

PROCEEDINGS

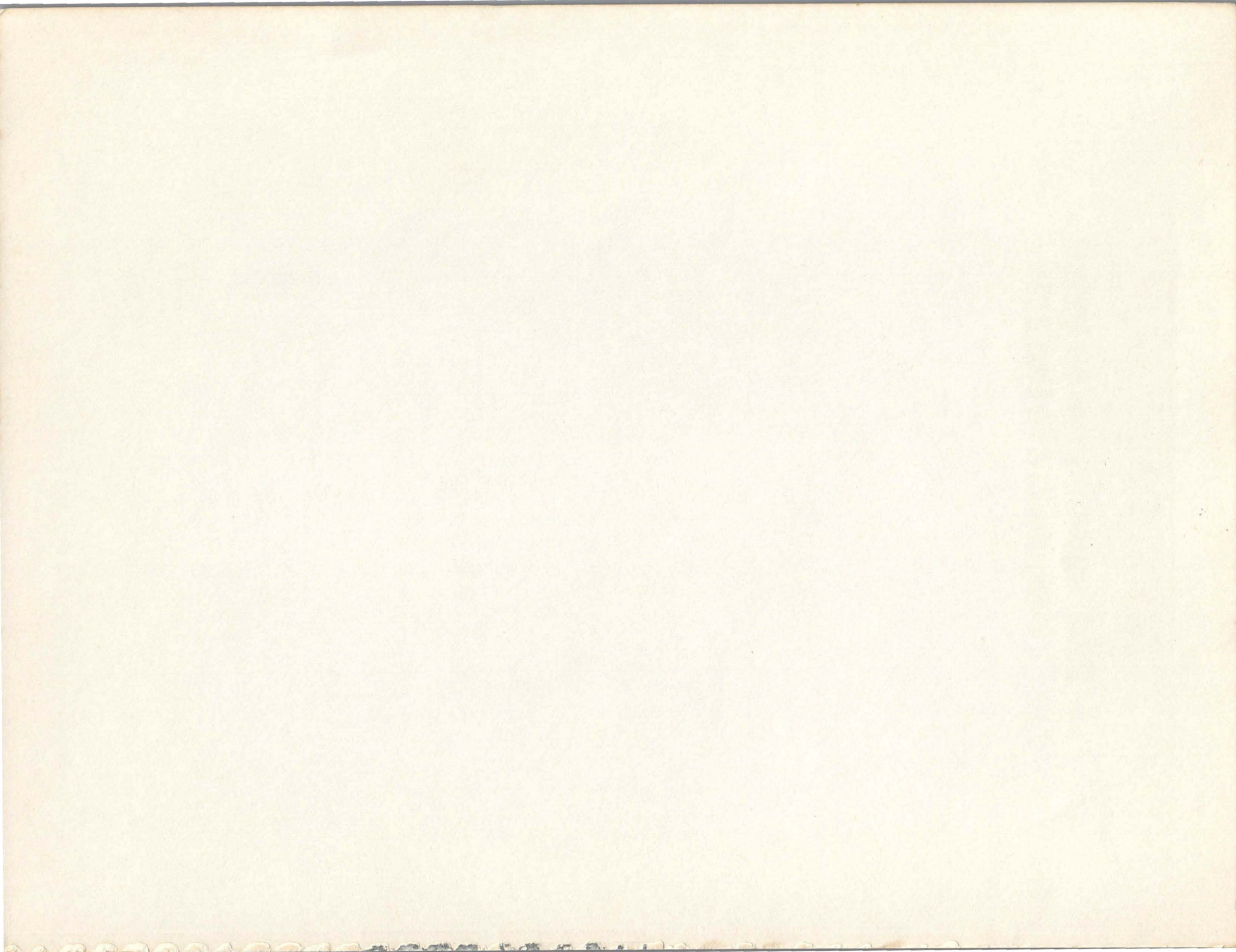
2nd. symposium on small computers in the arts

OCTOBER 15-17, 1982
PHILADELPHIA, PENNSYLVANIA

SPONSORED BY:
IEEE COMPUTER SOCIETY and IEEE PHILADELPHIA SECTION



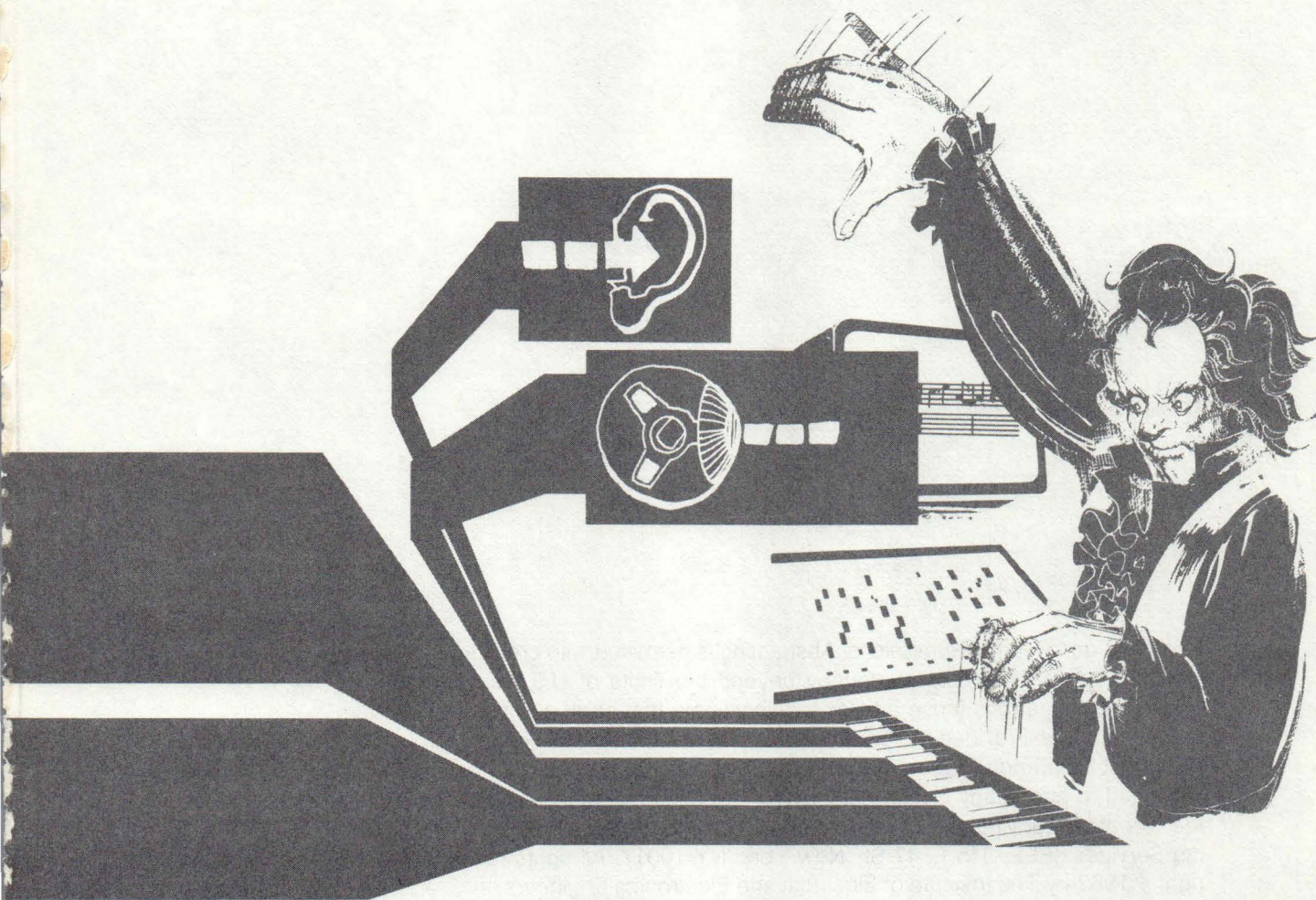
IEEE CATALOG NO. 82CH1831-7
LIBRARY OF CONGRESS NO.
IEEE COMPUTER SOCIETY CATALOG NO. 455




PROCEEDINGS **2nd. symposium on small computers in the arts**

OCTOBER 15-17, 1982
PHILADELPHIA, PENNSYLVANIA

SPONSORED BY:
IEEE COMPUTER SOCIETY and IEEE PHILADELPHIA SECTION



 THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

THE COMPUTER
SOCIETY
PRESS 

IEEE CATALOG NO. 82CH1831-7

IEEE COMPUTER SOCIETY CATALOG NO. 455

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

IEEE Service Center
445 Hoes Lane
Piscataway, NJ 08854

The papers appearing in this book comprise the proceedings of the meeting mentioned on the cover and title page. They reflect the authors' opinions and are published as presented and without change, in the interests of timely dissemination. Their inclusion in this publication does not necessarily constitute endorsement by the editors, IEEE Computer Society Press, or the Institute of Electrical and Electronics Engineers, Inc.

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

Copyright and Reprint Permissions: Abstracting is permitted with credit to the source. Libraries are permitted to photocopy beyond the limits of U.S. copyright law for private use of patrons those articles in this volume that carry a code at the bottom of the first page, provided the per-copy fee indicated in the code is paid through the Copyright Clearance Center, 21 Congress Street, Salem, MA 01970. Instructors are permitted to photocopy isolated articles for noncommercial classroom use without fee. For other copying, reprint or republication permission, write to Director, Publishing Services, IEEE, 345 E. 47 St., New York, NY 10017. All rights reserved. Copyright © 1982 by The Institute of Electrical and Electronics Engineers, Inc.

IEEE Catalog No. 81CH1631-1
Library of Congress No. 80-85189
Computer Society No. 349

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

IEEE Service Center
445 Hoes Lane
Piscataway, NJ 08854



The Institute of Electrical and Electronics Engineers, Inc.

1982
PROCEEDINGS
OF THE
SYMPOSIUM
ON
SMALL
COMPUTERS
IN THE
ARTS

OCTOBER 15-17, 1982
PHILADELPHIA

SPONSORED BY:
IEEE COMPUTER SOCIETY
IEEE PHILADELPHIA SECTION

ORGANIZED AND PRODUCED BY:
PERSONAL COMPUTER ARTS GROUP

PROCEEDINGS COMMITTEE

EDITORS..... Eric Podietz
Donna Mansfield
LAYOUT..... Laura Giannitrapani
SYMPOSIUM CHAIRMAN..... Dick Moberg

H I S T O R I C A L N O T E S

THE SYMPOSIUM ON SMALL COMPUTERS IN THE ARTS GREW OUT OF A COMPUTER MUSIC CONCERT HELD IN DOWNTOWN PHILADELPHIA IN 1978. IT WAS PLANNED AS PART OF THE PERSONAL COMPUTING '78 SHOW HELD AT THE CIVIC CENTER. JOHN DILKS, THE FOUNDER OF THE SHOW, GRACIOUSLY BACKED THE IDEA AND PROVIDED A HOTEL BALLROOM FOR THE EVENT. AS WORD OF THE UPCOMING CONCERT SPREAD, WE RECEIVED CALLS FROM PEOPLE AS FAR AWAY AS THE WEST COAST ASKING IF THEY COULD PARTICIPATE. ONE MUSICIAN FROM NEW YORK ACTUALLY ARRANGED A PIECE FOR COMPUTER AND CLARINET ESPECIALLY FOR THIS CONCERT. THE EVENING OF THE CONCERT OVER 500 PERSONS SHOWED UP AND TRIED TO SQUEEZE INTO A ROOM THAT ONLY HELD 300. THE CONCERT WAS RECORDED, AN ALBUM WAS MADE, AND IT IS NOW SOLD BY CREATIVE COMPUTING.

THE SUCCESS OF THAT CONCERT LED THE ORGANIZERS TO FORM AN INFORMAL GROUP TO PRODUCE SIMILAR EVENTS AND TO ACT AS A CLEARINGHOUSE FOR THOSE INTERESTED IN COMPUTER APPLICATIONS IN THE ARTS. THE 1979 PERSONAL COMPUTER MUSIC FESTIVAL, SPONSORED BY THE GROUP, INCLUDED TALKS AND DEMONSTRATIONS DURING THE DAY IN ADDITION TO THE EVENING CONCERT. IN 1980, A SEPARATE DAY OF COMPUTER GRAPHICS TALKS AND DEMONSTRATIONS WAS ADDED TO MAKE THE PERSONAL COMPUTER ARTS FESTIVAL. ALL THESE EVENTS WERE HELD AT THE PERSONAL COMPUTING SHOWS IN PHILADELPHIA.

IT HAD ALWAYS BEEN OUR DESIRE TO SOME DAY ORGANIZE A MAJOR MEETING SOLELY DEDICATED TO THE USE OF SMALL COMPUTERS IN THE ARTS. THIS DREAM BECAME A REALITY WITH THE 1981 SYMPOSIUM, THANKS TO THE SUPPORT OF THE IEEE COMPUTER SOCIETY AND THE IEEE PHILADELPHIA SECTION.

AT LAST YEAR'S SYMPOSIUM, HELD NOV 20-22, 1981, 150 COMPUTER ARTS ENTHUSIASTS DESCENDED UPON THE HOLIDAY INN ON THE UNIVERSITY OF PENNSYLVANIA CAMPUS. AN INFORMAL AND VERY INTERACTIVE ATMOSPHERE PREVAILED MAKING THE SYMPOSIUM A GREAT SUCCESS WITH RESPECT TO ITS GOALS OF INCREASING THE PARTICIPANTS' AWARENESS OF THE USES OF SMALL COMPUTERS IN THE ARTS. RESPONSE TO THE SYMPOSIUM AND SUCCEEDING INTEREST LED TO THE PUBLICATION OF 'SCAN', A MONTHLY NEWSLETTER ON SMALL COMPUTERS IN THE ARTS.

THE PERSONAL COMPUTER ARTS GROUP, AS A VOLUNTEER NOT-FOR-PROFIT-GROUP, CONTINUES TO PROMOTE THE USE OF COMPUTERS IN THE ARTS THROUGH ITS NEWSLETTER 'SCAN', CONCERTS, AND OTHER EVENTS. TO CONTACT THE GROUP, WRITE TO: PERSONAL COMPUTER ARTS GROUP, BOX 1954, PHILADELPHIA, PA. 19105.

P R E F A C E

OVER THIS PAST YEAR WE HAVE SEEN THE USE OF COMPUTERS IN THE ARTS POPULARIZED THROUGH MOVIES, TV DOCUMENTARIES, AND MAJOR NEWS MAGAZINES. IT SEEMS THAT MORE AND MORE ARTISTS AND MUSICIANS CAN'T HELP QUESTIONING HOW THEY COULD USE COMPUTERS IN THEIR OWN WORK, WHETHER IT BE A NEW WAY OF VISUALIZING MUSIC OR A MEANS OF TESTING AN ANIMATION.

NEW SOPHISTICATED INTEGRATED CIRCUITS GEARED FOR SPECIFIC SOUND/GRAPHICS FUNCTIONS HAVE DRIVEN THE COST OF A PERSONAL COMPUTER DOWN AND THEIR CAPABILITIES UP. MANUFACTURERS OF TRADITIONALLY SELF CONTAINED, DEDICATED EQUIPMENT, SUCH AS CHARACTER GENERATORS AND SYNTHESIZERS, HAVE BEEN DESIGNING INTO THEIR PRODUCTS INTERFACES TO POPULAR MICRO-COMPUTERS. SOFTWARE IS MORE AVAILABLE FOR A WIDER RANGE OF APPLICATIONS AND ALLOWS FOR MORE DIRECT, INTUITIVE USE. POWERFUL SOFTWARE DEVELOPMENT TOOLS ARE IN TURN PERMITTING EVEN MORE SOPHISTICATED APPLICATIONS TO BE DEVELOPED. THE GROWTH OF COMPUTER NETWORKING IS PROVIDING ARTISTS WITH NEW MEANS OF COMMUNICATION.

IN GENERAL, PEOPLE ARE FINDING MORE AND MORE WAYS TO APPLY SMALL COMPUTERS IN THE CREATIVE ARTS. NEW DEVELOPMENTS IN MACHINE INTELLIGENCE, PATTERN RECOGNITION AND HUMAN-COMPUTER INTERFACES ALLOW FOR MORE INTUITIVE AND INTERACTIVE USE OF THE COMPUTER.

THE PAPERS IN THIS YEAR'S PROCEEDINGS REFLECT THESE TRENDS, BOTH DEMONSTRATING AND PROPOSING HOW COMPUTERS CAN BE USED IN VIDEO, DANCE, WEAVING, MUSIC COMPOSITION AND SYNTHESIS, LASER SHOWS, INTERACTIVE SCULPTURE, VIDEOTEX, ANIMATION, AND PAINTING.

TABLE OF CONTENTS

HISTORICAL NOTES.	v
PREFACE	vi
The Microcomputer as a Musical Instrument R.H. Lord	1
Noise in Real Time Digital Sound Generation A.C. Ashcraft, F.H. Covitz, and K.A. Sproul	5
Sonic Set Theory: A Tonal Music Theory for Computers (Tools for Algorithmic Composition with an Algorithm for Chord Selection for Dessert). L. Spiegel	15
Merging Event Lists in Real-Time. J.W. Mauchly	23
Automatic Computer Composition of Bluegrass Tunes M. Keith	29
Graphics Software Interaction Using the Apple Computer. A.C. Flynn	35
Microcomputer Drawing W.J. Kolomyjec	39
Analytical Computer Art J. Jacobson	47
Pantomation: A System for Position Tracking. T. DeWitt and P. Edelstein	61
Real Time Animation Techniques With Microcomputers. F. Dietrich	71
An Application of a Microcomputer as an Interactive Art Medium. G.K. Shortess	75
An Artist's Computer System W. Wright	79
Videotext as Personal Expression. W. Richmond and S. Rubin	87
Computer Painting with Rodin. M. Nahas and H. Huitric	95
Computer Graphics and Woven Fabric Design L. Giannitrapani	105
Computers and Choreography. C.M. Stadler	107
Designing a Recursive Framework for Evolutive Poetry. N.J. Davison	111
Laser Shows and the Use of Computers. A. Jackson	115
AUTHOR INDEX.	119

THE MICROCOMPUTER AS A MUSICAL INSTRUMENT

by Richard H. Lord

UPWARD CONCEPTS
Bennett Rd., Durham, NH 03824

ABSTRACT

This paper briefly surveys types of music synthesizer hardware. It then examines plug-in cards for the APPLE computer and explores the Mountain Computer music system and its potential as a performance instrument. Types of music keyboards are briefly discussed.

INTRODUCTION

The recent evolution of microcomputer technology has made many exciting ideas possible. The synthesis of music has benefitted greatly from these advances. Many commercial synthesizers now employ digital circuitry in their design. Even more exciting is the incorporation of the personal computer into the synthesizer architecture, allowing extensive control of every aspect of the synthesis mechanism.

These new approaches permit anyone to experiment with the quality and timbre of the created sounds in very sophisticated ways, either for the authentic imitation of conventional instruments or the creation of entirely new sounds.

OVERVIEW

Before considering the nature of the newer synthesizers, let us briefly review the traditional approaches. This paper will not attempt a detailed description of these techniques since many sources of this information already exist.

The first real music synthesizers were pipe organs, and their history goes back a few thousand years. One cannot overlook the fact that the finer examples of pipe organ design provide the musician with a marvelous palette of tone color which was not easily imitated by the electronic counterparts that began to appear in the 1950's. The electronic organs did introduce some basic concepts of electronic sound production which provide the fundamentals of modern instruments. Most early electronic organs provided a waveform signal for each pitch the instrument could sound. These waveforms were usually square or sawtooth forms containing a rich family of harmonics. The keyboard switched these outputs on or off and the resulting output was filtered to create string or flute sounds. The technique of starting with lots

of harmonics and filtering the output to get a desired timbre is called subtractive synthesis. This method appears in classical instruments as well as electronic ones. The reed families, in particular, generate a raw, raspy buzz which is modified by filtration.

As electronic organs became more sophisticated, the harsh keyboard switches were replaced with simple envelope generators which softened attack and created sustain after the key was released. A need to control this envelope more precisely, and the desire to modify pitch and filter parameters as well, created the market for the analog synthesizer. Of course, the economics of creating a full analog synthesizer for each key of an organ was (and still is) prohibitive, so early synthesizers did not play very many notes at one time. Complex multi-track recording techniques were used to painstakingly assemble full orchestrations such as the work of Carlos or Tomita. These instruments were frequently provided with patch panels so that more exotic sounds could be created by interconnecting the oscillators, filters and envelope generators in unusual combinations.

