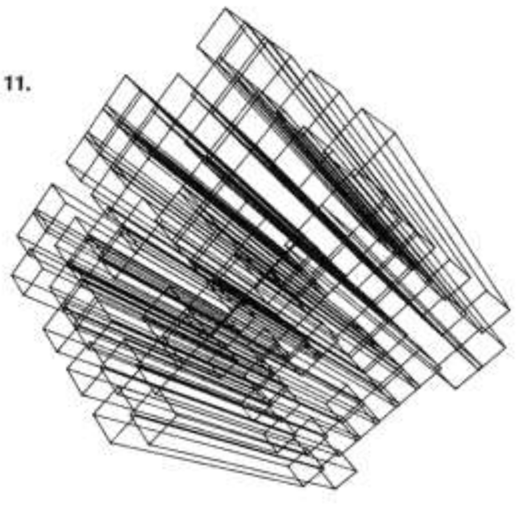
FIG. 10.





Finally, this three-dimensional wireframe model can be easily modified. As an example, we can supply the random-number test used earlier and modify the dimensions of our figure. About half of the squares are plotted in Fig. 11.

FIG. 11.



REALISM AND THE SYNTHETIC PICTURE

IT IS POSSIBLE to construct simple wireframe models of solids. Very complex figures can also be constructed with wireframe models. The hundred squares, or rather, the hundred cubes, can be converted into a representation of a three-dimensional object. The cubes consist of six polygons. By extension, almost any sort of object can be represented by building complex, polygonal wireframe models. Such constructions, while useful, are only the palest reflection of the visual reality that surrounds us. Our everyday visual reality consists of objects and space. Relatively small numbers of those objects are wireframe objects; most objects have complex, highly variable opaque surfaces that reflect colored light in many different ways.

Thus one of the most immediate problems confronting us in creating a computer-generated image that would mirror reality is the depiction of opaque objects. We cannot see through most objects. This problem is known in computer graphics as the hidden-line or hidden-surface problem. How can the machine eliminate or erase the lines or surfaces that are hidden by the surface of a real object as we see it?

Much work has been done on this problem. There are a variety of approaches and most of them involve a sorting procedure. A three-dimensional scene, with the attendant *z* axis depth information, contains numeric data about the distance of various objects from the eye. These polygonal surfaces can be sorted by the computer before they are displayed. Obviously, it will require a considerable amount of computation time to perform the sort if a picture has a large number of polygons.

One of the most common techniques used to sort polygons relies on the fact that some of the polygons are facing toward the viewer, while some are facing away. Those that are facing away would not be visible and therefore need not be displayed. This technique works for single objects, but it cannot resolve the display of multiple, overlapping objects. Other techniques examine the *z* depth of each pixel to be displayed in a video display system. Of course, even a medium-resolution display will require considerable time for a search if a considerable number of surfaces are involved.

After the image has been searched and the hidden lines or surfaces have been eliminated, illumination must be provided. The question of illumination has two aspects. First, what is the quantity and quality of the illumination, and, second, what happens to the light when it strikes the surface of the objects.

The light impinging upon a scene may come from a single discrete source, from multiple sources, or from diffuse sources. In the case of light coming from specific sources, shadows must be calculated.

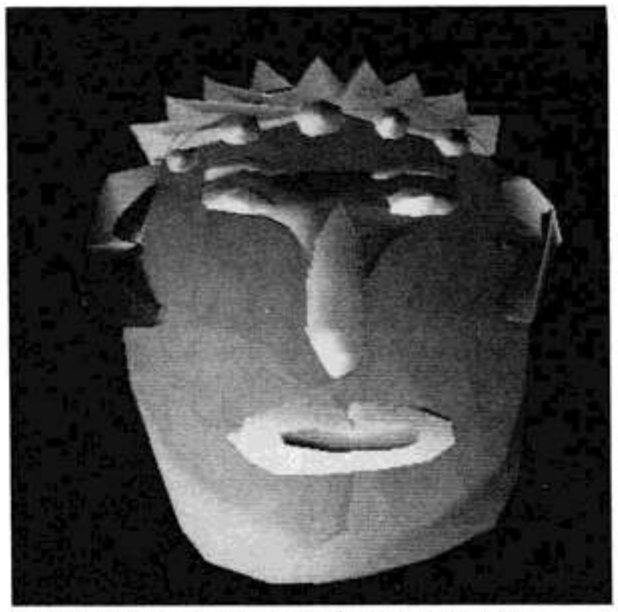


FIG. 12a. Isaac Victor Kerlow. *Mask 1.1,* 1984 screen photograph. VAX 11/780 and Grinell 270 color terminal. © 1984 Isaac Victor Kerlow.

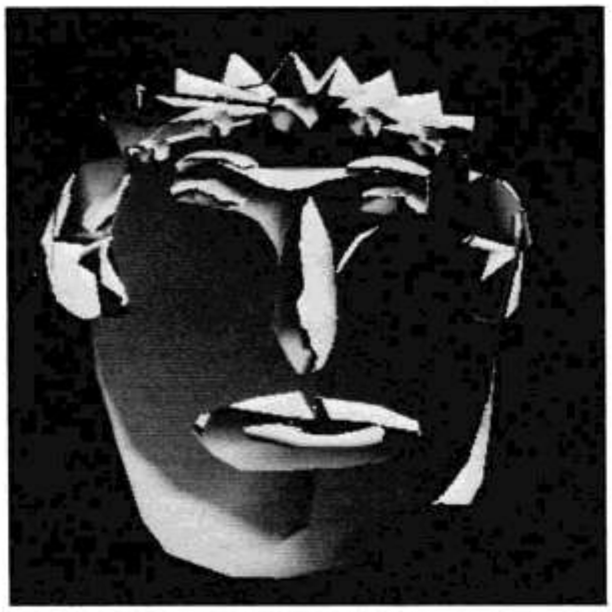


FIG. 12b. Isaac Victor Kerlow. *Mask 1.2,* 1984 screen photograph. VAX 11/780 and Grinell 270 color terminal. © 1984 Isaac Victor Kerlow.