Meanwhile, another form of synthesis was beginning to gain a foothold. Way back in the 1700's, mathematicians had discovered that complex sounds could be broken down into a series of sinusoidal waves. Adding these sine waves back together would create the original waveform. The Hammond organ made use of a limited form of this additive synthesis. Attempts to imitate (and thus understand) complex instruments such as the trumpet, showed that a series of sine waves could indeed reproduce these sounds. The trick was that each sine wave had its own envelope, and some of these changed pitch during the attack or decay. This looked suspiciously like the earlier problem of providing many channels of analog synthesizers, except now each note would require many channels. This clearly limited this technique to academic music research labs, where complex machines could be built.

As all this was happening, digital computers started to become fast, efficient and available. Some researchers recognized that both additive and subtractive synthesis could be performed mathematically by a digital computer. These calculations were too complex to be used for direct creation of sound, but an output file could be created which could then be converted directly to analog signals. All the complexities of multiple

notes of multiple sine waves could all be added together into a single digital data stream. This is the ultimate technique for producing the most subtle control of the final output. However, it requires storage of at least a million bytes of information for each minute of audio output. It also takes a great deal of computer time to create this file, so that live performance or interactive control is impossible. The lack of interaction tends to limit the usefulness of this method for exploring new sounds.

MICROCOMPUTER SYNTHESIS

Recent advances in large-scale circuit integration have allowed both computers and specialized controllers to be developed. Manufacturers of electronic musical instruments have begun to recognize this potential and start incorporating these devices into their designs.

The PROPHET 5 analog synthesizer by Sequential Circuits is an example of the adaptation of an analog synthesizer. A Z-80 microcomputer and battery-powered memory allow 120 complete presets to be stored in this machine permanently (or replaced with 120 more, stored on tape). The microcomputer sets up the envelope and filter patches and sets up all the analog parameters, as well as scanning the keyboard and providing control voltages. Five complete channels are supplied, and the microcomputer selects five of the keys held down. The synthesizer itself, is entirely analog with conventional voltage-controlled oscillators, filters, amplifiers and envelope generators. In this product, the microcomputer is not really involved in the production of the sound.

An entirely different approach is employed by instruments such as the CASIOTONE 202. Here, custom integrated circuits generate the sound by direct digital synthesis. The manufacturer carefully created 59 different presets that can be selected by the performer. No mixing or modification of these voices is possible, but the variety and quality of the presets is very suitable for a performing instrument. In this synthesizer, two channels are provided to supply the transient and steady-state aspects of the instrument sound. Each channel has a special IC which scans the keyboard, selects the eight most recently played keys, and provides complete additive waveform synthesis and envelope generation for these eight notes, summing the result and sending the digital output to a 14-bit digital-to-analog converter (DAC). Each channel then is passed through a simple analog filter and then mixed to form the final output. The manufacturer has not released much information on these custom IC's, but they appear to be doing some very powerful computation.

Surprisingly few high-quality synthesizers have appeared for personal computers. There are 8-bit DAC's available for most computers. Experimenters like Hal Chamberlin have gotten surprising amounts of music from these, but the dynamic range and frequency response of this method are very limited and the microcomputer is very busy creating samples for the DAC and has no time to scan keyboards.

The APPLE computer has fared better than the others, with two rather interesting types of synthesizer cards available. The ALF synthesizers are programmable pitch generators with amplitude controls. Two different cards are available. One provides 9 channels with 16 amplitude levels and the other produces 3 channels with a more precise 256 amplitude levels. Pitch is selected by dividing the system clock by a programmable number. The only waveform possible is a square wave. This can be given different characteristics by changing the envelope, but not much variety is possible. This synthesizer does provide a convenient note editing system that makes it rather pleasant for conventional composing, where the desire is to quickly hear examples of counterpoint or harmony. The entry system is by far the simplest to use. It allows repeats and transposed repeats and can create many interesting effects. It is rather like programming a player piano though, since no keyboard input or tonal variation is provided.

The second synthesizer card for the APPLE is the Mountain Computer 16-channel digital music system. This is actually two cards that fit in adjacent slots of the computer. The system is a true digital additive synthesizer that allows musical instruments to be precisely defined. Each of the 16 channels can have its own waveform and envelope profile. In addition, pitch and amplitude can be given a complex attack profile. A single instrument could use all 16 oscillators, though usually two are assigned to each voice. The system provides 30,000 samples per second for each oscillator so that the sound quality is quite high. Compositions can be entered and edited separately from instrument definitions and new orchestrations can be tried easily.

The software delivered with this synthesizer does not allow input from a music keyboard. Two manufacturers supply keyboards and software to work with the Mountain Computer system. Both of these systems allow the user to define preset instruments to be played from the keyboard. Sequences of notes can be recorded and played back. The keyboard can be played in a different voice while the recorded notes are played. Both systems allow the selection of different tunings than the equal-tempered scale. The Passport Designs system allows FM effects in the instrument definitions. The Alpha-Syntauri system is available with a velocity-sensing keyboard that can be linked to various parameters. Both systems play up to eight notes at once and assign two oscillators to each note with simple ADSR envelopes for each oscillator.

PROBLEMS

While this synthesizer and its software can provide much enjoyment and useful experimentation, the present systems fall slightly short of the mark. The size of the scores that can be edited by the Mountain Computer software is quite limited. Parts can be compiled and merged, but even then, there are few complex scores that can run more than a few minutes. It is not presently possible to automatically bring in a new

movement from disk. In fact, the Mountain Computer software (written in a language called XPL) frequently pauses during simple editing operations so that new pieces of the program can be swapped into the APPLE's memory.

Instrument definition can also be a problem. As mentioned earlier, the three different software systems have somewhat different instrument specifications. Also, the output filter of the Mountain Computer card does not provide a very satisfactory frequency response curve, limiting the quality of the instrument definitions. It is also very difficult to generate random noise as a component of the sound. It is simple to generate a random wavetable which is periodically repeated, but the ear has incredible ability in its power to sense periodicity. The result is a harsh buzz, not white noise.

The frequency response and noise generation problems can be solved with hardware. A card can be added to provide better output circuitry and to generate random numbers when addressed in ROM address space. Ideally, this card could also perform keyboard scanning and provide a digitally programmable analog filter for special effects that cannot readily be created with digital synthesis. Larger composition files could be possible if someone were willing to code the software more efficiently.

IN CONCERT

The possibilities of digital synthesis for a performance instrument are very exciting. Existing software permits a performer to generate exotic instruments and call them into action with the touch of a single key. Bass line accompaniments can be recorded and easily called in.

Much work needs to be done on the design of keyboards. At present, velocity input (from the timing between switch closures) is the only parameter available on the Alpha-Syntauri. Some commercial synthesizers are making use of key pressure as well. So far, nothing approaches the control of a piano. The subtlety of tone that a violinist can impart to the string is very much over the horizon. The pipe organist must be content with on/off control, but pianists universally complain of the insensitivity of synthesizer keyboards. If each key could sense velocity, position, and force in each of two directions and these parameters could be linked into the instrument definitions, many possibilities might exist. If authentic "feel" were part of the requirement, it is even possible to provide force feedback through solenoids mounted under each key and modulated by a keyboard micro-computer.

SOME CONCLUDING THOUGHTS

There appear to be at least three directions of experimentation suggested by the available microcomputer synthesis systems. For those interested in composition, these programs allow scores to be entered and edited with some of the flexibility of a word processor. The results can be orchestrated and heard immediately. Runs of 64-th notes beyond the dexterity of the most

accomplished musicians can be played on this system. If your interests are centered on the analysis and imitation of musical instruments, these systems allow you to construct complex instrument definitions and hear them as you modify them. These instruments can be saved on disk and called up at will. The performing musician can create many new effects and can record accompaniment to enhance the performance.

The Mountain Computer synthesizer cards have far more capability than has been exploited by the present software. Sixteen oscillators may be a limitation for performance with exotic instruments, but much can be done with this hardware and many more powerful digital synthesizers are likely to become available. While performance instruments may still be in their infancy, anyone interested in the design of keyboards, digital hardware or software in connection with these new instruments should find many stimulating possibilities available.

NOISE IN REAL TIME DIGITAL SOUND GENERATION

A.C. Ashcraft, F.H. Covitz and K.A. Sproul

A B S T R A C T

This paper discusses the most important software sources of digital noise in sounds generated by the cyclic table look up, Digital to Analog Converter (DAC) method of sound synthesis. These sources are: 1) Quantization, 2) Truncation, and 3) Interpolation. The relative importances of these sources are discussed as they are affected by system parameters such as the number of bits available to represent amplitudes in the waveform tables, the number of bits which can be converted by the DAC, the number of entries in the waveform tables, whether or not interpolation is possible, and the accuracy of the interpolation if it is possible. In this paper we propose a partly hardware, partly software technique for doing linear interpolation in real time synthesis. We describe the residual digital noise of this technique, as well as the minimization of this noise by trade-offs between quantization noise, interpolation noise, memory usage and sampling frequency.

SOUND SYNTHESIS

The digital sound synthesis procedure which is the basis for the following discussion is the cyclic table look-up/DAC method. Our first implementation was with a 6502 machine language program which pre-computes sets of 256 byte waveform tables by additive Fourier synthesis. It then uses these tables for real time generation of waveform amplitudes by cyclic table look-up. A double precision frequency increment is added to a double precision table pointer for each voice in order to maintain pitch accuracy, however only the integer part of the table pointer is used to look up the waveform amplitudes (ie, no interpolation). An 8-bit DAC, a low pass filter, and an audio amplifier are the only sound generation hardware required. This program plays four part instrumental music with a sampling frequency of 8.8 khz. The program and its evolution have been described in detail in references 1-3.

An improved implementation is de-

scribed in the present article. It consists of a 68000 machine language program which runs on the MTU-130 computer equipped with MTU's 256k memory board containing a Motorola 68000 16-bit microprocessor. The added hardware for sound synthesis consists of two 12-bit multiplying DAC's, two 8-bit DAC's and a low pass filter. This new system will also play four part instrumental music in real time, however, it has a sampling frequency of 22 khz and does linear interpolation, resulting in a greatly improved high frequency response and signal to noise ratio compared with our earlier implementation.

DIGITAL NOISE

1) Quantization

The most fundamental software source of noise in digital sound synthesis is due to quantization. This noise results from the representation of a continuous analog variable, such as a waveform amplitude, by a binary integer having a precision limited by the number of bits. Quantization noise is quite serious in real time sound synthesis using the cyclic table look-up procedure because time and memory constraints generally make multiple precision operations out of the question. Until recently, all microcomputers in general use had an 8-bit data bus. This has limited single precision operations to an accuracy of $\pm 1/2$ part in 256. The signal to noise ratio expected from this inaccuracy is 512:1. Expressed in dB this is:

$$s/n = 20 \cdot \log(512) = 54.2 \text{ dB}$$

In general, the signal to noise ratio due to quantization is given by the formula:

$$s/n = 20 \cdot (n+1) \cdot \log(2)$$

Or approximately:

$$s/n = 6n+6$$

where n is the number of bits used to represent the number.

2) Truncation

The noise figures arrived at in the foregoing discussion are misleading if one goes no further with the analysis. Even if individual values in the waveform tables are accurate to the specified number of bits, and the DAC used has sufficient precision to reproduce these values as analog signals, the final result will be inaccurate if the fractional part of the table pointer is ignored. This error, called truncation error, results because successive entries in the tables can easily differ by more than one unit, particularly if higher partials are present. If one ignores the fractional part of the pointer and takes the lower of the two table entries on either side of the true pointer location, an erroneous waveform amplitude can result which differs from the true value by many times the quantization error.

Consider the rather brief 8-bit sine wave in table I and its graphical representation in figure 1. This table consists of sixteen equally spaced values scaled to cover the full amplitude range of 0 to 255. The amplitude returned from this table as a result of ignoring the fractional part of a table pointer having the value of 0.4 is 128 (as if the pointer had been 0.0), rather than the correct value of 148. This is the maximum error which can result from the use of this table. If the pointer had been 4.4, a value of 255 would have been returned rather than the correct value of 253.

Table I

Pointer	Amplitude	Pointer	Amplitude
0	128	8	128
1	177	9	79
2	218	10	38
3	245	11	11
4	255	12	1
5	245	13	11
6	218	14	38
7	177	15	79

The calculated RMS signal to noise ratio (s/n) resulting from truncation of the fractional part of the pointer is 19 dB for this size sine wave table, a far cry from the 54 dB expected from quantization at the 8-bit level. If the table had contained a more realistic number of entries, say 256, the s/n would have been 43 dB. A sine table with 1024 entries results in an s/n due to truncation of 55 dB, which when combined with 8-bit quantization results in an overall s/n of 48.5 dB. Table II shows that each quadrupling of the table size results in an improvement of the s/n due to truncation by about 12 dB.

Table II

Number of Table Entries	Truncation s/n, dB	Overall s/n, dB
16	19.2	19.1
64	31.1	30.5
256	43.0	40.9
1k	55.1	48.6
4k	67.2	52.5
16k	79.1	53.7
infinite	infinite	54.2

However, the overall effect, including quantization noise, approaches one of rapidly diminishing returns for increased use of memory for waveform tables larger than 1k. These RMS s/n calculations were done using a BASIC program published by H. Chamberlin on p. 394 of reference [4]. The program is also capable of RMS s/n calculations on arbitrary waveforms calculated by Fourier synthesis. In the reference, results are given for a sine wave as well as a complex wave containing 2nd, 3rd, 5th, 8th, 11th, 14th, and 17th partials of equal intensity. For this report, we have used four representative waveform types to cover a wider range of timbre effects:

Type 1:

$$A = \sum_{n=1}^m \sin(nx)$$

Type 2:

$$A = \sum_{n=1}^m 1/n \sin(nx)$$

Type 3:

$$A = \sum_{n=1}^{(m+1)/2} \left(\frac{1}{2n-1} \right) \sin([2n-1]x)$$

Type 4:

$$A = \sum_{n=1}^{(m+1)/2} \left(\frac{1}{2n-1} \right)^2 \sin([2n-1]x)$$

where n is the partial number and m is the maximum partial number. Figure 2 shows the appearance of a few waveforms from each type.

The type-1 waveforms are the richest in timbre and the hardest to generate accurately if one ignores the fractional part of the wave table pointer. The type-4 waveforms have the softest timbre and are the easiest to approximate. Type-3 waveforms approach the square wave as n becomes large and type-2 waveforms approach the sawtooth waveform. The type-2 waveforms are probably the most typical of those needed to create musical instrument sounds, and are used extensively in the discussion and examples to follow.

Figure 3 shows how truncation noise varies with the table size and timbre of the waveform. The results for the type-2 waveform are fairly accurately fit by the equation:

$$s/n = 19.9 \log(\text{size}) - 7 \log(m) - 4.7 \text{ (dB)}$$

The s/n frequency response of the type-4 waveforms is almost flat, and that of the type-1 waveforms rolls off more rapidly than that of the type-2 and -3 waveforms, between 3 and 5 dB/octave. The type-3 waveform response (not shown) is very similar to that of the type-2, about 2 dB/octave roll-off.

Figure 4 shows the combined effects of 8-bit quantization and truncation noise for the type-2 type waveforms as a function of table size and maximum partial number in the Fourier series. We are faced with diminishing returns for building larger and larger waveform tables. Note that the 256 byte example represents the performance of the program of reference [3].

3) Interpolation

The only way out of this difficulty is to face up to the necessity of interpolation between successive entries in the waveform tables, making use of the fractional part of the waveform table pointer. For the sake of computational simplicity, let us consider linear interpolation.

If amplitudes A_1 and A_2 are successive table entries in a waveform table, and the table pointer is between them, ie, the integer part points to A_1 and the fractional part of the pointer, F , is not zero, then the linearly interpolated amplitude A is:

$$A = A_1 + F(A_2 - A_1)$$

Implicit in this formula are two table look-up operations (with an index increment between them) to get A_1 and A_2 , a subtraction, a multiplication, and finally an addition rather than the single table look-up required for the truncation procedure used in the computer program of reference [3]. The subtraction may be avoided if a separate table of delta A values (D) is precomputed and stored in memory for access by the same index used for A_1 :

$$D = A_2 - A_1 \text{ , precomputed}$$

$$A = A_1 + F \cdot D$$

This operation still requires two table look-up operations, a multiplication and an addition. Let us assume for the moment, however, that the computations can be

accomplished quickly enough for real time music synthesis. What is the gain in s/n as a result of linear interpolation?

The RMS noise calculation program in ref. [4] was modified to return waveform amplitude values obtained by simple linear interpolation rather than truncation. The results obtained for a type-2 waveform set are shown in figure 5. 8-bit quantization noise corresponding to a s/n of 54.2 dB has been added to the calculated values to make easy comparison with figure 4 possible. Note that if linear interpolation in a table of only 256 entries is used, the results are as fully accurate as if a table of 16k entries had been used with truncation. Similar results quantized for a 12-bit amplitude are also shown. The results of linear interpolation in waveform tables of reasonable size are obviously good enough for 12-bit audio purposes. Quadratic and higher forms of interpolation are not necessary.

SOFTWARE INTERPOLATION

1) 6502 Software

If a 6502 microprocessor is used, the additional indexed table look up can be combined with the addition and requires only 5 μ sec. However the multiplication in a machine code algorithm could take as more than 100 μ sec, and worse yet, the time would be variable, depending upon the bit pattern of the factors. For a four voice system the sample time will therefore increase from the reasonably effective 114 μ sec of the program of ref. [3] (which has a Nyquist frequency of 4.4 kHz) up to a ponderous 500 μ sec or so. This would result in a decidedly lo-fi Nyquist frequency of 1 kHz even if the variable loop time were fixed using a FIFO buffer to give a stable sampling rate. The high frequency response could be partially restored by going to a single voice system, but this is not musically interesting enough to support the effort it would entail.

For delayed playback systems, neither the variability nor the reduced speed are of consequence, and just this approach was used for the demonstration given by H. Chamberlin at the IEEE sponsored 1981 Symposium on Small Computers in the Arts [5]. A 6502 machine language program running in a 1 MHz Rockwell AIM was used to calculate 12-bit samples at a rate of about 500/sec on the average. At this rate, time was available for not only interpolation between table entries, but also interpolation between waveforms, and other effects such as reverberation. The samples were stored on floppy discs as they were generated for later playback

with a very fast disc operating system, a FIFO, and low pass filter at a sample rate of 25 kHz.

Although the performance of the above system was excellent, the challenge still remained: could the improved s/n which results from interpolation be realized in a real time music synthesis system in a small computer? A member of the new generation of 16-bit microprocessors may provide the increase in computing power necessary: the Motorola 68000.

2) 68000 Software

The 68000 is fast. Consider a typical software operation such as "look up a value from a table, and store it to a port". It looks like this in 6502 assembler code:

```
LDA (Pointer),Y ; "Pointer" is low byte
                  ; of 16 bit zero page
                  ; pointer, Y = 0
STA Port          ; Absolute address of
                  ; output port is used
```

and takes 9 microseconds in an ordinary 1 MHz 6502. The same operation in 68000 assembler code takes only one instruction:

```
MOVE.W (A0),(A1) ; "Pointer" is stored
                  ; in register A0,
                  ; "Port" in A1
```

This instruction takes 1.5 microseconds in an "ordinary" 8 MHz 68000. Note also that the 68000 MOVE.W instruction moves a 16-bit word from a table of 16-bit values to the port, not an 8-bit byte (as was the case for the 6502). A MOVE.L instruction would have moved a 32-bit long word from the table to a 32-bit port in 2.5 microseconds. In addition to this speed-up of ordinary operations, the 68000 has a multiply instruction which takes a maximum of 8.5 microseconds to multiply two 16-bit numbers and generate a 32-bit product, and a divide instruction which takes 17 microseconds. Unfortunately, the exact timing of these instructions is a function of the bit pattern in their operands.

The following code fragment is an example of what might be written in 68000 assembler language to implement linear interpolation in a sound generation program. The program assumes the existence of 256 entry tables of 32 bit long-words, each entry having the waveform amplitude in the high order word and the difference to the next entry amplitude in the low order word. INCR is a frequency increment which is added to the permanent voice pointer in D1. The integer part of the waveform pointer eventually ends up in A0. D0 and D7 are used as scratch pad registers, and A0 is used as an indirect

address register for the table look-up. Both the amplitude and the difference to the next entry are moved to D0 by this operation. After the fractional part of the pointer (bits 10-18) is moved into D7, it can be multiplied by the difference contained in the low order word of D0, and finally added to the amplitude for storage to the 12-bit DAC. The cycles given are 8-MHz clock cycles. Each voice requires a maximum of 198/8 or 24.75 μ sec. per sample.

```
ADDI.L #INCR, D1 ;Advance pointer 16
ANDI.L #FBFFFFFF,D1 ;Mask off bit 26 16
MOVE.L D1, D0 ;Put in Scratch-pad 4
ANDI.L #03FC000F,D0 ;Mask off fract. 16
SWAP D0 ;Swap words in D0 4
MOVE.L D0, A0 ;Put pointer in A0 4
MOVE.L DISP(A0),D0 ;Ampl.,slope in D0 16
MOVE.L D1, D7 ;Frac. in D7 4
ASR.L #2, D7 ;Shift to lo-word 12
ANDI.L #0000FF00,D7 ;Mask off lo-byte 16
MULS D0, D7 ;Multiply <70
ADD.L D0, D7 ;Get corr. Ampl. 8
MOVE.L D7, (A1) ;STORE TO PORT 12
Total cycles/voice=<198
```

A 4-voice program using the above fragment would have a Nyquist frequency of 5kHz, not much higher than the 6502 program of ref.[3]. However a more serious flaw is the lack of predictability of the cycle time caused by the MULS instruction. Such a program could only be used in a delayed playback system or in a system equipped with a First In First Out (FIFO) buffer having enough capacity to smooth out the timing irregularities of the program.