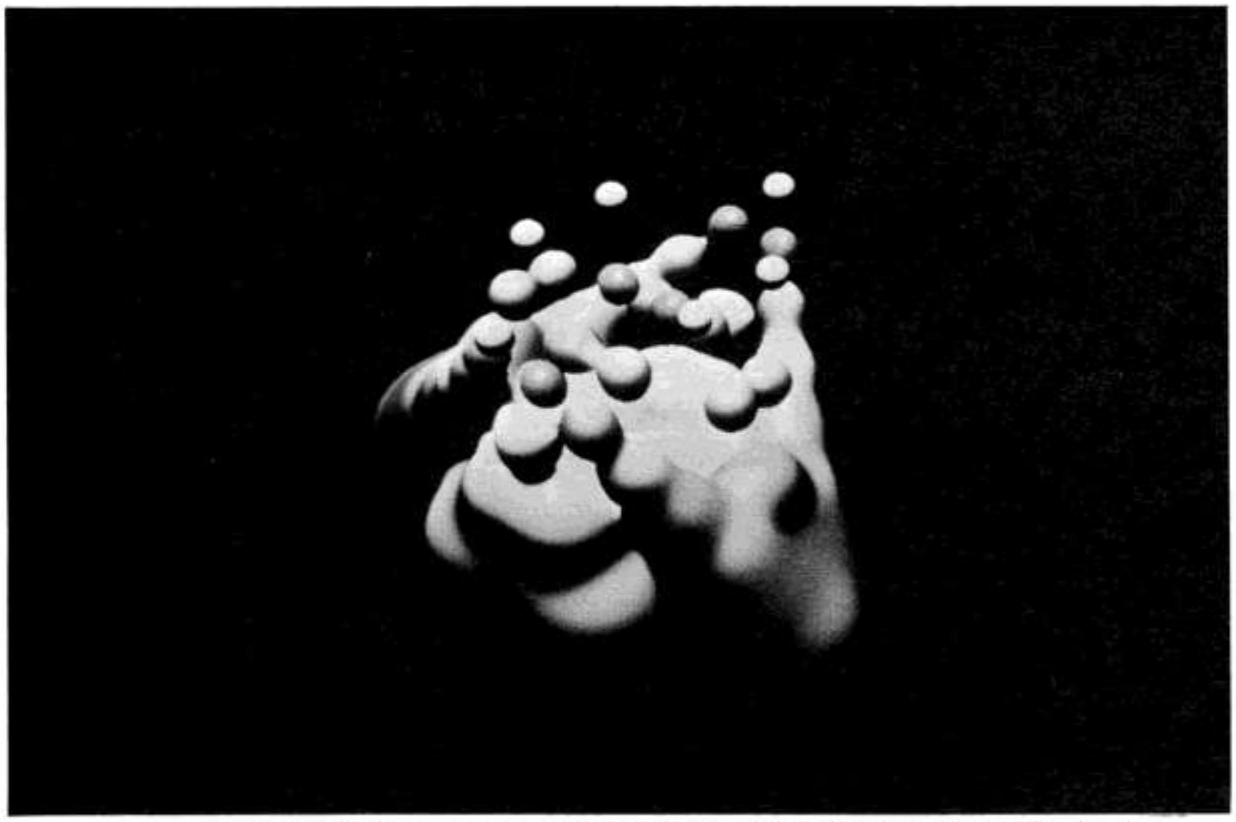


FIG. 13. Frank Dietrich. *Vedic Blobs,* 1984 screen photograph, 24-bit frame buffer. © 1984 Frank Dietrich.

As the light hits a surface it is reflected, absorbed, or transmitted. This simple description is absolutely accurate, yet it is somehow totally inadequate when we contemplate the extraordinary variety of ways that light can be modulated by surfaces. Consider such disparate objects as flesh, glass, water, leaves, or clouds. Mathematical procedures have been created that will account for some of the different ways in which surfaces reflect and transmit light.

Some algorithms calculate, pixel by pixel, the actual values of light that is reflected or transmitted. This is called ray tracing. Such ray tracing actually calculates the path of light rays from each pixel in the image. Needless to say, ray tracing requires large amounts of processing, but represents a great advantage in terms of pictorial accuracy.

THE SUCCESSES AND FAILURES OF THE SYNTHETIC PICTURE

IN LITERALLY TWO DECADES, computer graphics technology has evolved from rudimentary graphics systems to highly complex picture-making devices. High-speed computers coupled to very high resolution monitors capable of displaying millions of different hues, driven by enormously complex software, have created pictures such as *The Road to Point Reyes.* (See Fig. 1, p. 9.) But because of the technical difficulty, expense, and software complexity involved in making these detailed images, most of these images have been produced in industrial and scientific settings. Frequently, the software is written collaboratively. As microcomputer graphics systems increase in capacity and decrease in cost, the near future should see startling advances in low-cost synthetic realism. Naturally,

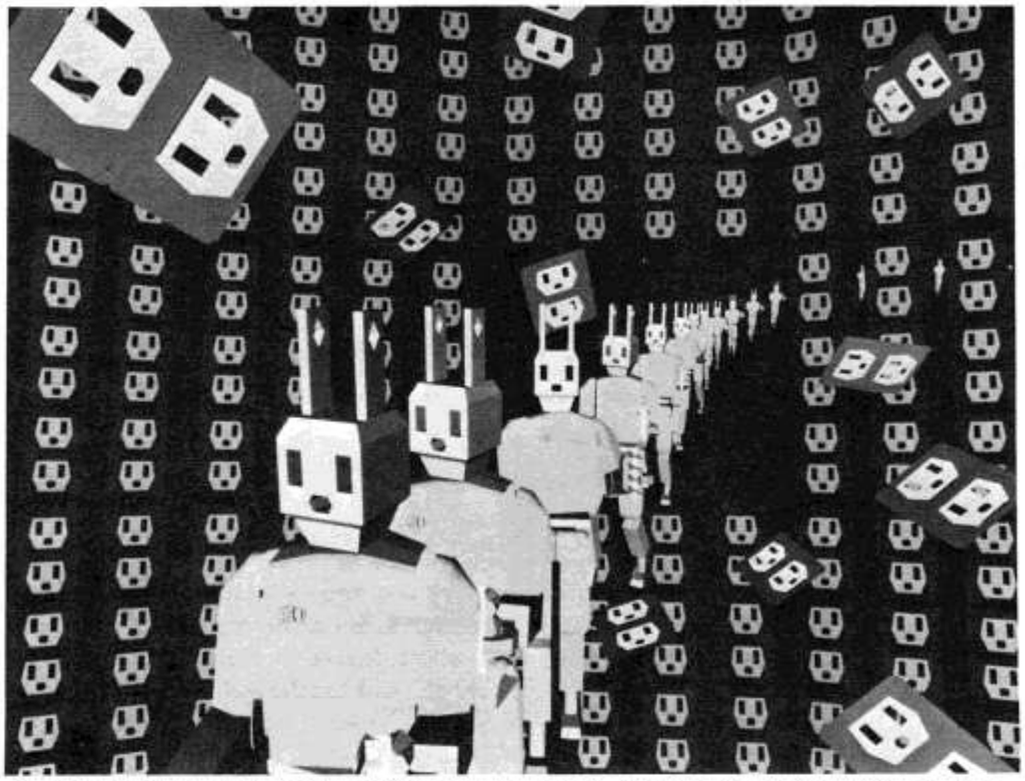


FIG. 14. Joe Pasquale. *Hello Plugs,* 1983 screen photograph. APL and Digital Effects Vision System software running on an IBM 4341 and PDP 11/34. © 1983 Joe Pasquale/Digital Effects.

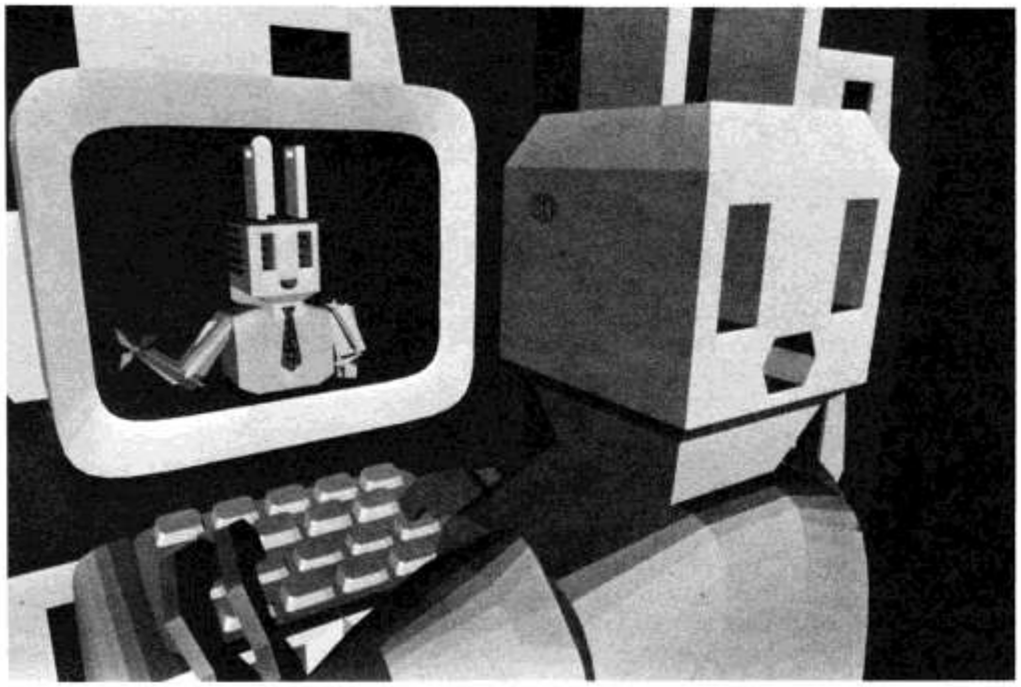


FIG. 15. Joe Pasquale. *Supervisor of Extensions,* 1983 screen photograph. © 1983 Joe Pasquale/Digital Effects.

there is great commercial incentive to develop the software techniques for modeling and simulating the visual world. Some of the most conspicuous examples have been in movie animations such as those used in *Tron* and *The Wrath of Khan.* Highly realistic computer images are also used in aircraft flight simulators. As hardware continues to improve and software becomes widely available, it is to be expected that more artists will be able to use these techniques of computer-generated illusionism.

While such simulations represent a great achievement, there remains much to be accomplished. An algorithmic simulation of the human body is very difficult. This should hardly come as a surprise—it is very difficult to accurately render the human body by traditional artistic means. Or, at least, it requires considerable training and study, not to mention talent. Indeed, it is possible to view art history as a record of artists' struggles to master illusionism. Many realistic computer images have a strange, wooden quality about them. It is obvious that many of the artist's hard-won conventional techniques have yet to be successfully applied to computer-generated images. Much of the computer work on the simulation of reality has centered on creating a photographic likeness of the world. Yet painters

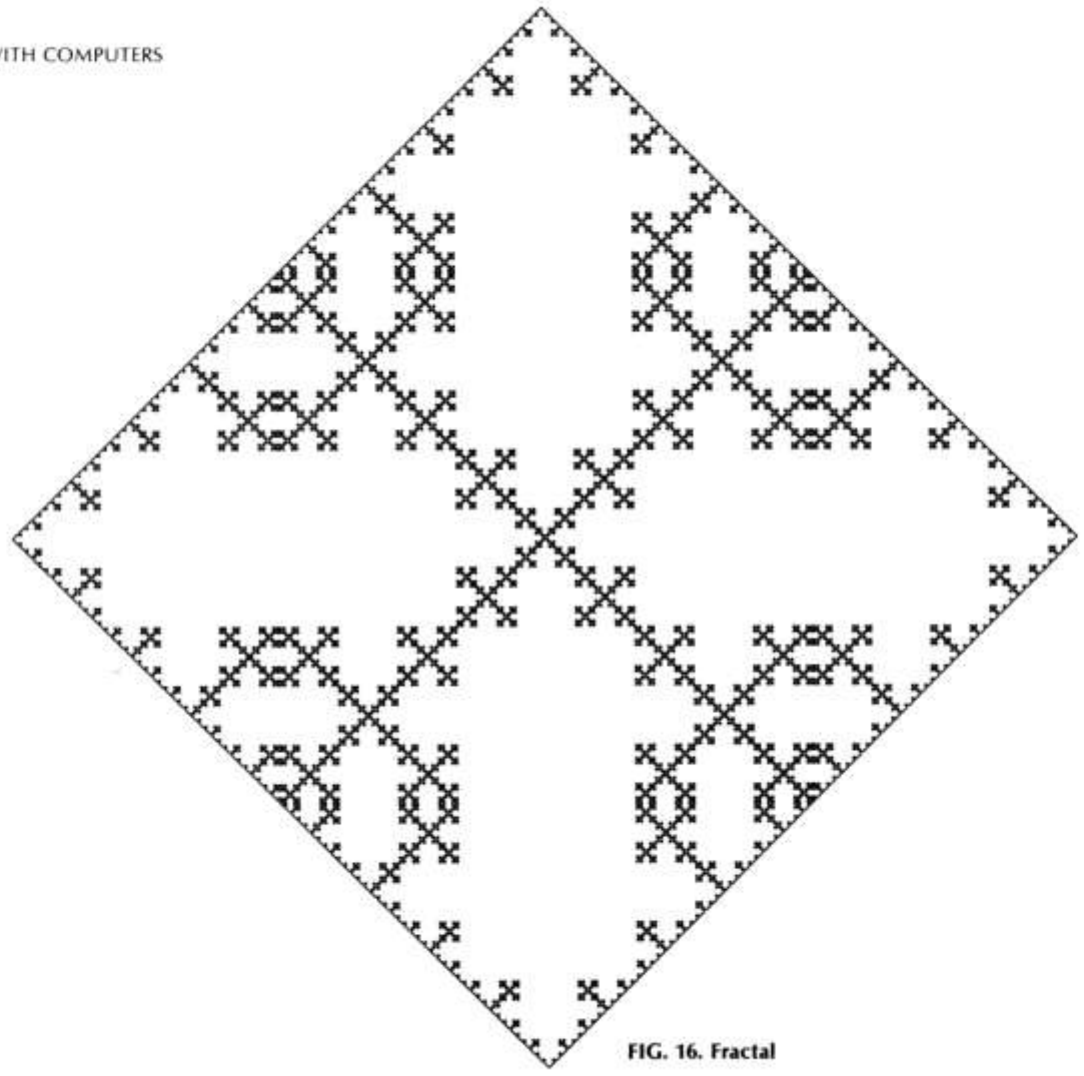


FIG. 16. Fractal

have long known that mere likeness is not a guarantee of success. Often, artists have selectively eliminated the details in a picture in order to enhance and enlighten our vision.

FRACTALS

ONE OF THE new mathematical techniques that has been used in creating images is fractal geometry. Although the subject has many highly technical aspects, computer graphics have made fractals instantly accessible to everyone. To greatly oversimplify, the geometric process essentially consists of taking some figure—typically, a line segment—and performing a subdivision or deformation of that segment. The same action is then performed on the resulting subsegments. The action is repeated again and again. Fig. 16 shows a pattern that has been created using a procedure developed by Benoit Mandelbrot, a French mathematician. One side of a square is divided into three equal line segments. Using three such line segments, a U-shaped figure is constructed between the first and second segments. This procedure is then used again on the resulting five line segments. After plotting the figure with the U shape swelling outward, the square is then plotted with the U shape swelling inward.

Such fractal procedures have also been used to create realistic scene simulations. The marriage of these fascinating mathematical operations with computer graphics offers us a hint of what the future may hold. The intellectual and visual vitality of fractal geometry and synthetic realism cannot fail to stimulate the artistic imagination.

11. PAST, PRESENT, AND FUTURE

IN THE PAST, computer art has elicited much interest. While the art world became enamoured with technology in the late sixties, it soon lost interest. The excitement of the famous Experiments in Art and Technology—EAT—soon gave way to waves of new experiments and trends. New realism and photorealism became dominant in the seventies and, in turn, were displaced by a rekindled interest in expressionism. Thus, the official New York art world has largely ignored computer art.

In many cases, the attitude was well justified. The computing community has unfortunately muddied the question by a very simple failure of language. The computing community has been guilty of confusing two things—pictures and art. In the early days of computer graphics, any picture that was made by a computer was automatically labeled "computer art." This semantic confusion probably can be forgiven, because of the initial excitement in being able to make a computer produce some kind of image. But it is, unfortunately, a confusion that is alive and well today. While many computer-produced images are extraordinary and fascinating, they are not necessarily art. Too often, the standard "computer art" image is some type of Spirograph-like, symmetrical figure. Such images do indeed intrigue the eye, but soon become tedious and cloying.

The situation is reminiscent of what followed the introduction of the kaleidoscope in England by Sir David Brewster (1781–1868). Brewster was a Scottish scientist who in 1819 published "Treatise on the Kaleidoscope." He constructed an instrument that was essentially identical to the modern kaleidoscope and coined a Greek name meaning "beautiful-image viewer." Brewster believed that the instrument would be useful in the arts and even thought that it might lead to a new art form, which he called "color music." The kaleidoscope became a wild success in London and Paris. Thousands were sold.