HARDWARE INTERPOLATION

1) Software Considerations

Since additional hardware seems to be inevitable, what about the possibility of doing the interpolation itself in hardware? In the 68000 program of the preceding section, all of the quantities needed for interpolation appeared in registers or in a memory move operation. The fractional part of the table pointer was in data register D0, and both the waveform amplitude and the difference were retrieved together with the "MOVE.L DISP(A0), D0" instruction. Suppose that the MOVE.L instruction were directed to store the 32-bit long word directly out to a port rather than to D0, and that the fractional part of the pointer were similarly moved out to another port. Latch hardware could insure simultaneous delivery of all the information to a group of multiplying DAC's and op-amps designed to perform the interpolation math. The 68000 program fragment accomplishing this

is considerably simpler than the earlier one, and much faster. In this program, each voice would require only 10 μ sec. per sample with an 8 MHz 68000. A four voice play loop based upon this fragment would have a sampling frequency of 24.39 kHz:

```

ADDI.L #INCR, D1      ;Advance pointer      16
ANDI.L D7, D1         ;Mask off bit 18      8
MOVE.L D1, (A2)       ;Frac. to PortA      12
MOVE.L D1, D0         ;Pointer in Scratch   4
ANDI.L D6, D0         ;Mask off Frac.      8
SWAP D0               ;Swap words in D0     4
MOVE.L D0, A0         ;Put Pointer in A0    4
MOVE.L DISP(A0), (A1);Amp,Slope to PortB 24
                        ;Total cycles/voice=80

```

2) Interpolation in the Presence of Several Sources of Quantization Noise

Before considering the hardware requirements for interpolation in detail, let us first consider the s/n ratio consequences of analog interpolation based upon signals each of which contain quantization noise. Both the amplitudes in the waveform tables and their associated differences are limited in precision to 16 bits by the 68000 and probably to 12 bits by considerations of cost and practicality. Similarly, the fractional part of the waveform pointer has a limited precision, possibly as low as 8-bits. If the program used earlier for analysis of s/n due to the inaccuracies of linear interpolation were modified to round off all values to their expected accuracy, the RMS s/n could be directly determined. The BASIC program listed in the appendix was designed to do this. Note that 100 trials were used in most of the results reported here rather than 1000 trials as recommended in ref [4]. A comparison of selected results based on 1000 trials showed that 100 was adequate. See Figure 6.

The program allows easy study of the trade-offs which must be made between table size and the number of bits allocated to the parameters which are to be brought together in the DAC circuitry. In the figures the table size and number of bits allocated to the parameters are given in the form S/a/d/f, where "S" is the number of entries in the waveform tables, "a" is the number of bits in the amplitude, "d" bits in the difference, and "f" bits in the fractional part. Thus 256/12/12/8 means a 256 entry table of 12-bit amplitudes and differences, with an 8-bit fractional part.

Note that if the sum of a and d is 16, then the table can be made up of composite 16-bit words, taking only half the space for given number of entries, and the MOVE.L DISP(A1), (A0) can be changed to

a MOVE.W instruction, saving 1 μ sec. per voice. If $d < a$, the program assumes that the least significant bit (LSB) of each number has the same significance level. When differences between two successive table entries exceeds the range of a number having b bits, the table value of the difference is limited at the value 2 raised to the power d-1. This introduces noise, but only for waveforms containing harmonics above a certain point. The alternative procedure (scaling the difference values so that they could cover the full range of amplitude differences, even though with less precision), results in the quantization noise of the difference values dominating the overall s/n. The latter procedure results in more noise over the whole sound spectrum and is thus less desirable.

Let us investigate this space saving possibility first. Figure 7 shows a graphical comparison of the s/n performance of three members of the 256/n/16-n/8 family. The 256/8/8/8 has the best overall performance, although for waveforms containing only the 1st, 2nd and 3rd partials, the 256/10/6/8 system is better, limited only by the 10 bit quantization. The 256/12/4/8 system is useless, even worse than the 256/8-truncation case. It fails because of the insufficient range of a 4 bit number (+8) to represent successive differences in a 12 bit table of waveform amplitudes.

Figures 8 and 9 show the corresponding members of the 512 and 1024 entry families. 10-Bit performance can be obtained from the 512/10/6/8 system up to the 9th partial in type-2 waveforms (sawtooth). The 1024/10/6/8 system is even better, showing almost flat s/n response out to the 17th partial for the sawtooth waveform. Neither the 512/12/4/8 nor the 1024/12/4/8 systems are useful. 12-Bit performance is unattainable until table size is much bigger, at least 8k.

If we abandon the hope of using 16-bit tables, we find that 10-bit performance is easy to achieve with small numbers of table entries, and that 12-bit performance is possible. Figure 10 shows the results of our analysis of the n/12/12/12 cases where n = 256, 512, 1024, and 2048. For clarity's sake only the results for the type 2 (sawtooth) waveforms are shown. Also shown for comparison purposes are the 256/8/8/8 and the 256/8-truncation cases. Note that the 256/12/12/12 system does better than 10-bit quantization out to the 7th partial. The s/n rolls off at about 8 dB/octave above that. The 512 entry case does as well out to the 20th partial, and the frequency response of the 1024 entry case is practically flat, approaching

12-bit quantization out to the 20th partial

The 256/12/12/12 case is a very attractive one to consider for an improved digital sound generation system based on a 68000 processor on the MTU 256k memory board:

a) It is conservative of memory Each waveform requires 1024 bytes of memory. If 32 waveforms were allocated to each instrument for dynamic timbre effects, 256k allows 7 complete instruments with 32k left over for program and song data files. The 512 entry system would significantly restrict the range of instrumental effects which could be achieved while only resulting in a modest incremental improvement in s/n.

b) It represents a significant improvement in performance The 15 to 30 dB improvement in s/n and the threefold improvement in Nyquist frequency is a very significant improvement in musical capability compared with our earlier efforts [3].

A few additional runs on the s/n program showed that 8 bits of information was sufficient for the fractional part. The 256/12/12/8 system is cheaper to implement, needing one less 12 bit DAC, and the performance shown in figure 11 is more than adequate. This is the system we decided to try.

3) Interpolation DAC Circuit Design

The circuit is designed around a set of memory mapped latches feeding data to 12-bit multiplying DAC's (AD7541) for the waveform amplitudes and table difference values (A1 and D), and 8-bit multiplying DAC's (AD7523) for the fractional part of the waveform pointer and overall gain control (F and G). Operational amplifiers are used to convert the 12-bit DAC outputs to bipolar voltages and to convert the 8-bit DAC outputs to unipolar voltages for the fractional part of the waveform pointer and the gain control. One of the 12-bit DAC's forms the product of the fractional part and the table difference, using the analog voltage derived from the 8-bit fractional part DAC as its reference. The reference voltage of the 8-bit fractional part DAC in turn is supplied by the unipolar analog output of the 8-bit gain DAC, which also supplies the reference to the 12-bit amplitude DAC. Another op.amp. sums the analog voltages corresponding to the waveform amplitude and the product of the fractional part and the table difference, giving the interpolated amplitude. The voltage emerging from the summing amplifier depends upon the digital outputs A1, D, F,

and G from the 68000 program in the following way:

$$V = -.5 \left[\frac{(A1-2048) \cdot G}{2048 \cdot 256} - \frac{(D-2048) \cdot F \cdot G}{2048 \cdot 256 \cdot 256} \right] \cdot V_r$$

Except for the minus sign between the terms, the above formula has the correct form for interpolation. If, when the waveform tables are computed, one calculates the value stored in the low order 16 bits of each 32-bit table entry by subtracting the next higher table amplitude from the current one (let $D' = A1-A2$ rather than $A2-A1$), the resulting difference, $D' = -D$, will give correct interpolation using the above formula.

The circuit diagram for the interpolation DAC is shown in figure 12. The output of the final summing amplifier is fed into a sharp low pass filter having a flat response out to 9.5 kHz. This filter is described on p 377 of reference [4].

BIBLIOGRAPHY

- [1]. H.Chamberlin, "A Sampling of Techniques for Computer Performance of Music", BYTE Magazine, Sept., 1977
- [2]. H.Chamberlin, "Advanced Real-Time Music Synthesis Techniques", BYTE Magazine, April, 1980
- [3]. F.H.Covitz and A.C.Ashcraft, "Analysis and Generation of Complex Sounds Using Small Computers", p. 33, Proceedings of the IEEE Symposium on Small Computers in the Arts, Nov. 20-22, 1981.
- [4]. H.Chamberlin, "Musical Applications of Microprocessors", Hayden Book Co. inc., Rochelle Park, N.J., 1980.
- [5]. H.Chamberlin, "Delayed Playback Music Synthesis Using Small Computers", p. 27, Proceedings of the IEEE Symposium on Small Computers in the Arts, Nov. 20-22, 1981.

APPENDIX

BASIC Program for Calculation of Signal to Noise Ratio In the Presence of Multiple Sources of Quantization Error

```

10 REM INTERPOLATION ERROR
20 DIM J(4),Q(4)
25 DIM D(4,21)
30 REM WAVE TYPE DEFINITIONS
32 LET J(1)=1
33 LET Q(1)=0
34 LET J(2)=1
35 LET Q(2)=1
36 LET J(3)=2

```



```

37 LET Q(3)=1
38 LET J(4)=2
39 LET Q(4)=2
110 REM #TRIALS
112 LET R=100
120 REM #ENTRIES IN TABLE =N
122 FOR B=8 TO 12
124 LET N=2^B
130 REM BITS TO DAC =DAC
132 FOR D1 =8 TO 12 STEP 2
134 LET X1=2^(D1-1)
140 REM BITS TO SLOPE =D2
141 LET D2=16-D1
142 LET X2=2^(D2-1)
143 REM BITS TO FRACT. PART = D3
144 LET D3=8
150 PRINT N;"//";D1;"//";D2;"//";D3
155 LET D3=D3^8
700 LET P1=2*3.14159
800 LET P2=.314159
1005 REM SELECT WAVE-TYPE
1006 FOR W=1 TO 4
1010 LET J1=J(W)
1011 LET Q1=Q(W)
1012 LET W1=W*(W)
1020 FOR H1=1 TO 21 STEP 4
1028 REM FIND MAXIMUM VALUE OF WAVEFORM
1030 LET S1=0
1033 LET N1=.5
1034 LET N2=2/N
1035 FOR T=0 TO N1 STEP N2
1040 GOSUB 2000
1042 LET S=ABS(S)
1050 IF S<=S1 THEN 1055
1051 LET S1=S
1055 NEXT T
1060 F=X1/S1
1110 REM M1=MEAN OF IDEAL
1120 LET M1=0
1121 LET M2=0
1122 LET T1=0
1130 FOR I=1 TO R
1140 LET T1=T1+P2
1150 IF T1<1 THEN 1160
1155 LET T1=T1-1
1160 LET T=T1
1161 GOSUB 2000
1170 LET S=S*F
1180 LET M1=M1+S
1190 LET S1=S
1191 REM S1 IS IDEAL AMPLITUDE
1195 REM LINEARLY INTERPOLATED AMPLITUDE
1200 GOSUB 3000
1210 LET M2=M2+(S1-S)
1220 NEXT I
1300 LET M1=M1/1000
1310 LET M2=M2/1000
1400 REM V1 IS SIGNAL POWER, V2 IS NOISE
1410 LET V1=0
1411 LET V2=0
1412 LET T1=0
1420 FOR I=1 TO R
1430 LET T1=T1+P2
1440 IF T1<1 THEN 1450
1445 LET T1=T1-1
1450 LET T=T1
1460 GOSUB 2000
1461 LET S=S*F

```

```

1470 LET V1=V1+(S-M1)*(S-M1)
1480 LET S1=S
1481 GOSUB 3000
1510 LET V2=V2+((S1-S)-M2)^2
1520 NEXT I
1600 LET D(W,H1)=4.3429*(LOG(V1)-LOG(V2))
1605 LET D(W,H1)=INT(10*D(W,H1)+.5)/10
1800 NEXT H1
1810 NEXT W
1812 FOR W=1 TO 4
1813 PRINT W;;
1814 FOR H1=1 TO 21 STEP 4
1815 PRINT D(W,H1);
1816 NEXT H1
1817 PRINT
1818 NEXT W
1820 NEXT D1
1830 NEXT B
1999STOP
2000 REM COMPLEX WAVEFORM GENERATOR
2050 LET A=P1*T
2060 LET S=0
2100 FOR H=1 TO H1 STEP J1
2110 LET S=S+(1/H)^Q1*SIN(H*A)
2120 NEXT H
2200 RETURN
3000 LET T=(INT(N*T1)/N)
3010 LET T2=INT(N*T1+1)/N
3020 LET T3=INT(.5+D3*(T1-T)/(T2-T))/D3
3040 GOSUB 2000
3050 LET S2=INT(.5+S*F)
3060 LET T=T2
3070 GOSUB 2000
3075 LET S=INT(.5+S*F)
3080 LET D9=S-S2
3090 IF D9<-X2 THEN 3115
3100 IF D9<X2 THEN 3120
3110 LET D9=X2
3112 GO TO 3120
3115 LET D9=-X2
3120 LET S=S2+T3*D9
3130 RETURN
3140 GOSUB 2000
9999 END

```

Some Results for the S/n/16-n/8 Cases
for S=256
and for n=8,10 and 12

256 / 8 / 8 / 8							
1	52.1	46.2	44	41.4	38.5	36.4	
2	52.1	50.7	47.5	48.9	49.3	47.1	
3	52.1	53.3	52.2	53.5	50.8	50.9	
4	52.1	51.3	52.3	53	52.7	52.6	
256 / 10 / 6 / 8							
1	63.2	40.1	28.7	23.1	19.4	17.4	
2	63.2	49.3	36.2	32	29.7	28.1	
3	63.2	51.5	39.3	35.6	33.5	32.1	
4	63.2	66.2	64.7	63.4	63.8	63.5	
256 / 12 / 4 / 8							
1	39	27.1	22.5	18.7	16	14.3	
2	39	31.9	29	27.3	26	25.1	
3	39	33.9	31.8	30.6	29.7	29	
4	39	38	37.9	37.9	38	38	

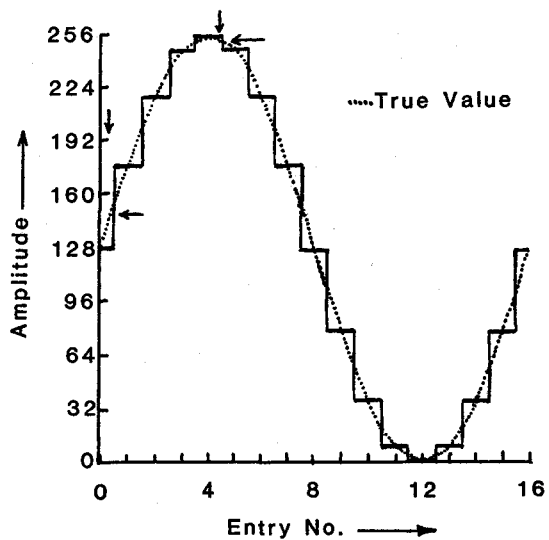


Figure 1) 8-Bit Sine Wave Table of 16 Entries

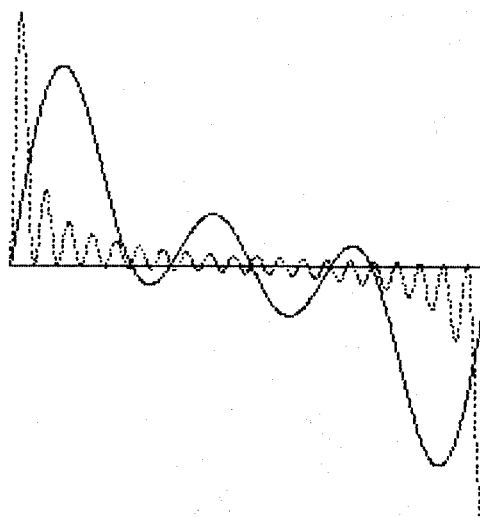


Figure 2a) Type-1 Waveforms, $m=3,20$

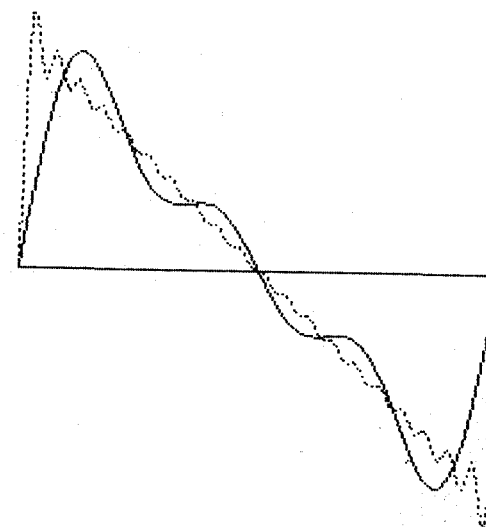


Figure 2b) Type-2 Waveforms, $m=3,20$

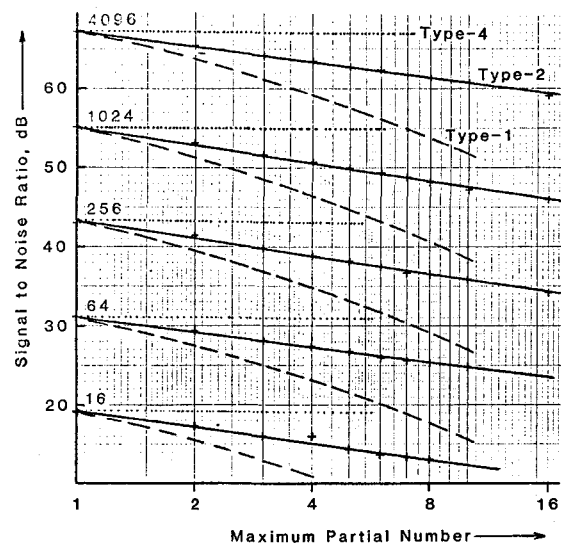


Figure 3) Effect of Table Size on Truncation Noise

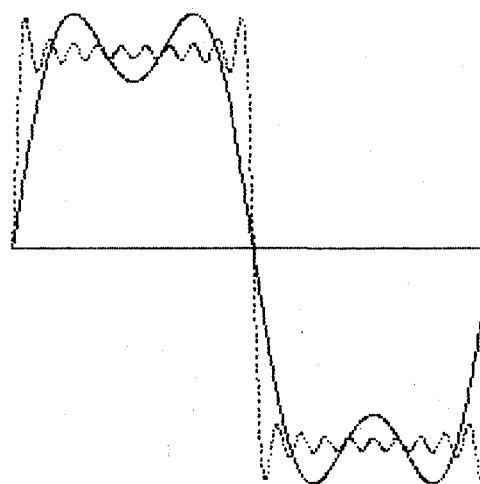


Figure 2c) Type-3 Waveforms, $m=3,19$

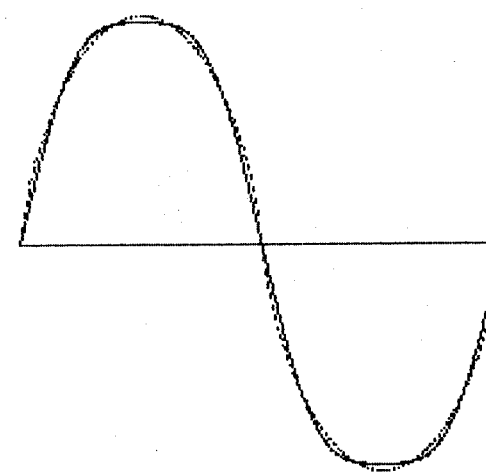


Figure 2d) Type-4 Waveforms, $m=3,19$

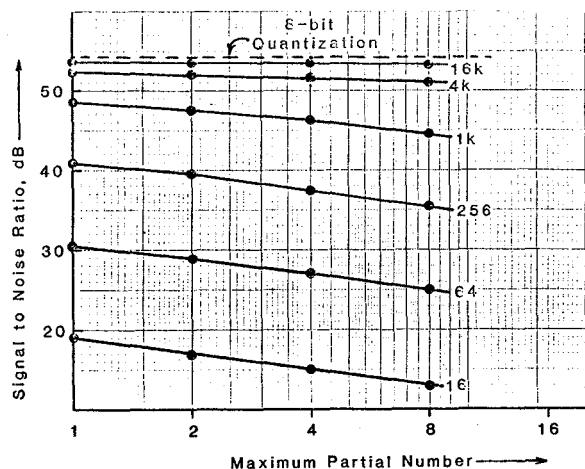


Figure 4) Combined Effects of 8-Bit Quantization and Truncation

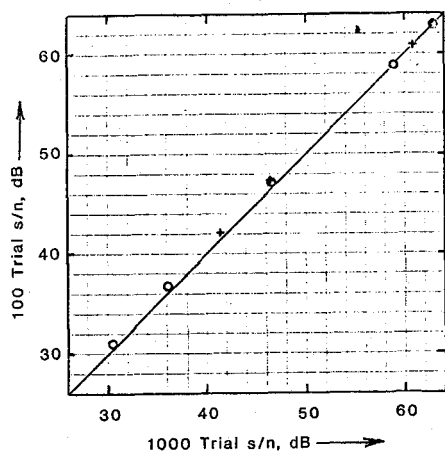


Figure 6) 100 Trials vs 1000 Trials for RMS s/n Calculations

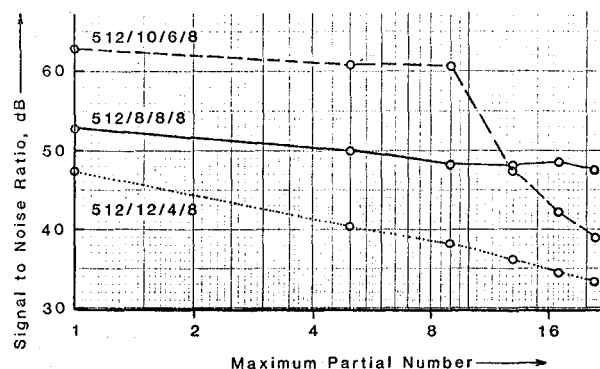


Figure 8) Performance of the 512/n/16-n/8 Cases of Multiple Quantization

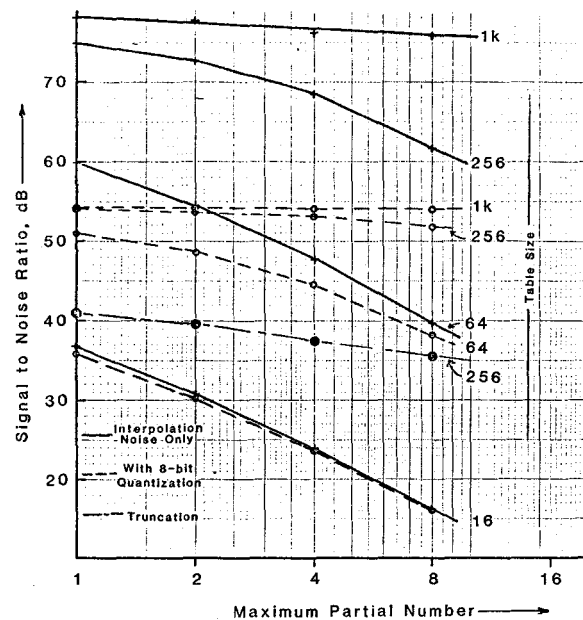


Figure 5) Residual Noise from Linear Interpolation Combined with 8 and 12-Bit Quantization

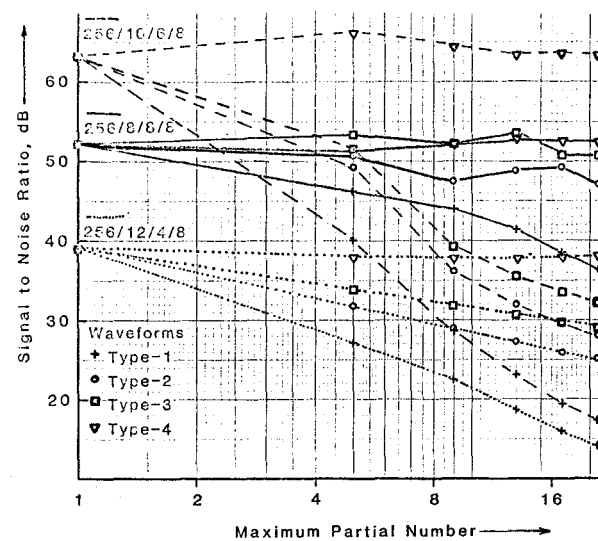


Figure 7) Performance of the 256/n/16-n/8 Cases of Multiple Quantization

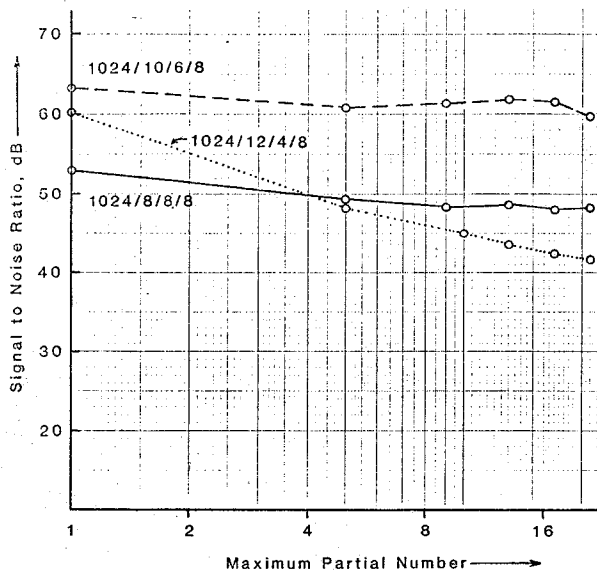


Figure 9) Performance of the 1024/n/16-n/8 Cases of Multiple Quantization

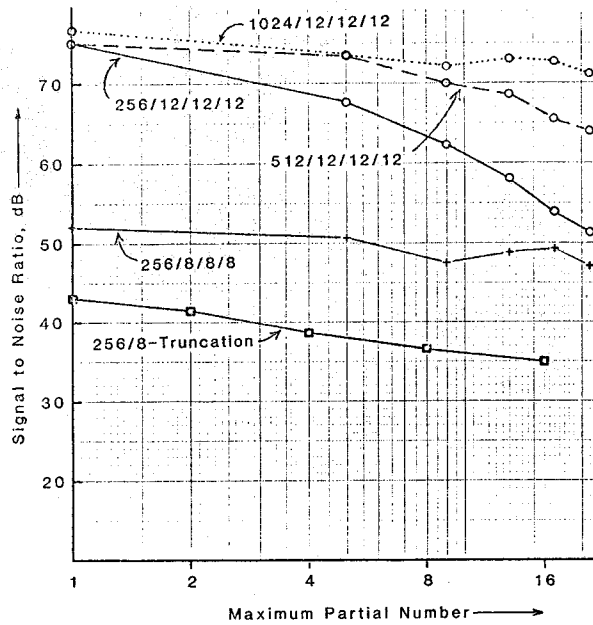


Figure 10) Performance of the S/12/12/12 Cases of Multiple Quantization

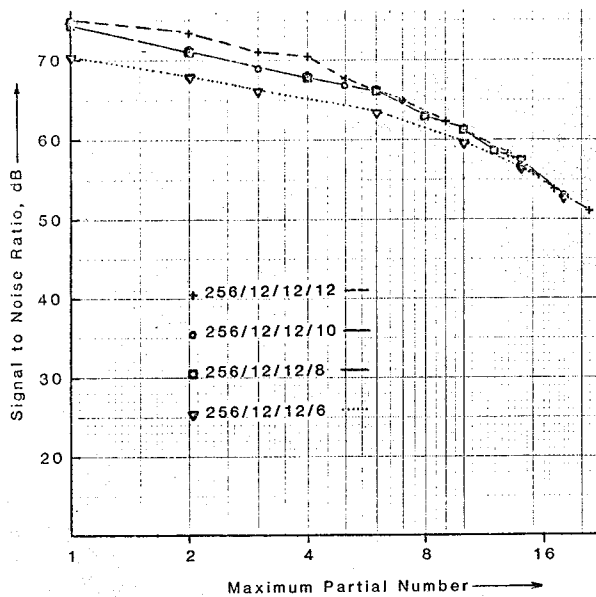


Figure 11) Performance of the 256/12/12/8 Case of Multiple Quantization

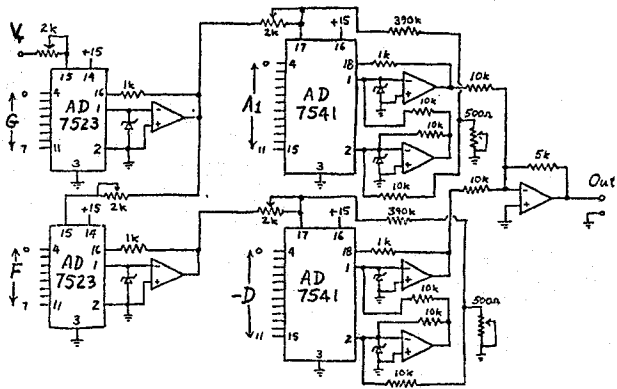


Figure 12) Interpolation DAC Circuit Design Schematic

SONIC SET THEORY: A TONAL MUSIC THEORY FOR COMPUTERS

Tools for Algorithmic Composition

With an Algorithm for Chord Selection for Dessert

by Laurie Spiedel
175 Duane St., NYC, NY 10013
August, 1982

For millenia, humans have wanted to be able to listen to the music of numbers. Pythagoras, with his "music of the spheres", and Kepler, in his "harmony of the planets" idea are the two intellectual adventurers who have most inspired us to this ideal. With computers, we can at last listen to number as music, create music as number made audible. But despite these long sought and newly arrived pleasures, we also want to use the tools of math and logic to create music meaningful in the same ways that traditional music has been. We wish to do so in order to expand the realm of music in an evolutionary manner, to make the pleasure of it's creation easier and accessible to more people, and to study and better comprehend its effect on us, to heighten our self-awareness and self-understanding.

Algorithmic music

We can separate algorithmic music into four main tendencies. The first of these is the sonic embodiment of extra-musically derived relationships (mathematical, visual, or physical). Kepler's idea, and translations into sound of mathematical series (prime numbers modulo 12, etc.) furnish good examples of this first-mentioned trend.

The second tendency consists of attempts to extend the boundaries of what we know as music, to extend the areas of the imagination and its products which we are able to experience in common. This can be exemplified by Xenakis's stochastic techniques.

The third is the description and composition by rule of structures derived from the study of traditional music, such as Hiller's ILIAC suite. Such work would include simulation programs which might do such things as create melodies against given chords, or conversely, or generate music in a particular style from nothing but rules. One goal here is the development of easier, more economical ways of describing large numbers of musical events.

The fourth trend in algorithmic composition would be the musical analog of artificial intelligence research. Musical "AI" might try to generate all the musical possibilities for a given situation and then "filter" out (by logical constraints) everything that won't "work" musically. Systems of such rules and

constraints would evolve, gradually, as embodiments of our self-understanding, and might tie in closely with research in cognition, perception, information theory, communications or game theory. I distinguish these somewhat overlapped third and fourth tendencies chiefly on the basis of their goals, the former predominantly desiring to create repertoire and to understand music as we have known it in the past, and the latter predominantly wishing to add to our self-knowledge and capabilities.

A Computer Music Theory

Music is subtle and mysterious in its ways of working on us, but - up to a point as yet to be approached - it's workings can be comprehended and described. Different periods have had different music theories and conceptual tools, intended to be useful to composers in creating works, and to players and listeners in interpreting them. Species counterpoint, solfeggio, Rameau's theories on harmony and figured bass are among the most significant of such tools for European music. As computers lead us into new realms of creative work in music, providing us with new compositional, elaborative, analytical, and descriptive tools for music, we may be able to benefit from a new model or theory of music which relates better to the concepts we use in working with these new tools than do the music theories of the past.

It is the purpose of this paper to begin to furnish such conceptual tools, to at least scratch the surface of a variant of music theory (tonal to start) which will fit more easily into the conceptual vocabulary of programming than those already in use. This beginning theoretical revision contains little that is new. It consists, instead, of a way of organizing what we already understand so that it will become, hopefully, a bit easier to describe musical relationships in computer terms. The two concepts which it introduces to music are set theory and the idea levels of indirection. (The idea of hierarchy already exists in traditional music theory, particularly in the domain of time.)

Musical Frequency Patterning is Unsterwise

The way that frequency is meaningful in music is not well represented by a linear scheme of organization. You can compose soundpieces within the equal tempered scale by treating it like a uniform, ordered, and unweighted set of integers 1 to 12, but the structures of such works will not fit easily to the way the ear makes sense of music. Music generally moves in units which consist of unequal numbers of scale steps. It moves among recognizable patterns by common tones, common harmonic content of tones, and by intervals which occur within established patterns (such as chords). In addition, some scale members (for example, tonic and dominant) occur with much higher frequency than do others.

In our culture, dominant patterns consisting of unequal intervals include tonal scales, triadic and other chords, and the patterns of movement of the roots of chords (which tend to be in fifths or thirds). An arbitrary mathematically derived number sequence in base 12 may be interesting to the mind, but it will be unlikely to be a highly controllable way to rouse the emotions through the manipulation of expectation in time, or by harmony or melody. Equal-step quantization within the octave is not how our music is set up.

Most music meaningful to us is tonal in its patterning, not chromatic (using all 12 tones within the equal tempered octave). The closest harmonically related chords to C major are not C-sharp or B major, but A minor and G major. Harmonic content (internal frequency ratios), not the proximity of frequencies, is what counts.

The question posed is this: How can we better deal with music's complex non-uniform non-scalar patterning with computer/mathematical logic? By "better" I mean both the abandonment of methods which are ineffectual in a computer language context, and the adoption or optimization of methods and concepts which permit us to make full musical use of the computer's unique capabilities.

Sets and Subsets in Musical Frequency

Tonal pitch collections, such as chords, scales, or modes, can be viewed as templates which can be laid over the equal tempered scale. Such templates as we are about to look at for the tonal/modal collection could be tried (found or invented) for any scale (microtonal, etc.), or any group or cluster of pitches. However, for purposes of simplicity and clarity, we'll restrict the application of this idea in this article to the conventional uses of the conventional equal tempered scale.

A template can also be regarded as a way of referencing a subset, of including and excluding set members by virtue of their relationship with each other rather than on the basis of their specific characteristics. Tonal music can therefore be

considered in terms of sets and subsets, maps and submaps, or templates on templates (sequential layers of pitch filters).

On the first level, that of the most inclusive or general set, we posit continuous frequency. Discrete pitch collections can be viewed as subsets of continuous frequency. As we select smaller subsets, we come to the equal tempered scale. The next most meaningful and general subsets include scales and modes (major, harmonic or melodic minor, dorian, phrygian, lydian, mixolydian, aeolian, pentatonic, and other more exotic modes). If we're dealing with modal music, we might go to the tetrachord as the next most generally meaningful subset. For tonal music, we go to the triad.

Triads are easy templates which can be dealt with even without a more general tonal template. There are 4 kinds, which are easily stored as arrays of offsets from the tonic of a chord. Once you know the root (or tonic) of the triad, you add these offsets, which are counted in terms of our standard minimal unit, the semitone. A major triad's 2 upper notes are at offsets of 4 and 7 semitones from the tonic, minor at 3 and 7, diminished at 3 and 6, and augmented at 4 and 8. Any of these addends plus a multiple of 12 (to offset the pattern to other octaves) will give you a note that is in the triad.

Here's a simple example of how this idea can be used:

```
REM Arpeggiate chords
REM minor then major on each
REM step of a rising bassline.
REM Works musically by keeping
REM common tones across bass movement.
```

```
Dimension Etscale(12 * number of octaves)
REM fill with Equal Tempered scale values
REM for sending to particular hardware
```

```
Dimension Major(4), Minor(4), Bass(13)
```

```
REM create rising bassline
REM (or you could compose one)
For I = 0 to 12
  Bass(I) = I
Next I
```

```
Minor(0)=0
Minor(1)=3
Minor(2)=7
Minor(3)=12
```

```
Major(0)=0
Major(1)=4
Major(2)=7
Major(3)=12
```

```
REM for each scale degree
For B = 0 to 12
```

```
REM arpeggiate minor chord
For I = 0 to 3
  Note = Etscale(Bass(B) + Minor(I))
  REM play note here
Next I
```

```

REM arpeggiate major chord
For I = 0 to 3
Note = Etscale(Bass(B) + Major(I))
REM play note here
Next I

```

```

REM move to next bass note
Next B

```

End

Triads aren't sufficient for tonality, though. Chords move by two main methods. The first type of chordal movement is to another chord whose root or "tonic" is a fifth away from the current one. To create this kind of root movement, we might substitute some kinds of movement through the following array, consisting of a cycle of fifths for our rising baseline in the above example:

```

Dim Cycle5(12)
REM fill it with
REM 0 7 2 9 4 11 6 1 8 3 10 5
REM C G D A E B F# C# G# D# A# F

```

The second most useful type of chord movement is by common tones, such as to a parallel or relative minor. In both these cases, 2 notes of a triad remain common while the third one changes. C major is "parallel" to C minor, the C and G being common to both while only the E and E-flat differ. C major and A minor are "relative" major and minor chords, the C and E being common to both. Parallel chords are easily interchanged by defining them as dyads a perfect fifth apart, and then algorithmically filling in the major or minor third of the chord.

Relative major and minor chords can easily be dealt with by using an array which consists of the union of the major and minor triads (A,C,E,G) and simply using a starting index of 1 for major and 0 for minor when reading 3 consecutive offsets from this array. (The array would not have the note names in it, but would contain offsets from the tonic in semitones, which would be usable in any key and for any chord in the key. Use them by adding them to your tonic's offset into the e.t. scale array.) Another way of dealing with relative major and minor chords is by using dyads, sets consisting of 2 notes a third apart, and taking the union of 2 intersecting dyad sets as a triad, but because there are 2 types of "thirds" this is not practical until we have the following tonal set to index into.

Tonality as a Great Simplifier

The above already grows more complex than is desirable. One really wants the simplest, most general, and most internally consistent scheme of organization possible, so as to think as little as possible about "bookkeeping" and as much as possible about music. Therefore, to most simply do tonal music, we'll start with the tonal pitch collection itself as our basic working set, rather than the 12-tone scale.

The tonal set can be viewed as a subset of, or template for, the 12-tone equal tempered pitch collection. It's internal intervals (offsets between adjacent members of the equal tempered superset) are 2,2,1,2,2,2,(1). The interval of each tonal pitch as an offset from the tonic of the key are 0,2,4,5,7,9,11,(12). These are stored in an array which will be used to index into the 12-tone superset array.

This reduces our basic working set of pitches to 7. (The word "octave" suddenly has meaning again.) Pitch manipulation numerically by octaves is done modulo 8, and it becomes possible to deal with pitches as an ordered set of continuous integers without having to abandon a widely accessible musical vocabulary.

To sum up (and so a bit further), here is a list of frequency sets, each being a subset of the set immediately above it.:

1. Continuous frequency: theoretical, actual, or analog:

0 (-----)

2. Audible frequency, discrete but perceived as continuous as it approaches or matches the resolution of our ears' discrete frequency sensing mechanisms:

> 30 Hz (-----) < 20 KHz

3. Quantized equal tempered scales: (Repeat all patterns from here on down over additional octaves by adding or subtracting multiples of 12)

Spacings:

(| | | | | | | | | |)

Conventional note names:

(c c# d d# e f f# g g# a a# b)

Interval from last note in semitones:

(1 1 1 1 1 1 1 1 1 1 1)

Interval from tonic in semitones:

(0 1 2 3 4 5 6 7 8 9 10 11)

4. Tonal subset of the equal tempered set:

As pattern or template for above:

(! - ! - ! ! - ! - ! - !)

Conventional note names:

(c - d - e f - s - a - b)

Interval from last note in semitones:

(1 2 2 1 2 2 2)

Offset from tonic in semitones:

(0 2 4 5 7 9 11)

Condensed version of above as it would appear in an array of scale degree intervals:

Note names (c d e f s a b)
Interval from last member (0 2 2 1 2 2 1)
Offset from tonic (0 2 4 5 7 9 11)
Index within scale (0 1 2 3 4 5 6)

The common Gregorian modes can be generated from the above subsets of the equal tempered scale by rotating the starting index within the tonal set modulo 12, or by creating a longer array which repeats the above offset cycle adding successive multiples of 12 for successive octaves:

(c-d-ef-s-a-bc-d-ef-s-a-b)
Major (! - ! - ! - ! - !)
Dorian (! - ! - ! - ! - !)
Phrygian (! - ! - ! - ! - !)
Lydian (! - ! - ! - ! - !)
Mixolydian (! - ! - ! - ! - !)
Aeolian (! - ! - ! - ! - !)
Locrian (! - ! - ! - ! - !)