We have all used kaleidoscopes with great delight. Yet few of us would describe these experiences as the pinnacle of art making. The kaleidoscope offers an interesting parallel to the computer as picture maker. Obviously, the computer as image viewer is capable of more than a kaleidoscope, yet many of the images produced with the computer have failed to go beyond the most obvious capabilities of the machine. Like the kaleidoscope, the computer effortlessly takes an image, mirrors the image, and repeats the image.

Often pictures made with paint programs suffer from a similar confusion between what is art and what is merely a picture. A clumsy, inept image that was drawn on a screen is described as art purely because it was drawn with a computer. Had the same image been created with a pen or paintbrush, it would be ignored. The magic of image making with a computer sometimes blurs our aesthetic sensibilities.

Often the issue of taste masks a more interesting issue: What methodology should

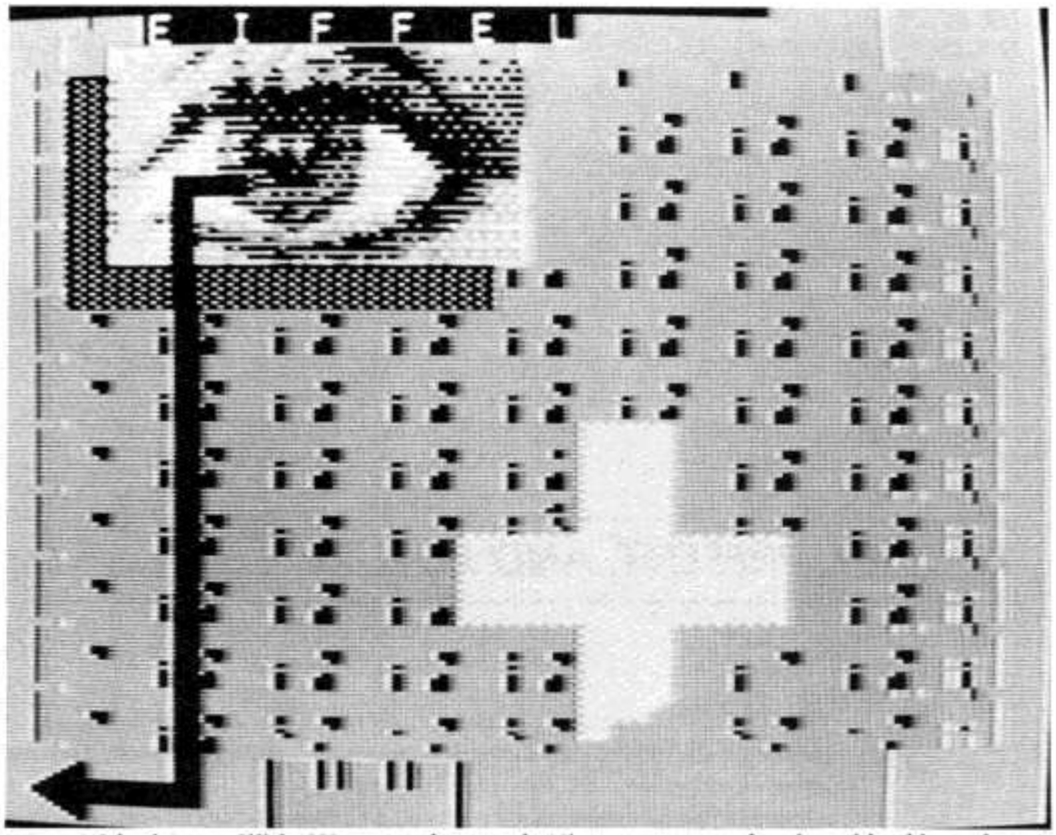


FIG. 1. Michael Arent. *Eiffel,* 1983 screen photograph. Microcomputer workstation with tablet and video digitizer. © 1983 Michael Arent.

the artist use in making images with the computer? The two extremes of this question are represented by images that are created without any direct intervention of the artist and images that are created totally by direct intervention of the artist. Artists like the Bangerts and Harold Cohen create drawings that take their form from the peculiarities of the software. When the program executes, the drawing will be made based purely on internal decisions of the software. Naturally, all of those decisions the software is making are ultimately made by the programmer-artist. At the other extreme are artworks made with paint programs. Such works are produced by what are essentially extensions of conventional art-making techniques. The computer is used simply as an electronic tool. Between these two extremes is a broad spectrum of art-making strategies that rely on both the intervention of the artist and the internal decision making of the software. Personally, it seems to be that both approaches are valid. If the image is exciting and interesting, whether it was produced with a generative process or a paint program is irrelevant. Much of the excitement in computer graphics derives from the vigor of the image making, and while many of these images are artistically naive, often they represent a dramatic new viewpoint. Just as those who would make pictures with the computer can learn a great deal by studying art and art history, by the same token, the artist can learn a great deal from the special problems of creating pictures with the computer. Computer graphics have shown us things we could not see before. What could be more exciting and stimulating to the artist?

As microcomputers continue to increase in power and speed while simultaneously decreasing in cost, graphics peripherals will follow this same pattern, although probably not as dramatically. As we have seen, the artistic and graphics applications of these machines are so diverse that it is hard to characterize all of the ways they will be employed. This book has concentrated on static two-dimensional uses. But the artist can employ these machines in many other ways. Like the camera, the computer can produce either individual still images or dynamic, moving images. Because of their very nature, dynamic images, on both video and conventional film, will continue to be very important. Dynamic computer imagery will probably evolve into a separate genre, related to still images as cinema is to still photography.