Putting this in other terms usable with arrays:

Mode	Interval	Offset
Major	2 2 1 2 2 2 1	or 0 2 4 5 7 9 11 12
Dorian	2 1 2 2 2 1 2	0 2 3 5 7 9 10 12
Phrygian	1 2 2 2 1 2 2	0 1 3 5 7 8 10 12
Lydian	2 2 2 1 2 2 1	0 2 4 6 7 9 11 12
Mixolydian	2 2 1 2 2 1 2	0 2 4 5 7 9 10 12
Aeolian	2 1 2 2 1 2 2	0 2 3 5 7 8 10 12
Locrian	1 2 2 1 2 2 2	0 1 3 5 6 8 10 12

5. Triadic subsets of the tonal set (From here down use modulo 8 arithmetic.):

As pattern or template for the major scale:

Tonal set (c d e f s a b c)
Names (c - e - s - - c)
Intervals (0 - 2 - 2 - - 3)
Offsets (0 - 2 - 4 - - 7)

Chord quality is automatically tonally correct when chords are described as rotations within this ordered subset or array:

c d e f s a b
(! - ! - ! - -) C (I) major
(- ! - ! - ! -) D (ii) minor
(- - ! - ! - !) E (iii) minor
(! - - ! - ! -) F (IV) major
(- ! - - ! - !) G (V) major
(! - ! - - ! -) A (vi) minor
(- ! - ! - - !) B (vii) diminished

or to make the pattern more visible:

cdefsbcddefsbcd
012345601234560

!.!.!.!.! I major
!.!.!.!.! ii minor
!.!.!.!.! iii minor
!.!.!.!.! IV major
!.!.!.!.! V major
!.!.!.!.! vi minor
!.!.!.!.! vii diminished

At this point, when we make triads, we no longer have to keep track what kind of chord qualities they have (major, minor, diminished), or of the differing numbers of semitones in the internal intervals of these different tonal chords. All triads consist of offsets of 0, 2, and 4 added to any number within the tonal pitch set (0 to 7); that number indicating the root of the chord. The triadic pattern moved as a template through the ordered tonal set automatically results in a triad of the proper chord quality when the tonal array in its turn is used as a template for indexing into the equal tempered scale array.

This conceptual organization makes it much easier to compute musical material by algorithm without abandoning the harmonic vocabulary in which non-computer music has most effectively reached us.

The above in less abstract and more useful form:

More computerishly put, this Set Theory of Musical Pitch includes these types of sets (Continua, Contisua, Collections, Chords (including Clusters), and Cycles):

#1. Continuum - Frequency: Not applicable to computers.

#2. Contisuum - Audible frequency. Limited by hardware for sound generation, by wordsize, and by the physiology of hearing. (I propose the term "contisuum" for sets of discrete contiguous elements):

#3. Contisuum - Quantized scale: An array containing values to output to oscillator hardware. These can be equally quantized (tempered, microtonal, or macrotonal scales), or much more bizarre.

#4. Collection - Tonal set: An array of indices into the equal tempered scale in #3 above as follows: 0,2,4,5,7,9,11...

#5. Chord - Triadic subset: Array of indices into #4 above (major = 0,2,4,7,...).

Other musically useful arrays which can be expressed as subsets of the tonal collection:

#6. Collection - Pentatonic (0,1,2,4,5) (= C,D,E,G,A). This collection is a subset of the major scale which excludes the two single-semitone intervals and the tritone (an interval of 6 semi-tones, or 3 tones), leaving only intervals of a whole-tone or larger.

#7. Chord - Fourth chords (0,1,2,4,5) (= C,D,G,A), which are wrap-arounds of the cycle of fifths. (I use the term "wrap-around" in from computer graphics to indicate a folding in modulo some finite range, in this case the octave.)

#8. Cycle - of fifths. The subset of the full cycle of fifths which falls within a single key signature, without adding sharps or flats, is (as offsets wrapped around into one octave: 3,0,5,2,6) (= F,C,G,D,A,E,B), is also useful. (These can be added to the index into the equal tempered scale array, for modulation to related keys in tonal pieces.)

#9. Cycles - Equal divisions of the octave:

- a. By 1/12 = equal tempered scale.
- b. By 1/6 = whole tone scale.
- c. By 1/4 = diminished seventh chord.
- d. By 1/3 = augmented chord.
- e. By 1/2 = the tritone (and octave).

#10. Cycle - Alternate major and minor thirds:

c e s b d f# a c# e s# b d# f# ...

#11. Cycle - Dovetailed minor seventh chords (courtesy of Ron Everett in Toronto). These consist of a repeating cycle of a major third followed by two minor thirds:

c e s b-flat d f a-flat c e-flat s-flat ...

#12. Composed - Those I'll leave to all of our imaginations.

Levels of Indirection in Musical Progression

The above is all good useful stuff for algorithmic composition, but conceptual mechanisms for referencing and assembling scales, triadic, and other patterns are still insufficient for the composition of tonal progressions. For the creation of chord progressions, I have found it useful to conceive of them as series of levels of indirection (removal) from a destination chord (generally the tonic). These can also be conceived of as nested loops, concentric circles, or in a tree configuration.

As a tree-like structure, all branches connect at nodes where selection is done, such that we move in, layer by layer toward the single central tree trunk. The difference between the standard binary tree model and what I describe here is twofold.

First, the purpose of this tree is not finding something which has been stored, not for searching. Its purpose is to provide a means of structuring paths of movement. Music is process, progression. (Arrival in music may threaten to stop the music. Note how Bach elides cadences, so that a chord which is heard as a strong arrival, or ending, is set up to be simultaneously heard as the beginning of a new phrase.)

Second, the direction of movement is opposite to that of a tree search. Instead of starting at a central known place and selecting where to branch to next, we may find ourselves just anywhere up in the branches, in the structure of possibilities of a given moment, and we know we always want to move one level down toward the trunk. We don't care what branch we're currently in, but must know how many levels out we are from the trunk so we know by what means to select where to go in the next level down. We wish to do this in some aesthetically effective manner, somehow not too obvious or expected, not consistent or predictable, but not too shocking, awkward, or unpredictable either. There must be underlying feelings of a sense of direction and that where we've arrived, though unexpected, makes sense.

As an example, here is an algorithm for a simple chord sequence generator based on the idea of levels of indirection from a chord of harmonic resolution, using frequency subsets (expressed as arrays of indices into other arrays, as per the above):

Tree of Levels of Indirection from Tonic chord:

Arrays used include the equal tempered scale ("etscale"), our 7 tonal intervallic offsets from the tonic ("tonality"), and the 3 offsets to the tonic which define a chord ("triad").

4 - Leaves:

iii chord (can go to other places besides the 2 chords on the next level down, but let's start simply).

```
REM Play arpeggiated triad
tonic = 3
for i = 0 to 3
note = etscale(tonality(tonic + triad(i)))
next i

REM select next chord by weighted probability
REM (or use other means)
which = random mod 25
if which < 9 then tonic = 1
else tonic = 6
```

3 - Twigs:

I chord or vi chord (can go to either chord on next level down).

(Repeat 2 steps above, with current values.)

2 - Branches:

IV or ii chords (can go to either chord on next level down).

(Repeat 2 steps above, with current values.)

1 - Boughs:

V or vii chords (can go to either chord on next level down).

(Repeat 2 steps above, with current values.)

0 - Trunk:

I (tonic) or vi (tonic of relative minor key)

```
REM Keep going even though we got here.
level = random mod 5
goto level
```

In case it isn't utterly and completely obvious, the above is simply a statement of logic, and will have to be coded for whatever machine, language, and oscillator hardware you may use. (I have a more extensive version of this algorithm running on my Apple II and Mountain Hardware oscillator boards, as part of my PASCAL composing system, "AMO" (A Musical Offerings). Rather than including a printout, though, I thought it would be clearer to state this idea in general form, so that it may be better understood, more widely tried out, played with, and adapted to different music systems.)

Though the 2 possible chords on each level of this tree have roots a third apart, I chose not to describe them as single arrays containing the union of the 2 triads, to select them by choosing a starting index of either 0 or 1 into each union array (vi-I, ii-IV, V-vii) because that would have limited the variability and generality of the algorithm.

The aesthetics of this algorithm, like many others which are possible, its personalizability and the nature of its expression, lie in the choice of which chords (clusters, modes, etc.) may be moved between, and in what order, and in the design of the method of decision making (in this case the simple weighting of the probabilities). I suggest starting, if you do decide to try running it, with the chords I have selected, and with 50-50 probabilities on all levels, before trying to vary it. (To make sure this progression algorithm really worked musically, I wanted to first test it within a harmonic style derived from J.S. Bach, which accounts for the chords used and their positions.)

As it stands above, variables available to play with include probabilities, the number of levels used, and the chords or tone clusters, modes or scales which are entered in the arrays. Among the variables not dealt with in the above are tempo, timbre, stereo location, density (number of simultaneous voices), range in octaves, and envelope parameters. Elaborative processes which could be added include the introduction of other shapes of arpeggios, the generation of melodic lines from the chords, instead of arpeggios, by use of leaps and passing tones, or the substitution of patterns (trills, tremolos, and other ornaments, or of motives or melodies) for notes.

Generalizing to Other Musical Dimensions

The structure applied above to chord progression can be used as a model for software which can generate routings through other areas of our frequency map as well. This can be done within any single level (for example, among scales instead of among chords), or to structure movement from one level of our set map (one area of our sonic subset theory) to another, that is to be used to redefine the actual working premises of our musical reality during a piece.

The same principles of organization can also be applied to other dimensions of music. Rhythm furnishes a good example, as it can easily be viewed as hierarchical (as a branching tree). The architecture of time can be hierarchically described, from 32 to 16 measure units, to 8 bar phrases, and on down to beat to beat rhythms. Progressions of harmony and rhythm merge in the concept of "harmonic rhythm" (the number of beats between regularly spaced chord changes), which can also be manipulated algorithmically. Rhythmic meter can be viewed as a branching tree, in which each beat-level can be subdivided in a variety of selectable ways. (A quarter note can be divided into 2 eighth notes, a 3 eighth note triplet, 4 sixteenth notes, etc.).

What, Why, and Whither Algorithmic Music?

Algorithms may be viewed as general compositional processes, as are canon and fugue, or as unique musical compositions, or as falling somewhere between those 2 extremes. Depending on their degree of interactivity (whether variables are read in from knobs, switches, keyboards, or other devices during a program run, or whether they are generated or stored within the software), such algorithms may be placed anywhere along the axis between "intelligent instruments" and "automated composition."

Algorithms may become a dominant musical form in the future for a number of reasons. They can be highly user interactive via realtime access to variables (anywhere from audio same to virtuoso's instrument). They are natural for the powerful musical instruments of the future which will permit maximal musical expression with minimal input, by the use of an increasingly small, select, and aesthetically powerful group of variables. They will be attractive to use by virtue of their overcoming the mechanical limitations of traditional instruments whereby the ratio of the number of notes played by the person to the number of notes played by the instrument is rarely better than one-to-one. Musical pattern generation, manipulation, and editing programs will greatly facilitate composition, as well as having wide application in music education.

Algorithmic music may turn out to be most desirable (so much so that economics may ensure its proliferation as a new musical common practice) because algorithms involving relatively small numbers of relatively powerful variables are extremely economical in terms of the amount of storage space necessary to describe a piece of music. In other words, the amount of stored (or telecommunicated) data necessary "per thousand" of musical notes played is small. Because a small number of variables can control the nature of a musical fabric, there is also a great potential for musical response to other technologically interfactable phenomena, such as temporal visual compositions.

The superset of high level variables can include global musical dimensions such as ranges, types of interpolation or transition (most easily done by selecting one of a group of a precomposed tables), rates (of acceleration, crescendo, or harmonic change). It may also include the selection or conditional use of such musical transformations as I described in my article of last year for this same Symposium and publication. The algorithmic selection and variation of musical decision making processes themselves is an obvious and interesting recursion. Other important parameters, processes, and principles will be derived from theories of perception, cognition, games, information, and communication, and may include density, level of "contrast" among sonic components, percentage of redundancy, degree of continuity, type of organizational structure, and others.

Algorithmic composition is a concept applicable to other arts besides music, one example being the "same of life" in computer graphics. Algorithmic interfaces of audible and visual musics will become increasingly meaningful as variables become more powerful and more cognitively oriented (above the level of any single sensory modality).

The ultimate algorithmic artworks may hope to describe, by rules based on understanding, all the characteristics of stimuli which are most meaningful to our consciousness, externally embodying the mirror reflection of the structure of that consciousness and its mechanisms of finding such meanings. Algorithms may eventually simulate the transformations which sound and image undergo in our imaginations, which have been so difficult to capture by conventional methods of musical and artistic expression.

The basic tools above are only a most simplistic beginning, designed to facilitate broader participation in all that waits to be explored.

- Laurie Spiegel
NYC, Aug. 1982

Alternative description to 1st algorithm above:



MERGING EVENT LISTS IN REAL-TIME

J. William Mauchly
Personal Computer Arts Group
Box 1954 Philadelphia, Pa. 19105

This paper describes the data structures and the processing algorithms for a program to perform real-time control of a music synthesis system. The program can combine multiple lists of notes or other timed events while servicing a performer's input device.

The computer has opened up new modes of interaction between musician and musical instrument. Besides the tonal palette that digital synthesis offers, time itself is a dimension we can control in fresh ways. Many computer-based music systems now offer the ability to record and playback musical passages that are performed on a keyboard. A musical score can be created from such a performance, and edited or manipulated in various ways.

When approaching the task of creating software that will perform functions like these, the choice of data structures is important. Two useful structures will be discussed here: linked lists and ring buffers. Linked lists can represent musical units like envelopes, melodies, and chords. However the job of processing lists can be time-consuming when several lists must be simultaneously traversed in real-time. The ring buffer is a data structure which simplifies output processing and eliminates sorting parallel event lists.

A TYPICAL MICROCOMPUTER MUSIC SYSTEM

A microcomputer can control a variety of music-generating devices. Many of the software concepts are the same whether the sound comes from a player piano or a digital synthesizer. The synthesizer could be an analog voltage-controlled type which is linked to the computer with some digital-to-analog converters. It could be a digital waveform generator like the Mountain Computer MusicSystem or the Casheab S-100 Synthesizer.

Let's assume for now that the music-generating device has some sort of hardware ramp generator or envelope generator. We need to tell it when notes start and stop, and of course the pitches, envelopes, and waveforms to use. (On some systems it is necessary for the computer to supply every amplitude change for the oscillators).

A typical software system will need to simultaneously play back a recorded performance and allow a performer to play along on a music keyboard; often some graphics are also live on the screen. The job of monitoring the keyboard, reading a score, and controlling some sort of synthesizer is handled by a microcomputer running a very quick program.

Usually many other features are desired which can be treated as separate functions. A recorded performance can be converted to a graphic score and edited. The software can also provide facilities for the control of the synthesizer "patches", which are specific to each type of sound-generating hardware. We will only be discussing the real-time performance program which must do "note-processing" on-the-fly.

Lets look at the tasks which must be accomplished simultaneously:

- 1) Determine what new keys have been pressed or released on the keyboard.
- 2) Find all notes in the active scores which should sound.
- 3) Translate keys or notes into synthesis parameter information: pitches, amplitudes, waveforms, and envelopes.
- 4) Send to the synthesizer the selected parameters at the exact time they are required.
- 5) In addition we may ask the computer to monitor the alpha-numeric keyboard and analog input devices, and to update a display screen.

To understand this complex task we must first consider the types of data we will be dealing with.

REPRESENTING MUSICAL STRUCTURES

NOTES

The fallacy that all music is composed of elemental units called notes has pervaded computer music. Early composers had to create a punch card for every note in a piece. It was the smallest and largest entity for building a score. Other ways of representing music have grown out of a desire for the structure of the music to be reflected in the score. For example, bar lines mark off measures in conventional notation. They aid in conceptualizing the music. A long list of notes becomes a phrase; a series of phrases can be repeated inside a passage, etc. Meanwhile, a single note has been broken into pieces: attack, decay, and so on. All these descriptions of musical events are efforts to express analytically what our ear knows intuitively.

MUSICAL SENSE

Good music carries emotional content or meaning, but also must make "sense" in an abstract way. The sense is in terms of organization or structure. As we listen, we look for patterns. The patterns which we recognize help us to predict what will happen next. It is the balance of the expected and the unexpected which keeps us interested or involved in listening. We are matching patterns, looking for repetitions in pitch, rhythm, and all kinds of shapes described by the music. The patterns may be very long or very very short. The complexity of this mental feat is staggering. Fortunately, a listener needn't understand the process to perform it; we do it quite naturally.

What we would like, and what we will never have, is a way of representing music which reveals its structure completely. That is a goal which we can individually strive for; there is no one way to solve the problem.

There are some basic data structures which can serve as a foundation for a very flexible music system. They can support higher level musical abstractions for those interested in exploring them.

INPUT EVENTS

A music keyboard is the typical input device. We would like the keyboard to be preprocessed, either in software or hardware, so that we only receive data when some key goes up or down. (That is, we don't have to interrogate every key to find out if it moved.) We will receive asynchronous key movements and perhaps some velocity value. So we can expect packets of data: the key's identification number, whether it was just pressed or just released, and perhaps a scalar value for its velocity. No time information is present; when a packet appears, it has just occurred.

We can generalize the motion of a key as an "event". An event is a discrete motion occurring at some discrete time. When a key goes down, that is an event; when that key is lifted again, that is a separate event. If there are analog inputs from footpedals or joysticks, they may be expressed as events also. Every event has three parts: the thing that moved, its new position, and the time that it occurred. For a keypress on a musical keyboard the position may be only a binary (up,down) value or it may contain some velocity or position information. As input data comes in from our performer, we will associate the present "time-of-day" to it, creating a series or sequence of events.

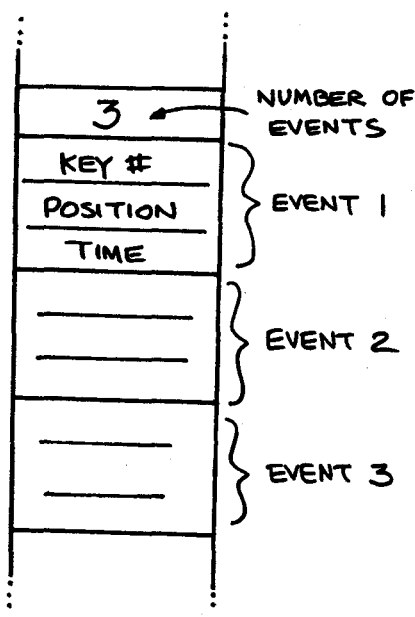
OUTPUT EVENTS

The real work that is critical to the music system is to feed the right parameters to the synthesizer at the right time.

At the lowest level we are just sending out numbers to some peripheral device. This is just like a poke from BASIC; we supply an address and a piece of data to put there. The address will be the identification number of some parameter we wish to supply (not necessarily the physical address in memory) and the data is the value we wish to change it to. The output too, then, can be seen as a series of events.

EVENT STRINGS

When a series of events are recorded, there are a number of ways to store the information. The time of an event can either be referenced to some absolute "start time" of the sequence, or it may represent the time since the preceding



AN EVENT STRING

event. Both ways have advantages. How the events are strung together is more important. For simplicity and economy of space, sequential storage of events seems appropriate.

A simple EVENT STRING has two parts. The event-count and the actual events. The events are stored sequentially in memory. When a performance is recorded, an event string can be created quite efficiently.

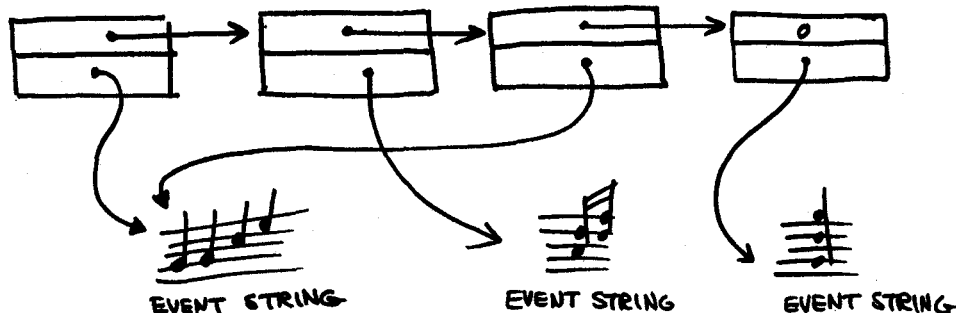
EVENT STRING

- 1) Number of events
- 2) event #1
- event #2
- event #3
- .
- .
- .