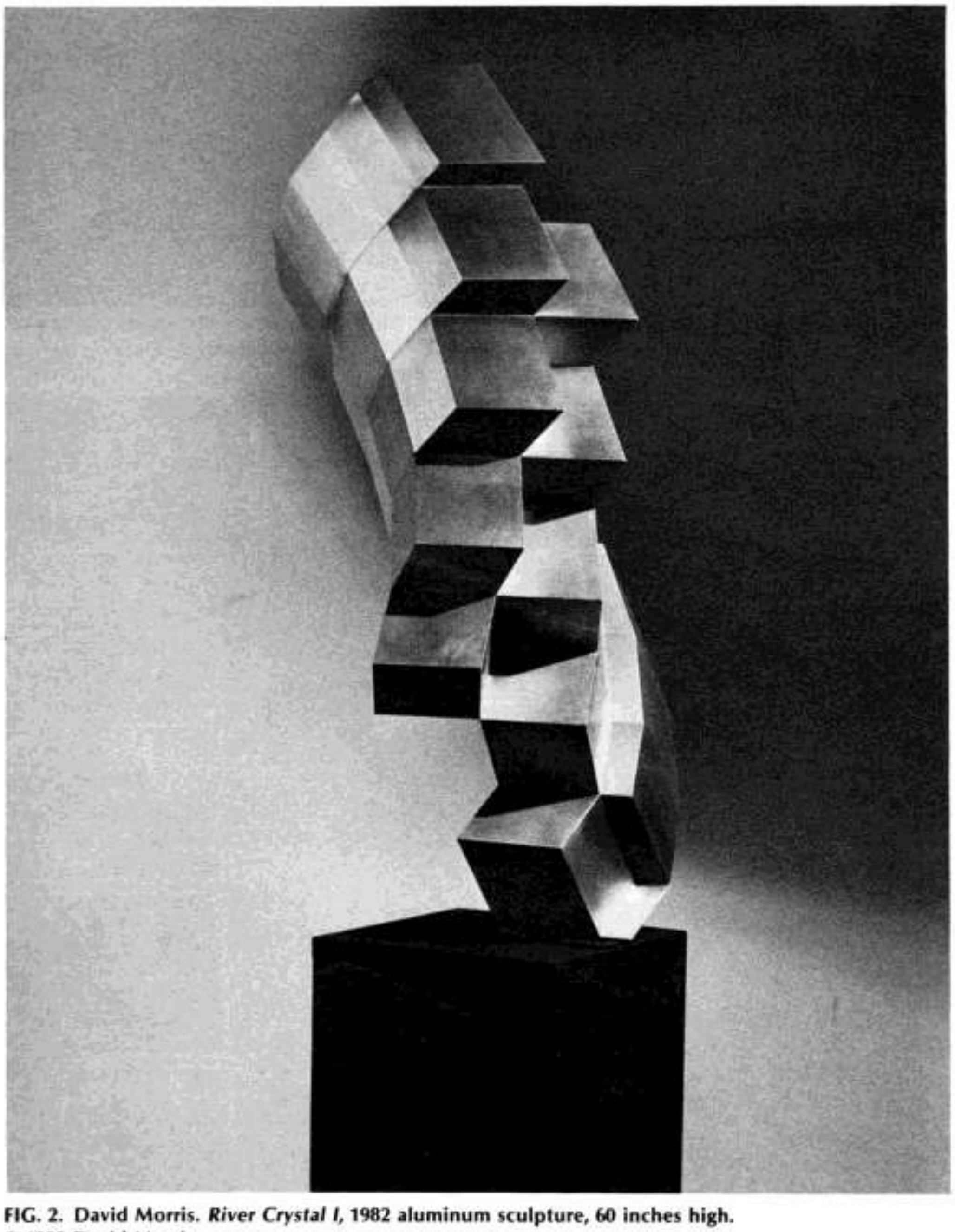


FIG. 2. David Morris. *River Crystal I,* 1982 aluminum sculpture, 60 inches high. © 1982 David Morris.

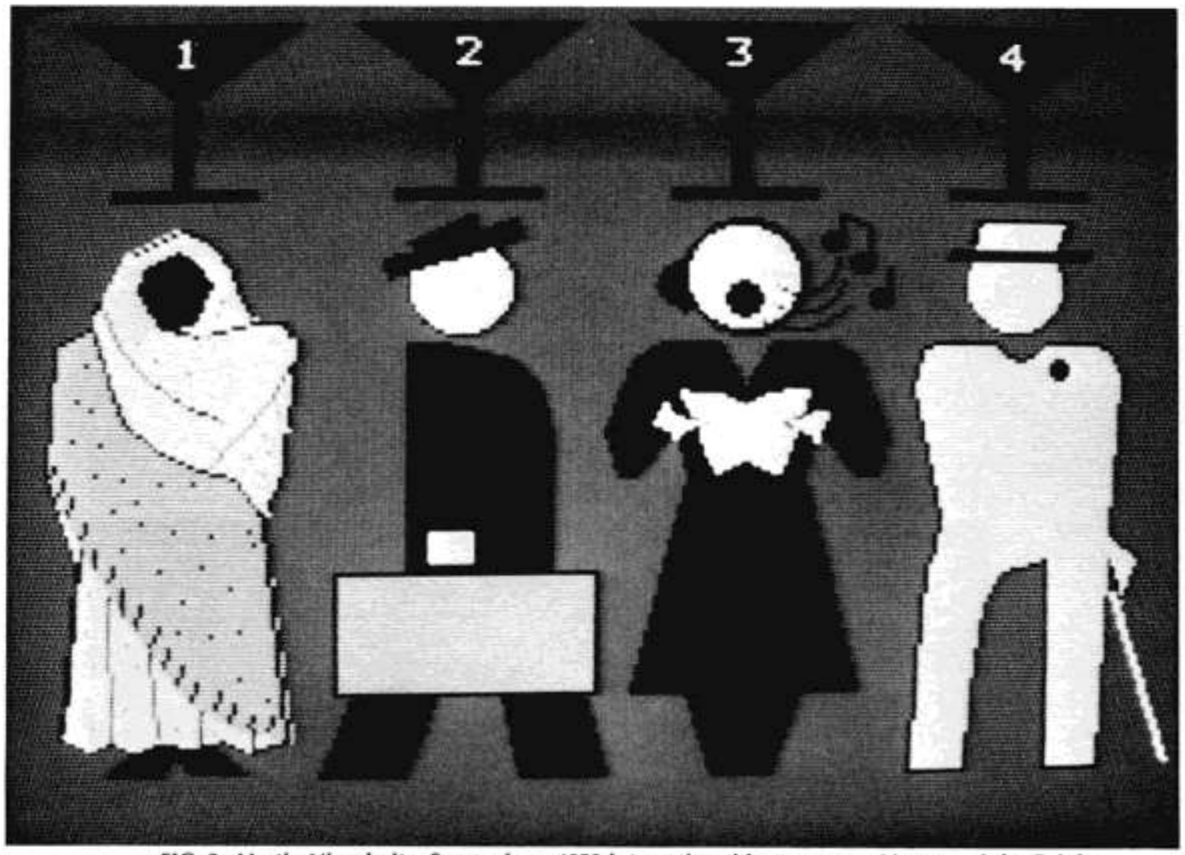


FIG. 3. Martin Nisenholtz. Screen from 1983 interactive videotex story, *Mystery of the Drink*, composed entirely of pictograms. © 1983 Martin Nisenholtz.

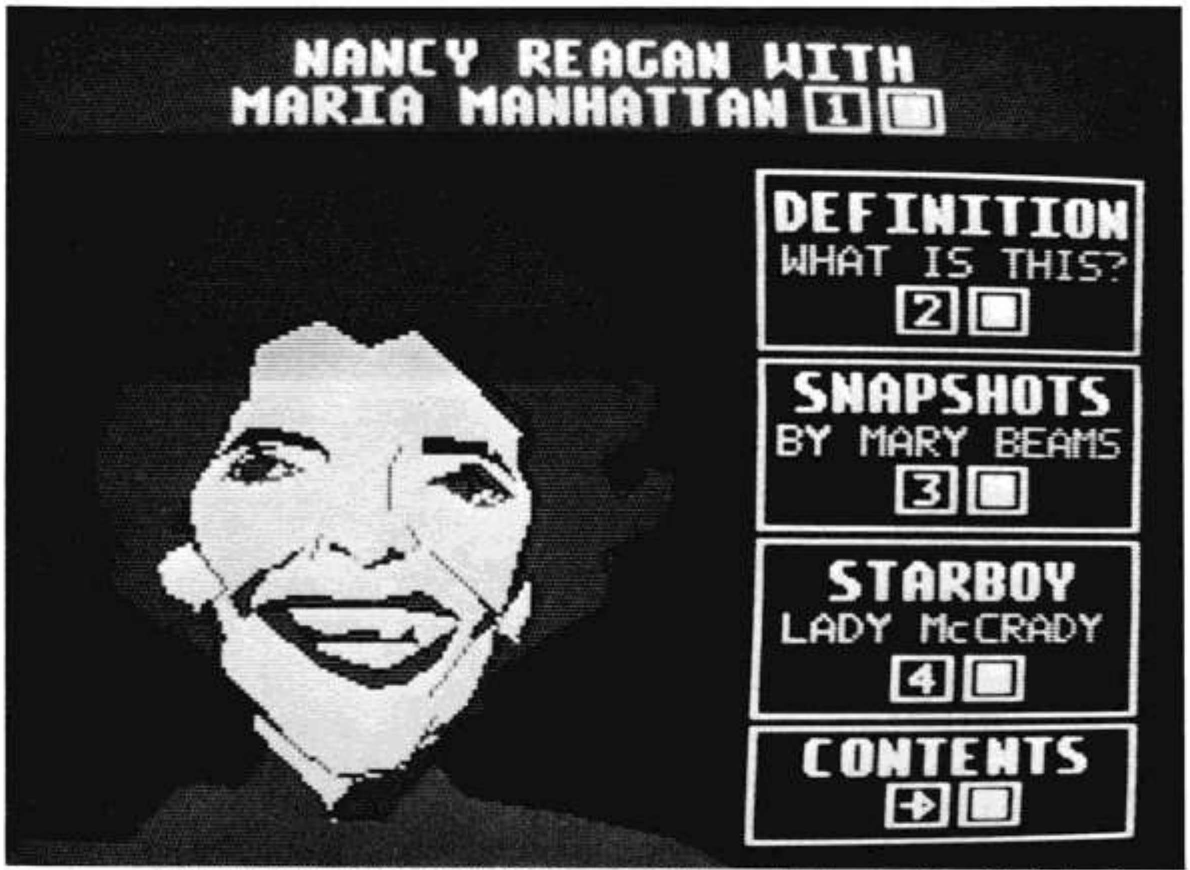


FIG. 4. Maria Manhattan and Martin Nisenholtz. 1983 cover videotex screen from *The Electronic Gallery*. © 1983 Martin Nisenholtz and Maria Manhattan.