Event strings can serve as a basic building block for all kinds of musical structures, like chords, melodies, drum rhythms, etc. As we start trying to edit event strings we notice their deficiencies. To insert a single event we must rewrite all the data following the edit point. If we wish to repeat a passage we need twice as much memory space. It's the same old problem programmers have had with text processing and disk storage. The answer is just as old: linked lists.

A linked list uses pointers to connect the elements in the list. A pointer is really a nice name for a memory address. Every element in a linked list has a special variable associated with it which points to (holds the address of) the next element in the list. The elements are chained together, each pointing to the next. the last element in the list has a "nil" pointer (a zero where the pointer should be).

We would like to link together event strings into a new structure; we'll call it a sequence. (The ambitious should allow for lists of lists of lists). A sequence is a linked list, where each element in the list has two parts.



AN EVENT 'SEQUENCE' (LINKED LIST)

SEQUENCE ELEMENT

- 1) pointer to next element
- 2) pointer to an event string

A detailed discussion of linked lists and list processing can be found in most texts on data structures. They seem to be very useful for expressing musical structures of many sorts.

SCORES

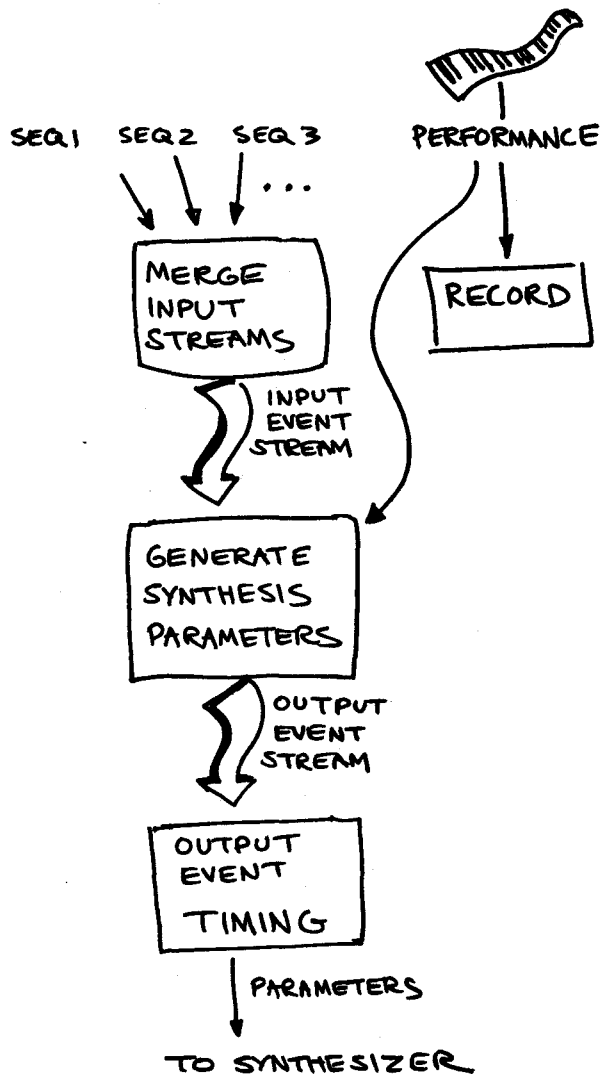
One other activity we should consider is the interpretation of scores. Perhaps the cleanest way of dealing with conventional scores is to pre-process them into a sequence of events. It is akin to compiling a program; the processing can be very sophisticated because we are not trying to do it "on-the-fly" while the program is running. That leaves the compiler all the time in the world to read graphic scores, interpret chord symbols, generate random elements and all those nice things we dream about.

But a score can be any number of things, depending on its purpose. Generally, it is a description of a musical performance in some language. But it can also be a description of a musical PROCESS. If we want the computer to take a performer's input and use it in some particular way, that is a kind of score. In computer terms, we need an interpreter, not a compiler.

REAL-TIME PROGRAMMING

Now we can see more clearly the type of job that needs to be done. Inputs are coming in real-time. We have to expand them into commands which the synthesizer will understand. We need to record the exact time of the input, and dump out from other lists other events at specific times. In addition special programs of our own devise ("interactive scores") are somehow running at the same time.

The only practical way to accomplish these concurrent tasks is to try to average the work over time. When a finger comes down on a key, there can be no perceptible delay before the sound emerges from the other end of the system. That task becomes a high priority. Anything else that has to occur at that moment must be ready to go; we should try to get as much stuff done ahead-of-time as possible.



Interrupt driven input and output provides just the "think-ahead" time we need.

First, an element common to all music systems must be defined: the "tick." The clock in a computer music system is an absolute time-base against which all events are scheduled. For our purposes we will assume that the clock "ticks" at the smallest resolvable time increment desired. For example, the Mountain Computer MusicSystem for the Apple II produces an 8 millisecond interrupt. (125 times a second). When an interrupt occurs, the computer starts executing an interrupt service routine. This program looks at the scheduled events; if one is scheduled for the present time, it is

enacted. Thus the closest that two events can occur in time is 8 milliseconds apart. The tempo of the music can then be defined in terms of ticks, or the ticks can be counted in software to produce a slower, dynamically variable time unit. If all notes are defined in terms of a 64th note, for example, then varying the number of ticks per 64th note will vary the tempo.

In the simplest case, the whole program is executed once every tick. If it has work to do, it will do it; if not, it will wait until the next tick. Unfortunately, music often contains great flurries of activity. A program must be very quick indeed if it is to keep up with the performer. We can separate the work into those things that are time-critical (input and output) and the things which we average out a little, as long as they get done. We will divide the task into three jobs, and find people to fill them:

- 1) follow any event lists and/or scores being played back
- 2) translate input events into output events
- 3) get data in and out on-time

THE EXECUTIVE

The EXECUTIVE will be the guy that decides what has to be done. He's the main program that's running all the time. He can follow up complicated scores and linked lists; and show quarterly reports on your monitor screen. He also controls all the lists and tables that usually hang around inside a

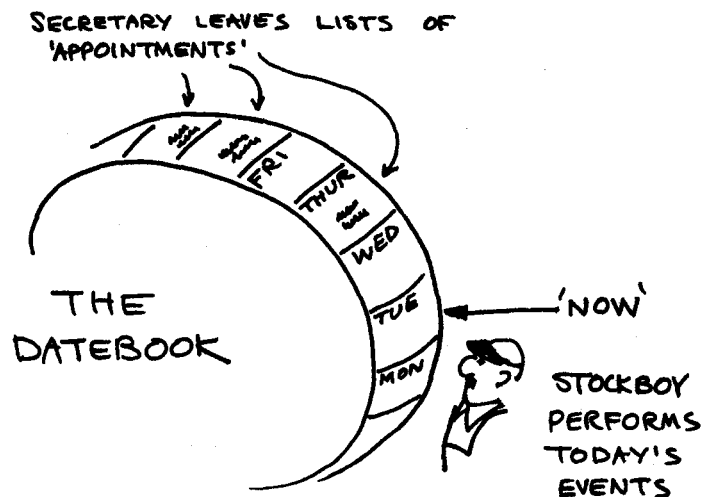
synthesizer, like what waveforms or tunings are to be used. The executive can get interrupted by other more urgent programs. As input, he takes the prerecorded sequences and scores. He looks through them, and comes up with a bunch of notes (input events) that are to occur within the next second or two. He sends them to the secretary to expedite.

THE SECRETARY

The secretary has the most work to do. She takes input events, either from the performer or from the executive, and turns them into output events. That involves taking a key number, and determining what "voice" it will take. That tells her what tables to look at to get pitch, octave, envelope, waveform, loudness, and any other relevant parameters. Depending on the architecture of the synthesizer, some of this information may have schedules associated with it. (use this number, wait 2 seconds, use that number). She has the job of writing all such OUTPUT EVENTS into an "datebook" of things to happen in the future.

THE STOCKBOY

The STOCKBOY makes sure everything gets shipped out right on schedule. He may have a lot to do so we will try to make it simple for him. When awakened, (a "tick" occurs) he checks the clock to see what time it is. Then he looks in his date book to see what he has to do. It's 9:00; it's time to send some new numbers to the synthesizer. He looks them up and sends them out. Now he can go back to sleep until the next



interruption. Usually his date book is almost empty; occassionally he has so much to do that another clock tick comes before he's done. That's why he has to be fast. There's no time to go searching through lists for things that might happen today.

THE DATEBOOK

That leave one more structure to be cleared up. There are two ways to keep track of timing. The first is the alarm clock method. Suppose that an event sequence is being played back, and the next event is to occur 122 clock ticks from now. We set up a software counter in memory, and initialize it to 122. This counter/timer must be decremented once each tick. That means extra work for the interrupt service routine (the stockboy) and gets him involved in executive affairs which are really none of his business.

The second approach is the datebook. The datebook is simply a buffer for the things which are to be output in the near future. Two people use it: the stockboy is reading it at a very regular rate; the secretary is writing in it in an erratic, non-sequential fashion. Each place in the datebook represents one tick of the clock. Instead of holding the actual data, each entry is a pointer to a linked list (remember them?) of output events.

The stockboy keeps track of what time it is, in other words, which entry in the datebook is "now". If the entry at "now" is a nil pointer, then evidently nothing needs to be done. On the other hand, he could find a pointer to an output event or a whole list of output events. All he has to do is shovel them out to the synthesizer.

The secretary's job is also simple thanks to the ring-buffer. As a pointer each entry in the datebook is a fixed size. (One word of memory, for example.) That means that any time in the future can easily be located in the datebook by its address. When she needs to schedule an event to occur in the future, whether its a new note sent by the executive or a continuation of some "voice" definition, she can insert it quite easily. She just uses the date of "now" and counts ahead so-many ticks. Without doing any searching to compare it with other scheduled events, she can insert the new event at the head of the list attached to that position in the buffer.

We can implement a datebook with an infinite storage capacity by letting it wrap around on itself. That turns it into a circular type of calendar or, to computer programmers, a ring-buffer. This is nothing more than an array which is addressed MOD some number.

The ring-buffer, too, is not a new concept. Like linked lists, it can make a program run more efficiently. Most importantly, it loosens the constraints on the music software by allowing the Secretary to "look ahead".

Obviously, this is just a rough sketch of some elements of a music system in terms of data objects and flow. I have tried to show how knowledge of data structures can affect the structure and the performance of a music software system.

As more musicians delve into the job of creating environments for music composition, they should realize that many of their problems have already been solved. One of the interesting things about music is how its structure reflects so much of the world around us.

Automatic Computer Composition of Bluegrass Tunes

Michael Keith
D46 Abbington Drive
Hightstown, NJ 08520

Introduction

This paper describes a program written in BASIC for the Apple II microcomputer that composes traditional bluegrass tunes, intended for playing by 5-string banjo and ensemble. Because of its instrumental orientation, I call the program **BANJO**, for "Bluegrass Algorithm for Note Juxtaposition & Organization." (besides, what's a program without a good acronym?). Briefly, the program combines a harmonic analysis of many existing bluegrass tunes with a fairly realistic note-generating algorithm. The output of the program is displayed in music tablature with chord markings. I will discuss some general philosophy of computer composition as well as the specific algorithm used in this program. Many pieces have been composed using the program; one example will be presented.

Background

The problem of composing music with a computer is not an easy one. Basically, as in other artificial-intelligence-type problems, the task is one of finding rules, hopefully relatively simple ones, that govern the structure of the program's output. In the case of music, both the melodic structure (the sequence of notes forming the main melody) and the harmonic structure (chord progressions) are important. Some types of music for which computer composition rules have been devised include two-part counterpoint [1], canons [2], and certain types of jazz [3]. Being a banjo player myself, bluegrass naturally suggested itself as another possibility.

The composing philosophy followed in this program relies heavily on automating chord progressions. I believe harmonic rules are

very important in producing acceptable music, on the grounds that as a musician I usually "think" in terms of chord progressions while playing (improvisation often is governed exclusively by harmonic progressions).

Automating chord progressions is simplified considerably by the following combinatorial fact of the 12-tone music scale: If we exclude chords containing adjacent semitones ("clashes"), then there are exactly 30 essentially different chords. "Essentially different" here means that every chord possible can be reduced by transposition or removing octaves to one of these 30 chords, which I call *fundamental chords*. More details on this can be found in [4]. Each of these 30 chords corresponds to a chord type (major, minor, augmented, ninth, etc).

It is easy, then, to analyze a selection of musical pieces of a certain type and make a list of the frequency of occurrence of chords. Each time we encounter a chord, we note its tonic (1 of 12 values) and its type (1 of 30 values). This is the basic idea of the analysis program - a bunch of bluegrass tunes are fed in and out comes a harmonic analysis. However, in addition to a frequency count, a much more important statistic which is calculated is the *transition probability* for each chord. The transition probability (A,B) is the probability that, given that the current chord is A, the next chord in the piece will be B.

It turns out that, in bluegrass, 99% of all chords are either major or minor. Thus, there are only two chord types to consider and 12 tonics, for a total of 24 chords. If we arrange these 24 chords conceptually along the X and Y axes of a graph, then the Z coordinate will graphically show us the

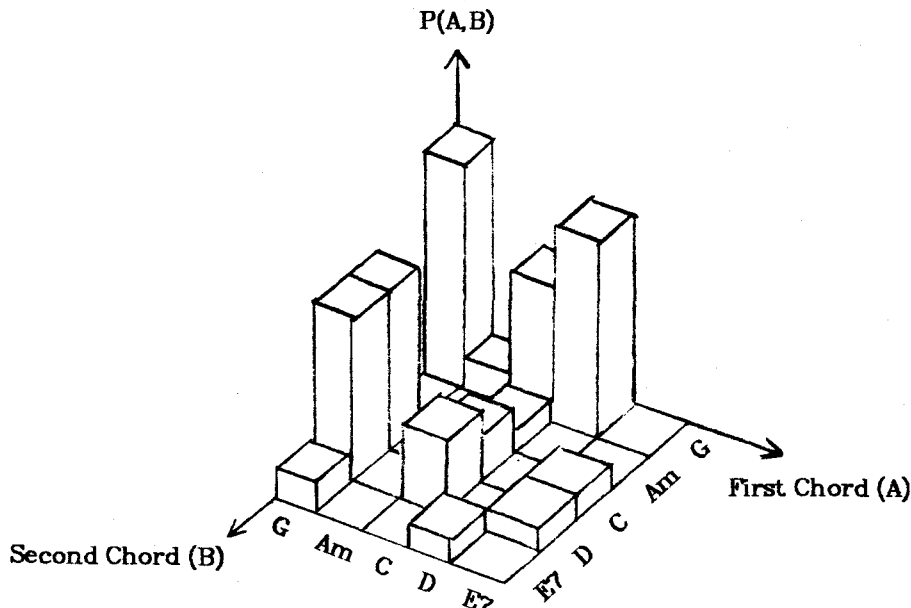


Figure 1: Sample chord transition probability graph, generated automatically by analyzing a number of bluegrass tunes. The analysis is stored on disk for later use by the composition program.

transition probability distribution. Figure 1 is such a graph for one particular analysis.

The first part of the composition algorithm, then, is to compute the chord progressions. This is done by simply using the analysis information in reverse (with a random number generator used to choose between the transition alternatives at each step). For simplicity, chords are only changed at measure boundaries. The particular analysis used in a composition is one of the parameters that can be specified by the user at run-time; thus, pieces of different harmonic structure can be easily generated.

Melody Algorithm

Once a chord progression has been "laid down" by the previous algorithm, the last step in producing a tune is to compose the melody. In order to understand the melody algorithm used in this program, some basics of bluegrass melody, and banjo playing in particular, must be understood.

The 5-string banjo, being a stringed instrument, was originally played by strumming and picking a few melody notes here and there. A major revolution in playing style occurred when Earl Scruggs popularized the "3-finger" style of playing in

the 1950's. This method involves no strumming at all, but consists of picking individual strings with three fingers of the right hand. The particular style used by Scruggs involves interspersing melody notes with "rolls" - arpeggios of notes in the current chord. Since the 5-string banjo uses an open G major tuning, and most songs are in G major, many of the rolls occur on open strings, which adds to the fullness of the sound of the instrument. Melody notes are distinguished from the background notes by emphasizing them in some way (playing them louder, with a different timbre, etc).

During the late '60s and through the '70s, another style came into widespread use. In this style, known as the *melodic* style, almost every note played is a melody note, and there are very few "filler" notes (which is what rolls are, essentially). The melodic style was originally used to imitate fiddle tunes on the banjo, but has since evolved into a sophisticated style incorporating elements of blues, jazz, and classical music. Melodic melodies often incorporate scales, for which a clever method of playing on the banjo with limited left hand movement exists. Sample Scruggs style and melodic style melody fragments are shown in Figure 2.

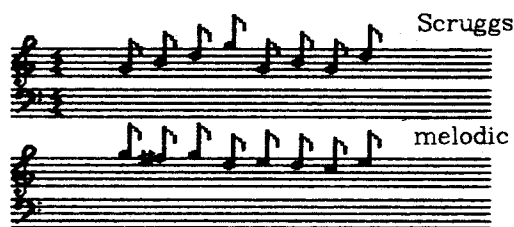


Figure 2: Sample melodies in Scruggs style and melodic style.

Our melody algorithm incorporates elements of both of these two predominant bluegrass styles. Each measure of the piece is randomly chosen as a Scruggs or melodic fragment, and a separate algorithm for each style is used to compute the actual notes. The percentage of each style is one of the parameters that can be changed by the user. This scheme is quite similar to that actually used by many modern bluegrass players, and hence is quite realistic. I will now briefly describe the complete composition program.

The Program

A bluegrass tune traditionally consists of one or more *sections* ("A" section, "B" section, etc). The BANJO program composes each section independently and then concatenates them to form a multi-section piece. This is not unrealistic, since the sections of a bluegrass piece do not usually have much interdependency.

The complete algorithm for one section of a piece is shown in Listing 1. The parameters labelled *p2*, *p3*, *p4*, and so on, are the various parameters (real numbers between 0 and 1, representing probabilities) that can be specified by the user on each run. Files of parameter sets can be stored for later use; thus, if a particularly "nice" set of parameters is discovered, these parameters can be easily recalled during a composition. A typical parameter set is shown in Figure 3.

Also visible in Listing 1 are the details of the melody-generating algorithms. As mentioned earlier, the Scruggs algorithm is

Number of measures = $2n+1$ (n =random integer)

Choose chords using transition probabilities.
(force measures 1 and NM to G major).

Choose rhythms randomly for each measure.

Rhythms are chosen from 3 different sets:
First measure (pickup notes), middle measure,
and last measure (no more than 3 beats).

For each measure, choose notes using this algorithm:

if $\text{rnd} < p2$ then measure=scruggs measure
else measure=melodic measure

if measure=scruggs then

```
{
  note = last note
  + single roll step (if  $\text{rnd} < p3$ )
  + double roll step (if  $\text{rnd} > p4$ )
  + backward roll step (otherwise)
  + whole/half step perturbation (if  $\text{rnd} < p5$ )
    (whole step if  $\text{rnd} < p6$ , half otherwise)
}
```

if measure=melodic then

```
{
  note = last note
  + single scale step (if  $\text{rnd} < p9$ )
  + double scale step (otherwise)
}
```

if $\text{rnd} < p8$ then change direction of scale
if $\text{rnd} < p7$ then restart melody on a new note.

Listing 1: The bluegrass composition algorithm (in structured English). rnd is a random-number-generating function

based on rolls, and the melodic algorithm is based around scales. Note that both algorithms use "first order" dependencies; that is, each note depends to some extent on the previous melody note. This is necessary to avoid the "random sound" that is present in some computer-composed music in which each note is computed independently of the surrounding notes. This is a small step toward the realization of recognizable melodies, motifs, and higher-order musical ideas.

Figure 4 shows a piece generated by the program, both printed in banjo tablature and regular music notation. This is a "typical" piece - neither the best nor worst that the program has composed - but it illustrates a typical piece's quality fairly well. It is particularly striking to notice several typical melodic-banjo fragments

CURRENT PARAMETERS:

HARMONIC FILENAME	ANFILE1
PROB OF MELODIC MELODY	.6
PROB OF SINGLE STEP (SM)	.5
PROB OF BACK STEP (SM)	.1
PROB OF PERTURBATION (SM)	.05
PROB OF WHOLE STEP PERTURB (SM)	.8
PROB OF SEQUENCE RESTART (MM)	.2
PROB OF DIR CHANGE (MM)	.3
PROB OF 2 HALF-STEPS (MM)	.2

Figure 3: Example parameter list used by the composition program. By varying the parameters, pieces of different varieties are produced.