Artists who work in three-dimensional media, such as sculptors, are also learning to use the computer. All of the CADCAM techniques that have been employed by industry are applicable by the sculptor. He or she can design a sculpture on the screen. The piece can then be colored with various paints, patterned, rotated, viewed from a variety of aspects, and set into various environments. When the design has been settled upon, the computer can engineer the object, draw up specifications and a bill of materials, calculate the costs, and finally draw the blueprints.

The inherently interactive nature of computers offers much potential in creating dialogues between people and the machines. Relatively little has been done in this area, but microcomputers would seem to be a natural medium for such undertakings. Interactive projects in New York City and São Paulo, Brazil, have used videotex in an artistic context.

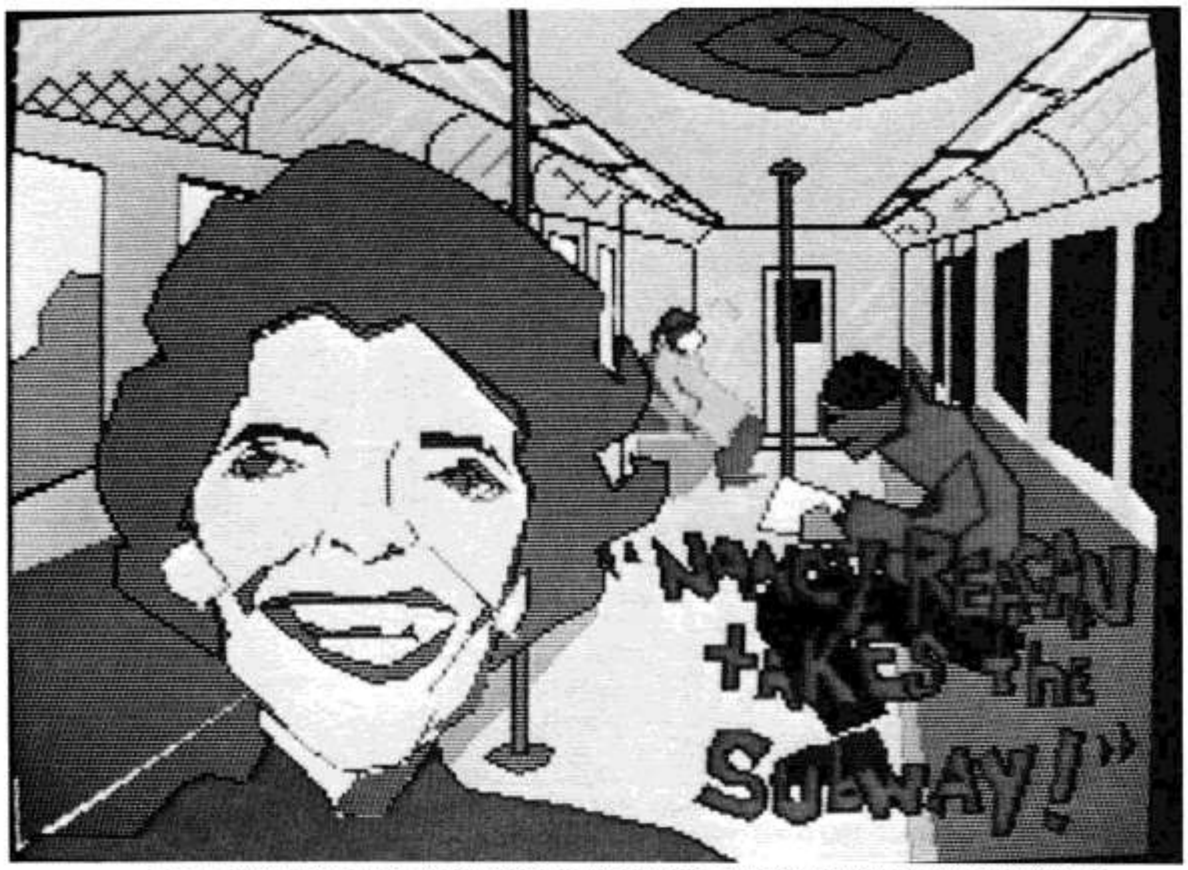


FIG. 5. Maria Manhattan. *Nancy Reagan Takes the Subway*, 1983 videotex screen. © 1983 Maria Manhattan.

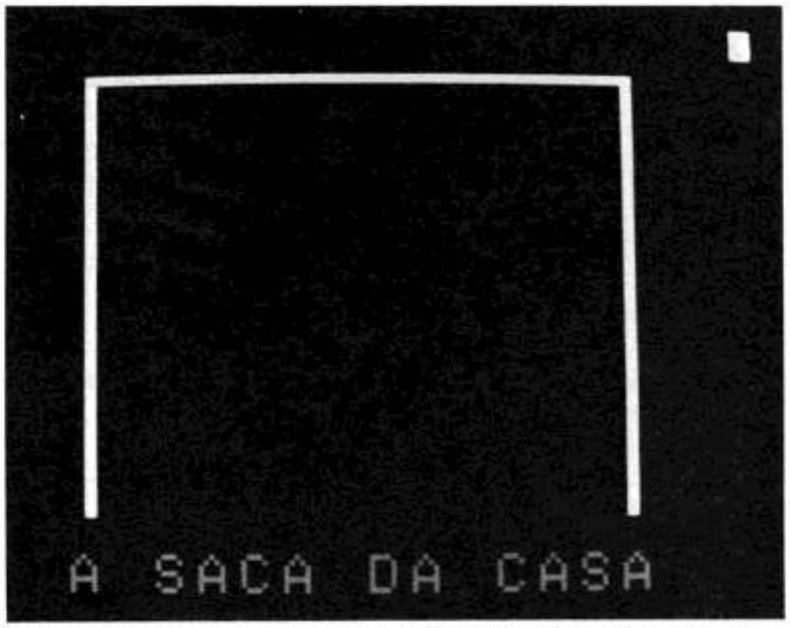


FIG. 6. Julio Plaza. *A Saca da Casa,* 1983 videotex screen. © 1983 Julio Plaza.