The figure displays two musical representations of a sample piece composed by the BANJO program. On the left, the piece is shown in tablature notation, consisting of five staves. Each staff begins with a 'G' chord indicator, and the notes are represented by numbers (fingering) on a five-line staff. On the right, the same piece is shown in regular music notation, consisting of six staves. Each staff begins with a treble clef and a key signature of one sharp (F#), and the notes are represented by standard musical notation (notes, stems, and beams) on a five-line staff.

Figure 4: Sample piece composed by the BANJO program. The piece is here shown both in tablature and in regular music notation.

appearing (such as the first five notes of the fifth measure), when no information of this type is explicitly included in the algorithm. All in all, I consider this piece, while not a masterpiece, quite pleasant and playable, and not obviously distinguishable from a piece that might have been composed by a person.

Finally, a few statistics on the program itself. It is written in Applesoft (interpretive) BASIC, occupies 4K of memory, and composes a piece the length of the above example in 20 seconds. I believe this program is another indication of the potential of the small computer in music composition and other areas of music research.

References

1. K. Ebcioglu, "Strict Counterpoint by Computer," *Proc. of 1980 International Computer Music Conference*.
2. M. Keith, "NCC: A program that composes canons," *Proc. of 1981 IEEE Symposium on Small Computers in the Arts*, IEEE Press.
3. C. Fry, "Computer Improvisation," *Proc. of 1980 International Computer Music Conference*.
4. M. Keith, "A Combinatorial Aspect of Computer Music Composition," *Proc. of 1979 Personal Computing Festival (NCC)*.

GRAPHICS SOFTWARE INTERACTION USING THE APPLE* COMPUTER

by Ame Choate Flynn

TechniGraphics
111 Hicks Street
Brooklyn, NY 11201

ABSTRACT

A drawing or painting goes through many stages before its completion. Different tools or implements are used for different effects. In this paper I will show how the creation of computer art using separate pieces of software is similar to the traditional method using traditional tools.

INTRODUCTION

As more "paper and print" artists transfer to the world of computer graphics, programmers are recognizing the need for better and easier-to-use software. Hardware peripherals and software utilities mimic traditional tools and also bring new freedom to art and artists.

One of my graphics or illustrations can go through eight or more stages. I use software written for the Apple II Plus to create and enhance my graphics. Certain software utilities may be used for only one purpose, but they may be the best tool for a specific job.

The objective is to know which utility to use when. Just as an artist may prefer oils to watercolors, a computer artist may favor one package over another. The more pieces of software an artist knows, the easier it will be to complete a picture or achieve a desired effect. Using these tools interactively, by bringing the software to the graphic saved on disk, mistakes can be overcome. If a picture is saved at regular intervals, it need never be irretrievably lost. It is always available for another update or enhancement.

SOFTWARE OVERVIEW

APPLE GRAPHICS TABLET

The Apple Graphics Tablet¹ is comparable to the traditional "sketch" process. I use this peripheral to lay out the rough elements of my drawings and may then 'work into' them with other pieces of software.

The Apple Graphics Tablet comes with a software package containing various routines. Perhaps the finest available "quickdraw" routine is

* Apple and Apple II Plus are registered trademarks of Apple Computer Inc., Cupertino, Calif.

included; the line drawn on the screen keep up with the users' hand movements on the tablet. The stylus feels comfortable; it is held and moved like a brush or pencil. 'Mountain' - Figure 1, was created entirely with the Apple Graphics Tablet.



Figure 1.

Other utilities are included on the disk. Some of them, and their uses, are:

Line, a point-to-point line drawing routine. Frame makes an open rectangle. Box creates a 'filled' frame which is useful as an erase function if the box is the same color as the background. Circle creates a circle based upon your midpoint and a radius specification. Slide will move a picture around the screen. To effect a slide you specify a beginning point and the move point.

The Tablet uses the eight Apple colors. Soon these eight colors begin to pall. The ability to fill in large areas with color will seem necessary. Other software utilities now come into play.

MICRO-PAINTER

I use Micro-Painter² to fill in an enclosed area with any of 21 colors or color-patterns. Taking a sketch created on the tablet (Windsurfer - Figure 2) and saved to disk, I can then make the sky light blue, the sea dark blue, and the sail areas various colors. I save the image again and take it back to the tablet - Figure 3 - working into the picture to create waves and rounded shapes.

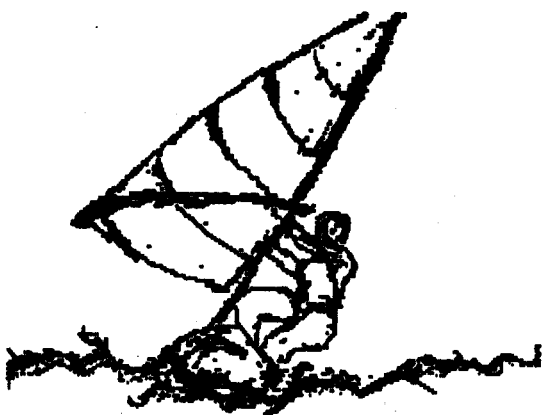


Figure 2.

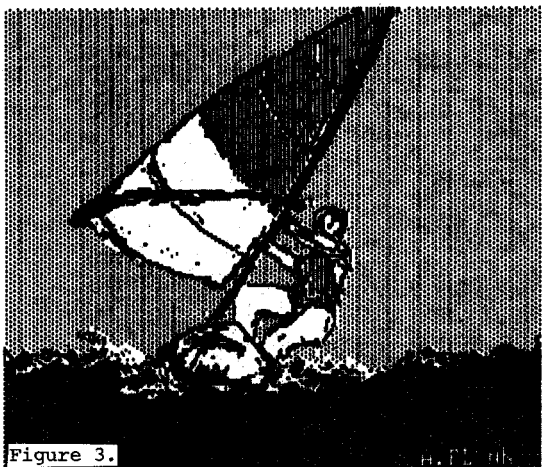


Figure 3.

Micro-Painter uses paddles or a joystick to move the cursor, and the keyboard to enter commands. Black can be changed to white, or green to blue, etc. with the use of the Negative function to change Apple colors to their complements.

A magnified view of the screen is achieved with the Microscope mode. This is a very handy feature when cleaning up text or making miniscule changes.

TEXT

Character Generators and Font Editors reach their intended use in the hands of an artist/designer. In the print or graphics world, text is as important as the design or illustration. The type style should fit the content and be legible.

Character Generators allow you to "type" your text directly on the high-resolution screen, or add it to a previously saved image.

Text or Font Editors let you create new type styles or modify existing ones. Some of the best known text generators and/or font editors are:

E-Z DRAW³ includes many fonts which can be displayed in different sizes, weights and widths.

HIGHER TEXT⁴ has many fonts and an editor which lets you see your font-in-progress in a variety of styles.

THE COMPLETE GRAPHICS SYSTEM⁵ has both a character generator and an editor. It can use fonts from other packages. It incorporates a kerning feature (for tightening up letters) and a leading feature (space between lines) to mimic professional typesetting systems. Figure 4 shows a variety of type styles that have been typed or created using the above software.

Kerned Old English
Non-Kerned
Countdown-Higher Text
Working Font by
Michael Callery
Anne McKay Font by
Ame Flynn

Figure 4.

TABLET-DRIVEN SOFTWARE

The Graphics Tablet is considered the easiest peripheral for use by the artist. In the past year several packages have appeared utilizing the Graphics Tablet. Existing packages have also been modified for use with the tablet.

Artists now have the freedom to choose a 'brush', pick up a color and 'paint' with it directly on the screen. They are no longer restricted to filling within lines. Color can now be put next to color for more dramatic and painterly effects.

SPECIAL EFFECTS⁶ includes a Brush Module for painting with a selection of 96 different brushes. Add these brush options to a choice of 108 colors and color patterns and the possibilities are endless. 'Storm' - Figure 5 was created in this manner.

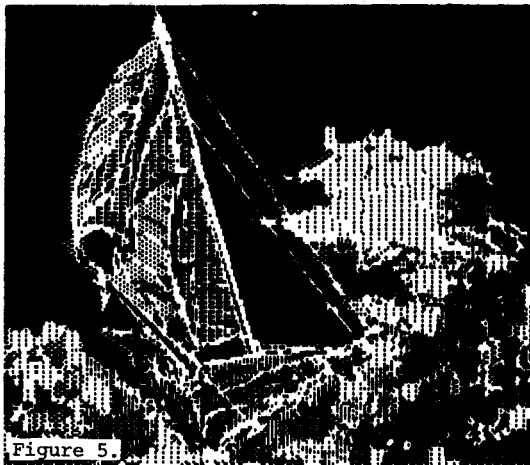


Figure 5.

Other packages for the tablet are THE COMPLETE GRAPHICS SYSTEM, UTOPIA⁷, GRAPHICS PROFESSIONAL SYSTEM⁸, and EDU-PAINT. All packages should be weighed equally when starting a graphic. Just because a package is mentioned in this paper does not imply a recommendation. These happen to be software packages I have used. There are other good graphics packages on the market that I have not mentioned due to space considerations.

CUMBERLAND GAP TO DONNER PASS

An example of this interactive-software process is provided with Figures 6, 7 and 8. These illustrations were produced for educational software (c) KG Productions. I turned the Cumberland Gap into the Donner Pass.

Figure 6 depicts the Cumberland Gap as drawn using the Graphics Tablet. In Figure 7, it has been 'flipped' (left is now right) using the Tricks Module of Special Effects. I then saved the picture, and used the Brush Module of Special Effects to add snow. I again saved the graphic and went back to the Graphics Tablet to create the final Donner Pass illustration. I worked into the snow, modified the mountains and added snow-covered pine trees. Starting this picture from scratch would have taken more time than was available. With the knowledge of what software could do, and when to use it, I saved much repetitive work.

Acknowledgement

Thanks to KG Productions for their permission to reproduce "Cumberland Gap" and "Donner Pass".

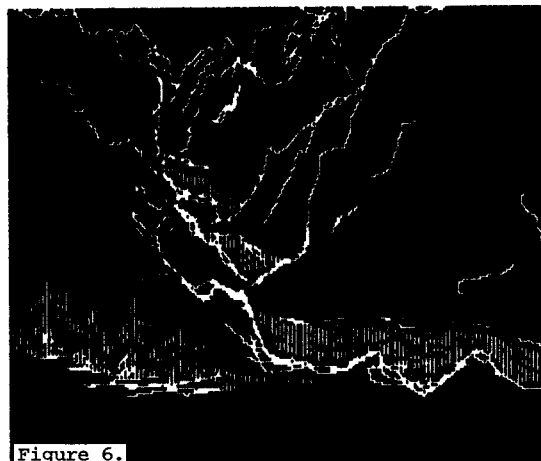


Figure 6.

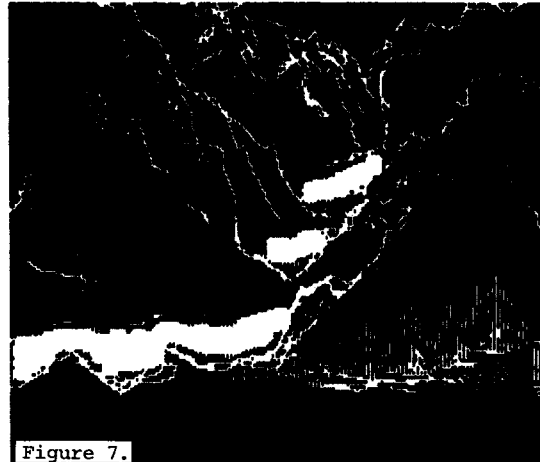


Figure 7.



Figure 8.

References

1. Apple Graphics Tablet by Apple Computer, Inc. Cupertino, California.
2. Micro Painter, (c) Datasoft, Inc., by Bob Bishop.
3. EZ Draw, (c) Sirius Software, by Nasir Gebelli and Jerry W. Jewell.
4. Higher Text, Synergitic Software, by Ron and Darrell Aldrich, (c) 1980.
5. The Complete Graphics System, (c) Penguin Software by Mark Pelezarski.
6. Special Effects, (c) Penguin Software, by Mark Pelezarski and David Lubar.
7. UTOPIA, by Todd Rundgren.
8. GPS, (c) Stoneware Software.
9. EDU-PAINT, (c) Edu-Ware.

MICROCOMPUTER DRAWING

by William J. Kolomyjec, Ph.D., M.F.A.

Department of Engineering Graphics
The Ohio State University, Columbus, Ohio

Abstract

This paper presents graphics software which will enable an Apple II PlusTM microcomputer to draw in an aesthetic manner. The palette consists of the fundamentals of lines, rectangles, circles, arcs, and spline curves. A joystick or paddle driven cursor is used to locate visual components on the screen. A clipping algorithm makes the screen boundaries a viewport for large shapes that exceed the screen limits. This program illustrates the potential of using a small computer as a medium for individual expression.

Introduction

Drawing with the microcomputer is easy if one has the appropriate graphics software. Presently, individuals who seek to do computer drawing must either buy drawing packages or write their own. Ideally, individuals should put together their own graphics package but not everyone has the desire nor, more often, the expertise. The computer art movement exemplified by organizations such as Small Computers in the Arts, should strive to make software available to the potential electronic artist. Consider this an installment for establishment of a new direction in small computer arts practice.

The software presented here incorporates what I feel are the essential components of computer drawing. It is not a commercial package. However, it will present a point of departure for the discussion of arts software.

The overall program design is original. It is comprised of graphic algorithms that are my own, obtained from friends, from public domain (magazines), or from textbooks. A description of the program and each subroutine will be given which will cite specific references. The program will allow the user to make visual statements. The computer becomes a tool in the hands of the user in a manner similar to image-creating tools in other aesthetic media.

Hardware

A standard Apple II PlusTM microcomputer with at least 48K of memory is required. An X-Y joystick with two buttons or two game paddles with buttons is necessary.

Operation

Upon running the program a text menu is displayed. The following FUNCTION OPTIONS are provided:

- Ø = ERASE SCREEN
- 1 = LINE (END POINTS)
- 2 = LINE (CONTINUOUS)
- 3 = RECTANGLES (OPPOSITE CORNERS)
- 4 = CIRCLE (CENTER & RADIUS)
- 5 = ARC (THRU THREE POINTS)
- 6 = SPLINE (UP TO 15 POINTS)
- 7 = EXIT

The user selects an option and presses RETURN. Only option Ø erases the drawing, entering modes 1-6 will not erase the screen. Option 7 will exit the program.

In each of the drawing modes (1-6) depressing paddle button Ø will perform the function, depressing paddle button 1 will return to the function option menu. Spline is an exception: Pushing Ø's will define the curve, pushing 1 will draw it. The mode is not exited until 1 is pushed again after the curve is complete, i.e., twice. That's all there is to it.

Main Program

The main program, lines 100-999, typifies good graphics style. Line 120 keeps the program above the graphics portion of memory.¹ Line 130 jumps to a subroutine that performs initialization of clipping and joystick routines as well as other program variables. Line 140 jumps to the TEXT MENU subroutine.

After the user enters the option, line 150, it is checked for being in the appropriate acceptable response range. Line 170 branches to the chosen function. Each function is a subroutine in itself. A branch to each function entails: 1) switching into graphics without screen erasure, 2) performing the function, and 3) upon return, the menu is re-displayed and control is transferred to line 150 once again. The exceptions are clear screen and exit.

Subroutines

CPLOTSUB, 1000-1070. This subroutine converts X, Y, P plot codes into X1,Y1,X2,Y2 vectors. It

checks for valid pen control values, line 1020. Once a vector or line segment is obtained the clipping routine is called, line 1040. Clipping works as follows: If the line segment is off the screen, nothing happens. If the line segment is partially off the screen, clipping is performed and it is plotted. If the line is totally on the screen, it is plotted.

DON'T ERASE, 1250-1260. A series of soft switches are POKE'd to re-enter graphics without clearing screen memory.²

RECTANGLE SUB, 1500-1550. This general rectangle drawing algorithm generates the plotcodes for a rectangle given a starting location and a height and width. The source of this routine and others like it is Computer Graphics,³ a fine computer graphics primer/workbook.

ARC SUB, 1700-1800. This is a general arc drawing subroutine that requires 6 parameters: X and Y coordinates of the center, a radius, a beginning and final angle (in degrees) and a flag to indicate drawing direction, clockwise or counter clockwise. The sampling rate varies and is determined by the radius.

DELAY, 2000-2010. This is a short subroutine that does nothing more than waste time. It acts as an adjustment to paddle button sensitivity (the heavier your touch - the longer the delay). The variable determining the duration of this loop is located in line 8020.

LINE1, 3000-3080. This is the first function option subroutine. The cursor is displayed (line 3010). Depressing button 0 will continually define the respective end points of lines. After the first end point is entered that pixel (discrete screen coordinate) is turned on (line 3030), after the second location is entered, the line segment is drawn (line 3060). Depressing button 1 will exit the routine and consequently return to the main menu.

LINE2, 3500-3590. This routine is similar to the LINE1 subroutine with the exception that every time a location is entered after the second location, the line is connected to the previous location producing a continuous line. Depressing button 1 will exit the routine and return to the menu.

CURSOR RECTANGLES, 4000-4120. Two locations are entered via positioning the cursor defining opposite corners of a rectangle. These locations are momentarily displayed. The computer checks the coordinates of these locations, determines the smallest X and Y, switching them if necessary, and calculates the height and width of the rectangle (lines 4070-4090). With this data the RECTANGLE subroutine is called (line 4100). This process continues until button 1 is depressed.

CIRCLE, 4500-4620. This function routine uses the ARC SUB and hence the variables: FANG,

LANG and IDIR% are immediately defined. The remaining variables, XCT, YCT, and R are defined after a center location and a point on the circumference are provided, using the cursor. The radius is computed using the Pythagorean Theorem in line 4600. Note that before the circle is drawn the center location pixel is turned off, line 4590. This is a clue for "selective erasure." The point is simply drawn over in black (HCOLOR = 4).

With all six parameters required by the ARC subroutine defined, a circle can be plotted. Note that it is possible for part of the circle to lie off the visible screen or outside the viewport. Fortunately CPLOTSUB in conjunction with CLIP takes care of these occurrences. This function routine, like the others, will continue in this mode until button 1 is depressed.

ARC THRU THREE POINTS, 5000-5220. The logic in this routine is by far the most complex to this point, save the spline algorithm. Once three locations are defined using the cursor, an arc of a circle will be drawn through those points in the order that they are provided. Clearly, use of the existing ARC subroutine is appropriate but this tends to add to the complexity of this routine. There are probably easier ways to do this but I haven't found anything else that will work as well.

The key to this routine is the use of the THETA function which returns the angle (in radians) formed between a point in a plane and the plus X-axis vector of that coordinate system. But this is only half the problem. Based on the parameters required by the ARC subroutine a starting angle, FANG, ending angle, LANG, and a direction are required. This is accomplished in lines 5140-5190. Incidentally, MANG is the middle angle between FANG and LANG which determines the appropriate arc (and not its complement) that is required.

SPLINE, 5500-5670 and CURVE COORD FINDER, 6500-6560. These two routines need not be separate but are in order to make the code easier to understand. This algorithm was given to me by a friend, Christopher Scussel. Anyway, I'm not sure where he got it, or why it works, but it does.

The routine gathers screen locations into an array, plotting and storing them until button 1 is pushed. Note: this is the one exception to exiting from a function routine. The algorithm then takes over and plots a smooth curve or spline curve through the points. Truly amazing! The cursor reappears when the curve is complete and, unless button 1 is depressed to exit, another curve may be defined and plotted.

CURSOR2SUB, 7000-7070. This is the heart of the interactive graphics capability of this software. The cursor, which is a high-resolution shape, is controlled using the analog inputs (joystick or paddles). The location of the center of the cursor at any time it appears on the screen is

X%, Y%. The cursor is deliberately confined to the Page2 graphics screen (280 X 192) and hence X% can only be in the range 0 to 279 and Y% from 0 to 191. The loop on line 7030 controls the blinking quality of the cursor and must be performed an even number of times or little cursors get left behind. See the Applesoft manual⁴ or Apple II User's Guide⁵ for more about XDRAW. The two IF statements, lines 7040-7050, check the paddle button status.⁶ If either button is depressed the button flag variable, FL%, is redefined and the routine is exited.

INITCURSORSUB, 7500-7560. This routine is only used once at the beginning of the program. It POKE's a shape (the cursor) into an obscure part of memory (\$3000 - \$30FF). This shape can then be used by the XDRAW function in the cursor routine.