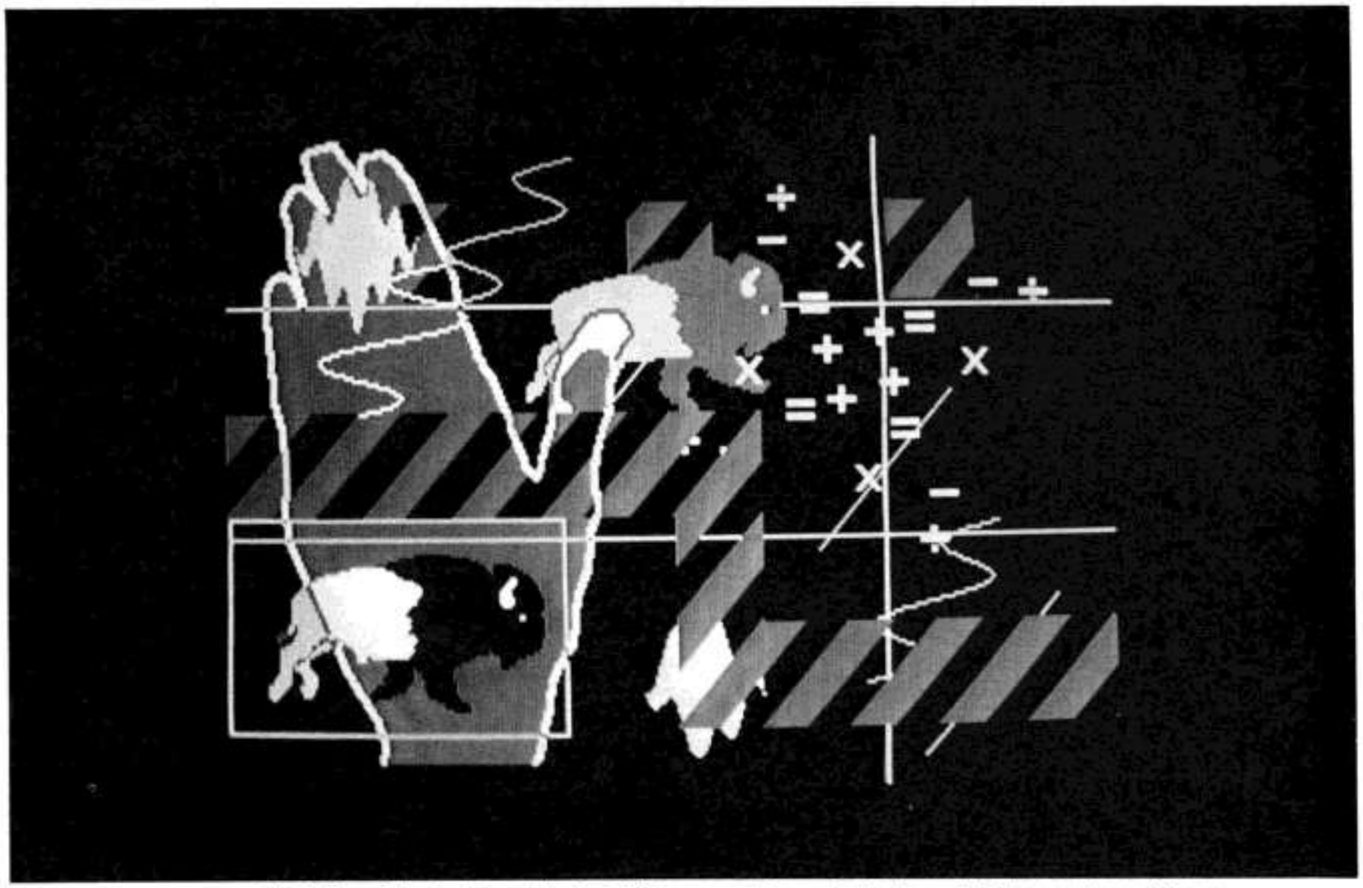


FIG. 7. Jane Veeder. From *Floater,* 1983 screen photograph. Datamax UV-1 computer and Zgrass software. © 1983 Jane Veeder.

Videotex is a graphics-oriented television information system that allows the viewer to use a keyboard to exchange information with the system. Julio Plaza, a Spanish-born Brazilian artist, created an interactive videotex project called *Art on Line.* The viewers could, if they desired, react to the artistic images on the screen, both choosing alternative images and selecting different artists, as well as being able to leave criticisms and messages.

Another fertile area would seem to be the application of robotics and real-time control. Robots are used in the automotive industry to paint auto bodies. Why not use them to paint pictures? Robots also could be used in, say, environmental art pieces; and microcomputers with real-time capabilities would seem to be an obvious means of controlling various processes and events in environmental artworks.

In most cases, the hardware exists to realize almost any sort of imaginable art project. The software does not. It seems that the real future of computer-generated artworks will depend on the development of appropriate software.

GLOSSARY

Additive colors—A set of primary colors comprised of red, green, and blue. When mixed together they produce white.

Algorithm—A precisely defined mathematical procedure. Commonly embodied in computer software as a means of performing some type of operation.

Aliasing—The "stair-stepping" effect resulting from coarse resolution of nonvertical or nonhorizontal graphic elements. Sometimes called "jaggies."

Bit-mapped display—A graphics display in which each picture element—whether a pixel on a screen or a dot from a matrix printer—can be directly controlled or manipulated.

CADCAM—Computer-aided design, computer-aided manufacturing. Sometimes CAD or CAM. An integrated system for designing, drafting, and engineering architectural structures or industrial products.

Calligraphic—A line- or vector-drawing display device. Lines are generated as "strokes" between endpoints.

Cathode ray tube (CRT)—A display device that uses a raster scan of electrons striking a phosphor-coated inner surface to create an image. Televison sets and video monitors both use cathode ray tubes.

Clipping—The removal of graphic information outside a designated viewing area, or viewport. Can be performed by hardware, software, or both.

Color lookup table—A list of the intensity values for the red, green, and blue signals in a video display.

Composite video signal—The standard commercial video signal used by television, video cassettes, and many personal computers. The signal is defined by the National Television Standards Committee (NTSC).

Computer graphics—Any sort of pictorial information generated by a computer.

Coordinates—Numeric values that define the position of points on a surface. The most common form is the Cartesian coordinate, which locates points on a two-dimensional surface.

Cursor—A pointer indicating the current position on the screen. It can be controlled by the keyboard, or by another input device such as a mouse.

Digitizer—A graphics input device, typically a flat tablet that can generate coordinates from the indications of the user. Also refers to a device that can scan an image and generate both coordinate and color-intensity information.

Display device—Commonly refers to video equipment used to produce computer-generated visual information, such as a monitor.

Dot-addressable—See Bit-mapped.

Dot matrix printer—A printer that generates characters or images using a series of small dots.

Frame buffer or frame store—A segment of memory that stores the information presented on a video display. It stores the information about both location and color value of each pixel.

Hard copy—Information generated by a computer and stored in a permanent and accessible form. Typically, this would be a printout, a plot, or a screen photograph.

Hidden surface—Surfaces of objects that cannot be seen in the real world because of the opacity of most objects. Must be eliminated by software in a computer image by means of various software techniques.

Host computer—The computer, whether a micro or mainframe, to which a peripheral device is connected.

Ink-jet printer—A color or monochrome printer that uses very small droplets of ink to create an image.

Interlacing—A procedure in which alternatively odd and even lines are displayed by the raster scan of a video display. Standard American television uses 60 such scans per second, to produce 30 complete images in one second. Image flickering is subdued by interlacing.

Light pen—A graphic input device that, when pointed at the monitor screen, can determine if a pixel is on or off. Used for drawing on screen, and menu selections.

Joystick—Mechanical graphic input device typically used to control the relative position of the cursor on the screen of a CRT. Used for games and drawing on screen.

Monitor—Video display device. Can refer to television sets as well as high-resolution RGB devices.

Mouse—Input device often used with graphics systems. Commonly provides relative positional information for a cursor on the screen. Used for drawing and menu selections.

Paint system—A computer graphics drawing and painting system, usually consisting of a computer; monitor; graphics input device, such as a pad; and a paint program. Mimics the conventional art-making process electronically.

Persistence—The tendency for screen phosphors to glow for a period of time after being struck by the raster electron beam. Both fast and slow phosphors are used in various types of display devices.

Pixel—Acronymn for picture element. The smallest discrete graphic unit or addressable point capable of being manipulated by the computer system. The monitor screen consists of an array of pixels.

Plotter—A mechanical drawing device controlled by a computer. Typically, plotters use ink pens, but some machines utilize electrostatic plotting, photo plotting, or other means of producing images.

Primitive—A simple graphic element that can be called up with software. A line or circle would be an example of a primitive.

Raster scan—The zigzag path traced by the electron beam

in a CRT. The electron beam passes over each pixel as it scans the screen from top to bottom, right to left, tracing across the odd and even lines in alternate sweeps. *See* Interlacing.

Real-time graphics—Dynamic computer graphics images that appear to simulate actual motion.

Resolution—A measure of the number of discrete units that make up an image. Typically applied to the number of pixels that make up the displayed image.

RGB—Red, green, blue. The three computer-generated color signals that form the image in high-quality color CRTs.

Synthetic image—The illusionistic or realistic image that is generated by computer graphics.

Three-dimensional (3-D)—Computer graphics imagery that has an illusion of true perspective space. 3-D graphics are projected on a two-dimensional surface of the screen or plotter paper.

Transformation—Mathematical operations on an image that produce a variety of different views. The most important transformations are translating (moving the image about), scaling (making the image larger or smaller), and rotating the image.

Two dimensional (2-D)—Relatively simple computer graphics imagery that does not contain the illusion of true perspective space.

Vector—A line. *Also see* Calligraphic.

Videotex—A computer information service that combines both text and graphics.

Viewport—A rectangular area into which a segment of an image is drawn. Usually can be freely altered in *x* and *y* dimensions.

Window—A two-dimensional portion of an image that is projected onto the viewing surface or viewport.

Wireframe—An image of an object created with lines that define a series of polygons.

World coordinates or world space—A coordinate system denoting the actual dimensions of a depicted object.

BIBLIOGRAPHY

Abelson, Harold, and Andrea DiSessa. *Turtle Geometry.* Cambridge, MA: MIT Press, 1981.

Artwick, Bruce. *Applied Concepts in Microcomputer Graphics.* Englewood Cliffs, NJ: Prentice-Hall, 1984.

Cohen, Harold, and Becky Cohen. *Art and Computers.* Los Altos, CA: W. Kaufmann, 1984.

Foley, J.D., and A. van Dam. *Fundamentals of Interactive Computer Graphics.* Reading, MA: Addison-Wesley, 1981.

Greenberg, Donald, et al. *The Computer Image.* Reading, MA: Addison-Wesley, 1982.

Jankel, Annabel, and Rocky Morton. *Creative Computer Graphics.* Cambridge, England: Cambridge University Press, 1984.

Kerlow, I. V., and J. Rosebush. *Computer Graphics for Artists and Designers.* New York: Van Nostrand, Reinhold, 1985.

Knuth, Donald E. *TEX and METAFONT.* Bedford, MA: Digital Press, 1979.

Leavitt, Ruth. *Artist and Computer.* New York: Harmony Books, 1976.

Mandelbrot, Benoit. *The Fractal Geometry of Nature.* New York: W. H. Freeman, 1983.

Newman, W. M., and R. F. Sproull. *Principles of Interactive Computer Graphics.* 2d ed. New York: McGraw-Hill, 1979.

Peterson, Dale. *Genesis II.* Reston, VA: Reston Publishing, 1983.

Prueitt, Melvin. *Art and the Computer.* New York: McGraw-Hill, 1984.

PERIODICALS

Computer Graphics, "A Quarterly Report of SIGGRAPH ACM." The Association for Computing Machinery, 11 West 42nd Street, New York, NY 10036

Computer Graphics News. Scherago Associates Publishing, Inc., 1515 Broadway, New York, NY 10036

Computer Graphics World. PennWell Publishing Company, 1714 Stockton Street, San Francisco, CA 94133

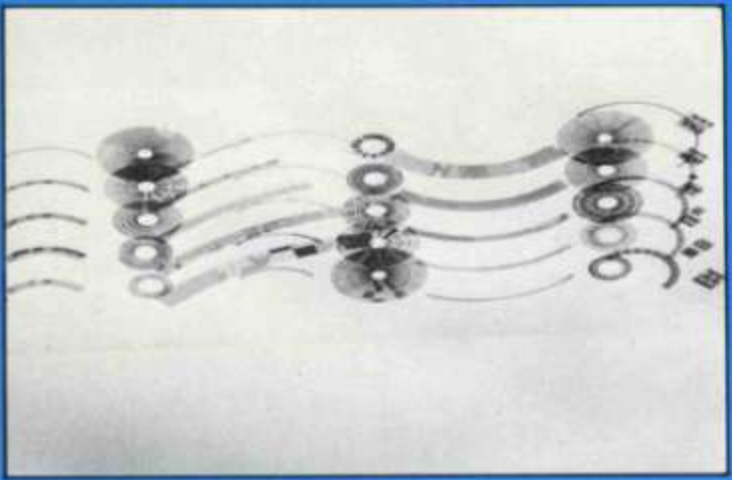
Computer Pictures. 330 West 42nd Street, New York, NY 10036

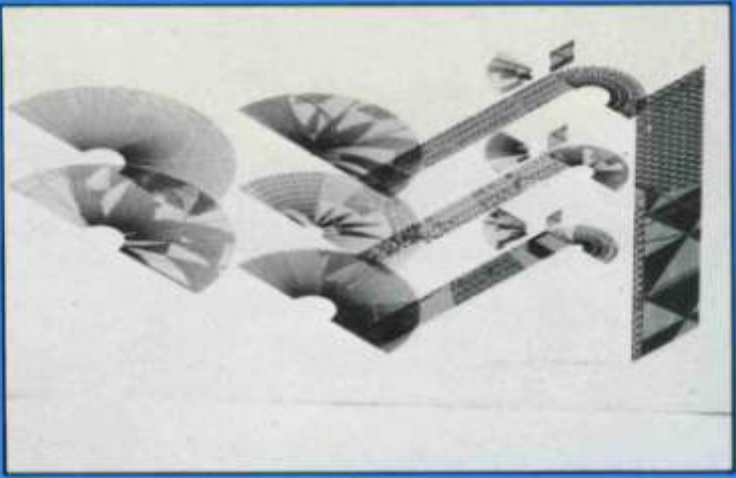
Computers and Graphics. Pergamon Press, Inc., Maxwell House, Fairview Park, Elmsford, NY 10523

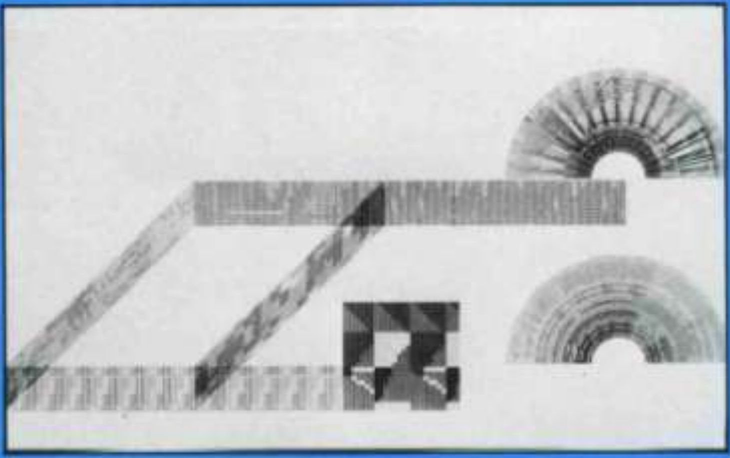
Leonardo. Pergamon Press, Inc., Maxwell House, Fairview Park, Elmsford, NY 10523

INDEX

COMPUTERS







No ordinary computer graphics book, this is the first guide to using computers for *artistic* drawings and other innovative artmaking. Visual artists working in traditional mediums as well as computer sophisticates already generating computer graphics will welcome this broad sampling of styles of computer art and the step-by-step instructions with computer code. Soon you can be "computing" art in your own individual style.

Perigee Books
are published by
The Putnam Publishing Group

0 48228 51136
ISBN 0-399-51136-9

$9.95 Price Higher in Canada

8507