COMPUTE CENTER AND RADIUS, 8500-8570. This algorithm was taken from a book of BASIC algorithms.⁷ Once the errors it was published with were removed, it was implemented to calculate the center and radius of a circle given three points. It has also been improved to prevent divide-by-zero errors (by adding 0.00001 in appropriate places) when three points get too close to coinciding with a straight line where, of course, there is no answer.

THETA FUNCTION, 8750-8850. This subroutine returns an angle when given a coordinate location. Its function was previously discussed in the ARC THRU THREE POINTS subroutine narrative. The credit here again goes to C. Scussel.

TEXT MENU, 9000-9110. The main menu is provided by this routine. Upon its execution the text screen is cleared and the function options list is printed. Control is then transferred back to the main program. I have found that a graphics menu works better, that is, using the keyboard is too visually distracting. A graphics menu allows the cursor position to "pick" the function option. This version of the program will not be given.

CLIPPING INIT, 10000-10030. Like INITCURSORSUB this routine is only used once by the program. It performs a little housekeeping and sets the string variables used by the clipping routine to the null string, line 10010. Line 10020 defines the boundaries of the viewport which exactly coincides with the screen limits.

CLIP SUBROUTINE, 20000-20430. This is the famous Cohen-Sutherland clipping algorithm.⁸ It is somewhat ill-suited to BASIC but works well enough. This particular adaptation was brought to my attention via a magazine article.⁹ I have modified it to work more efficiently in conjunction with CLOTSUB but it slows the program down and takes a lot of program memory. Without clipping, fatal errors would occur any time the screen boundaries were exceeded causing the program to halt prematurely. Clipping is a must for the majority of interactive graphics programs.

Conclusion

Computer software is what gives intelligence to the computer enabling it to function as a tool. The program given at the end of this article will enable an Apple IITM microcomputer to perform like an aesthetic tool. In keeping with my philosophy¹⁰ the mind which is the source of inspiration is facilitated by the hand using the computer with appropriate software to generate or create a visual statement. Software can be art too!

This program was written by me in my own style. I like to use mnemonic variable names and I tend to write in a manner that makes things clear for me. Programming style, like aesthetics, is very personal. Individual style is revealed in programs because people write them. The Walt Disney film, TRON, presented programs as personifications which emphasize this point.

A final word about the Applesoft code that follows. It had to be compressed. The majority of the REM statements were removed and multiple statements per line had to be used extensively.¹¹ This technique saves space and makes the program fit into memory but makes it less readable and for this I apologise. Compiling really improves the speed but requires expensive compiler software.

Take the time to enter it into an Apple. Learn from it! Make beautiful pictures with it and, most of all, enjoy it. Any art making experience should be positive. Try computer drawing as a medium for your personal expression.

References

1. Poole, L. et al. Apple II User's Guide. Osborne/McGraw Hill. Berkeley, Ca.: 1981. p. 204.
2. Ibid. p. 205.
3. Demel, J.T. et al. Computer Graphics. Creative Publishing Company. College Station, TX.: 1979. p. 25.
4. Applesoft: Basic Programming Reference Manual. Apple Computer Co. (Product # A2L0006). Cupertino, CA.: 1981. p. 98.
5. Poole, L. et al. Apple II User's Guide. Osborne/McGraw Hill. Berkeley, CA.: 1981. p. 312.
6. Ibid. p. 344.
7. Tracton, K. 57 Practical Programs & Games in Basic. TAB Books. Blue Ridge Summit, PA.: 1978. p. 30.
8. Newman, W.M. and Sproull, R.F. Principles of Interactive Computer Graphics. McGraw-Hill Book Co. N.Y.: 1973. p. 123.

9. Hansen, C. "A graphics package for the Apple." Creative Computing Magazine, Vol. 8, No. 7, July 82, p. 110.
10. Kolomyjec, W.J. "Thoughts on computer aesthetics and the future role of small computers." Proceedings Symposium on Small Computers in the Arts. Philadelphia, PA. Nov. 20-22, 1981. pp. 59-62.
11. Applesoft: Basic Programming Reference Manual. Apple Computer Co. (Product # A210006). Cupertino, CA.: 1981. pp. 118-120.

Program Listing

```

100 REM <<<<<<<< COMPUTER DRAWING >>>>>
>>>>
110 REM <<< COPYRIGHT 1982 W.J.KOLOMYJEC
>>>
120 LOMEM: 24576
130 GOSUB 8000
140 GOSUB 9000
150 PRINT : INPUT "ENTER OPTION:";O%
160 IF O% < 0 OR O% > 7 THEN 140
170 ON O% + 1 GOTO 180,190,200,210,220,23
0,240,250
180 HGR2 : GOTO 140: REM ERASE
190 GOSUB 1250: GOSUB 3000: GOTO 140
200 GOSUB 1250: GOSUB 3500: GOTO 140
210 GOSUB 1250: GOSUB 4000: GOTO 140
220 GOSUB 1250: GOSUB 4500: GOTO 140
230 GOSUB 1250: GOSUB 5000: GOTO 140
240 GOSUB 1250: GOSUB 5500: GOTO 140
250 HOME : VTAB 10: PRINT "END COMPUTER D
RAWING"
999 END
1000 REM <<<<< CLOTSUB >>>>>
1010 IF P = 2 THEN X1 = X:Y1 = Y: RETURN

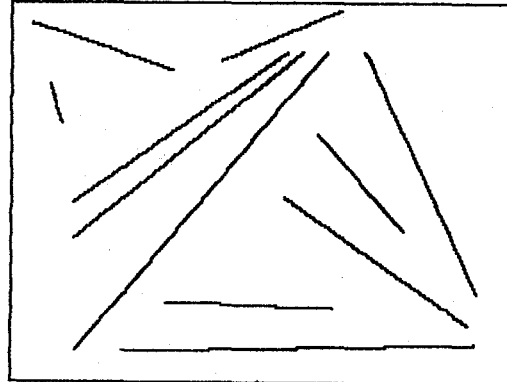
1020 IF P < > 1 THEN STOP
1030 X2 = X:Y2 = Y
1040 GOSUB 20000: REM CLIPSUB
1050 IF OUT = 1 THEN 1070
1060 HPLLOT X1,Y1 TO X2,Y2
1070 X1 = X:Y1 = Y: RETURN
1250 REM <<<<< DON'T ERASE >>>>>
1260 POKE 49232,0: POKE 49239,0: POKE 492
37,0: RETURN
1500 REM <<<<< RECTANGLE SUB >>>>>
1510 X = XR:Y = YR:P = 2: GOSUB 1000
1520 X = XR + W:Y = YR:P = 1: GOSUB 1000
1530 X = XR + W:Y = YR + H:P = 1: GOSUB 10
00
1540 X = XR:Y = YR + H:P = 1: GOSUB 1000
1550 X = XR:Y = YR:P = 1: GOSUB 1000: RETU
RN
1700 REM <<<<< ARC SUB >>>>>
1710 IF R < = 8 THEN NS = 10: GOTO 1760
1720 IF R > = 9 AND R < 16 THEN NS = 20:
GOTO 1760
1730 IF R > = 17 AND R < 32 THEN NS = 30
: GOTO 1760
1740 IF R > = 33 AND R < 64 THEN NS = 45
: GOTO 1760
1750 IF R > = 65 THEN NS = 90
1760 AL = ABS (LANG - FANG):N = INT ((AL
/ 360) * NS + 0.5)

```

```

1770 F = FANG / 57.295779:L = LANG / 57.29
5779
1780 FOR J = 1 TO N
1790 PCT = (J - 1) / (N - 1 + 0.00001)
1800 IF IDIR% < 0 THEN PCT = 1.0 - PCT
1810 AN = (L - F) * PCT + F
1820 X = INT (R * COS (AN) + XCT + 0.5):
Y = INT (R * SIN (AN) + YCT + 0.5)
1830 IF J < > 1 THEN 1850
1840 P = 2: GOSUB 1000: GOTO 1860
1850 P = 1: GOSUB 1000
1860 NEXT J: RETURN
2000 REM <<<<< DELAY >>>>>
2010 FOR D0 = 1 TO DL: NEXT D0: RETURN

```

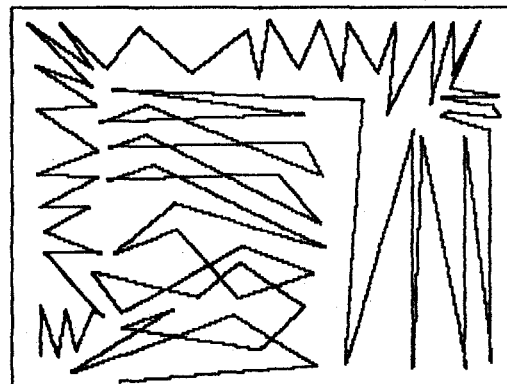


```

3000 REM <<<<< LINE1 >>>>>
3010 GOSUB 7000: REM CURSOR
3020 IF F1% = 1 THEN 3080
3030 X1 = X%:Y1 = Y%: HPLLOT X1,Y1
3040 GOSUB 2000: GOSUB 7000
3050 IF F1% = 1 THEN 3080
3060 X2 = X%:Y2 = Y%: HPLLOT X1,Y1 TO X2,Y2

3070 GOSUB 2000: GOTO 3010
3080 RETURN

```

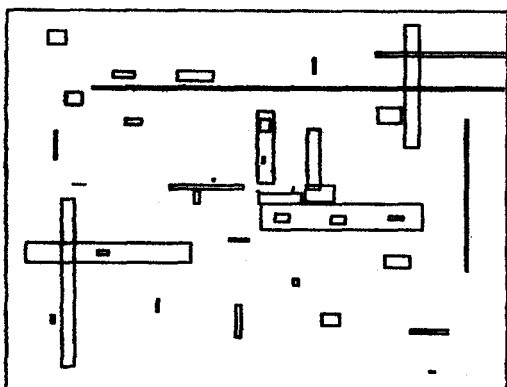


```

3500 REM <<<<< LINE2 >>>>>
3510 GOSUB 7000
3520 IF F1% = 1 THEN 3590
3530 X1 = X%:Y1 = Y%: HPLLOT X1,Y1
3540 GOSUB 2000
3550 GOSUB 7000
3560 IF F1% = 1 THEN 3590
3570 X2 = X%:Y2 = Y%: HPLLOT X1,Y1 TO X2,Y2

3580 GOSUB 2000:X1 = X2:Y1 = Y2: GOTO 355
0
3590 RETURN

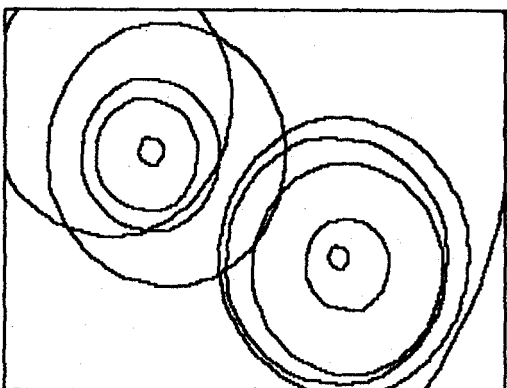
```



```

4000 REM <<<<< CURSOR RECTANGLES >>>>>
4010 GOSUB 7000
4020 IF F1% = 1 THEN 4120
4030 XR = X%:YR = Y%: HPLLOT XR,YR
4040 GOSUB 2000: GOSUB 7000
4050 IF F1% = 1 THEN 4120
4060 XS = X%:YS = Y%: HPLLOT XS,YS
4070 IF XS < XR THEN T = XR:XR = XS:XS = T
4080 IF YS < YR THEN T = YR:YR = YS:YS = T
4090 H = ABS (YS - YR):W = ABS (XS - XR)
4100 GOSUB 1500: REM RECTSUB
4110 GOSUB 2000: GOTO 4010
4120 RETURN

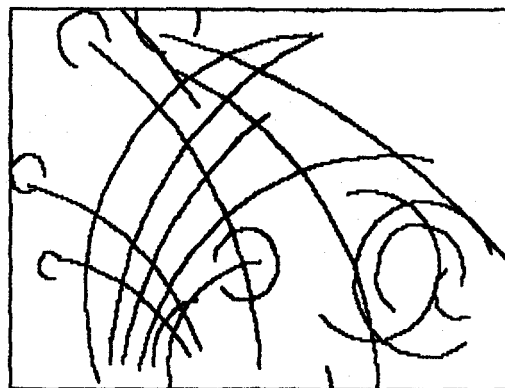
```



```

4500 REM <<<<< CIRCLE >>>>>
4510 FANG = 0:LANG = 360:IDIR% = - 1
4520 GOSUB 7000
4530 IF F1% = 1 THEN 4620
4540 X1 = X%:Y1 = Y%: HPLLOT X1,Y1
4550 GOSUB 2000: GOSUB 7000
4560 IF F1% = 1 THEN 4620
4570 X2 = X%:Y2 = Y%: HPLLOT X2,Y2
4580 GOSUB 2000
4590 HCOLOR= 4: HPLLOT X1,Y1: HCOLOR= 3: REM REMOVE CENTER
4600 XD = ABS (X2 - X1):YD = ABS (Y2 - Y1):R = SQR (XD * XD + YD * YD)
4610 XCT = X1:YCT = Y1: GOSUB 1700: GOTO 4520
4620 RETURN

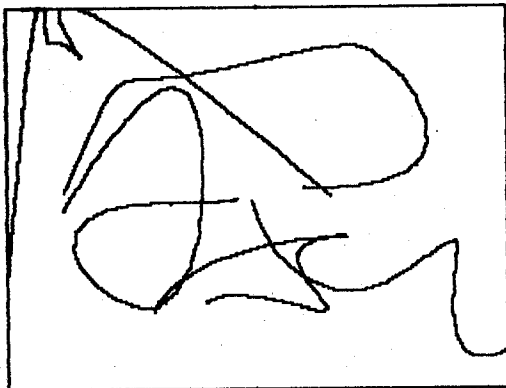
```



```

5000 REM <<<<< ARC THRU THREE POINTS >>>>>
5010 GOSUB 7000
5020 IF F1% = 1 THEN 5220
5030 X1 = X%:Y1 = Y%: HPLLOT X1,Y1
5040 GOSUB 2000: GOSUB 7000
5050 IF F1% = 1 THEN 5220
5060 X2 = X%:Y2 = Y%: HPLLOT X2,Y2
5070 GOSUB 2000: GOSUB 7000
5080 IF F1% = 1 THEN 5220
5090 X3 = X%:Y3 = Y%: HPLLOT X3,Y3
5100 GOSUB 2000: GOSUB 8500: REM CALC X0,Y0,R
5110 XCT = X0:YCT = Y0:XX = X1 - XCT:YY = Y1 - YCT: GOSUB 8750:FANG = THETA * 57.295779
5120 XX = X2 - XCT:YY = Y2 - YCT: GOSUB 8750:MANG = THETA * 57.295779
5130 XX = X3 - XCT:YY = Y3 - YCT: GOSUB 8750:LANG = THETA * 57.295779
5140 IF FANG < MANG AND MANG < LANG THEN IDIR% = 1: GOTO 5200
5150 IF FANG < LANG AND LANG < MANG THEN IDIR% = - 1:TEMP = LANG:LANG = FANG + 360:FANG = TEMP: GOTO 5200
5160 IF MANG < FANG AND FANG < LANG THEN IDIR% = - 1:TEMP = LANG:LANG = FANG + 360:MANG = TEMP: GOTO 5200
5170 IF MANG < LANG AND LANG < FANG THEN IDIR% = 1:MANG = MANG + 360:LANG = LANG + 360: GOTO 5200
5180 IF LANG < FANG AND FANG < MANG THEN IDIR% = 1:LANG = LANG + 360: GOTO 5200
5190 IDIR% = - 1:TEMP = LANG:LANG = FANG:FANG = TEMP
5200 GOSUB 1700
5210 GOTO 5010
5220 RETURN

```



```

5500 REM <<<<< SPLINE >>>>>
5510 K = 1
5520 GOSUB 7000: GOSUB 2000
5530 IF F1% = 1 THEN 5590
5540 HPLLOT X%,Y%
5550 C(K,1) = X%:C(K,2) = Y%
5560 K = K + 1
5570 IF K > NP% THEN K = NP%: REM LIMIT
CHECK
5580 GOTO 5520
5590 KNT = K - 1: NS = IRS * KNT: P = 2
5600 FOR J = 1 TO NS
5610 T = (J - 1) / (NS - 1) * (KNT - 1) +
1
5620 GOSUB 6500
5630 IF J < > 1 THEN P = 1
5640 GOSUB 1000
5650 NEXT J
5660 GOSUB 7000: IF F1% = 1 THEN RETURN

5670 GOSUB 2000: K = 1: GOTO 5540
6500 REM <<<<< CURVE COORD FINDER >>>>>
6510 X = 0: Y = 0: D = 0: S = - 1
6520 FOR I = 1 TO KNT
6530 F = EXP (T - I + 0.001): F = 2 / (F -
1 / F)
6540 X = X + S * C(I,1) * F: Y = Y + S * C
(I,2) * F
6550 D = D + S * F: S = - S
6560 NEXT I: X = X / D: Y = Y / D: RETURN
7000 REM <<<<< CURSOR2SUB >>>>>
7010 XP = PDL (0) / 255: YP = PDL (1) / 2
55
7020 X% = 279 * XP + 0.5: Y% = 191 * YP + 0
.5
7030 FOR J = 1 TO LOOP: XDRAW 1 AT X%,Y%:
NEXT J
7040 IF PEEK (P0) > 127 THEN F1% = 0: GO
TO 7070
7050 IF PEEK (P1) > 127 THEN F1% = 1: RE
TURN
7060 GOTO 7010
7070 RETURN
7500 REM <<<<< INITCURSOR2SUB >>>>>
7510 ROT = 0: SCALE = 2: LOOP = 6
7520 DATA 1,0,4,0,36,76,145,58
7530 DATA 63,63,23,82,9,36,4,0
7540 FOR J = 768 TO 783: READ D: POKE J,D
: NEXT J
7550 POKE 232,0: POKE 233,3
7560 RETURN

```

```

8000 REM <<<<< INIT PROGRAM VARIABLES >>
>>>
8010 HGR : HGR2 : HCOLOR = 3
8020 DL = 50: P0 = 49249: P1 = 49250: F1% = 0

8030 GOSUB 7500: REM INIT CURSOR
8040 GOSUB 10000: REM CLIPPING INIT
8050 IRS = 5: NP% = 15: DIM C(NP%,2)
8060 RETURN
8500 REM <<<<< COMPUTE CENTER & RADIUS >
>>>>
8510 T1 = (Y2 - Y1) / (X2 - X1 + 0.00001):
T2 = (Y3 - Y1) / (X3 - X1 + 0.00001)
8520 T3 = ((X2 - X1) * (X2 + X1)) + ((Y2 -
Y1) * (Y2 + Y1))
8530 T4 = T3 / (2 * (X2 - X1) + 0.00001)
8540 T5 = ((X3 - X1) * (X3 + X1)) + ((Y3 -
Y1) * (Y3 + Y1))
8550 T6 = T5 / (2 * (X3 - X1) + 0.00001)
8560 Y0 = (T6 - T4) / (T2 - T1): X0 = T6 -
(T2 * Y0)
8570 R = SQR ((X3 - X0) ^ 2 + (Y3 - Y0) ^
2): RETURN

8750 REM <<<<< THETA FUNCTION >>>>>
8760 PI = 4 * ATN (1.): UNDF = 0
8770 IF XX < 0 THEN THETA = PI + ATN (YY
/ XX): RETURN
8780 IF XX = 0 THEN 8800
8790 GOTO 8830
8800 IF YY < 0 THEN THETA = PI * 1.5: RET
URN
8810 IF YY = 0 THEN THETA = UNDF: RETURN

8820 THETA = PI / 2: RETURN
8830 IF YY < 0 THEN THETA = 2 * PI + ATN
(YY / XX)
8840 IF YY = 0 THEN THETA = 0: RETURN
8850 THETA = ATN (YY / XX): RETURN
9000 REM <<<<< TEXT MENU >>>>>
9010 HOME : TEXT : VTAB 5
9020 PRINT "FUNCTION OPTIONS:"
9030 PRINT : PRINT " 0=ERASE SCREEN"
9040 PRINT " 1=LINE (END POINTS)"
9050 PRINT " 2=LINE (CONTINUOUS)"
9060 PRINT " 3=RECTANGLE (OPPOSITE CORN
ERS)"
9070 PRINT " 4=CIRCLE (CENTER & RADIUS)"
"
9080 PRINT " 5=ARC (THRU THREE POINTS)"
"
9090 PRINT " 6=SPLINE (UP TO 15 POINTS)"
"
9100 PRINT " 7=EXIT"
9110 RETURN
10000 REM <<<<< CLIPPING INIT >>>>>
10010 X = FRE (0): EH$ = "": ET$ = "": E1$ =
"": E2$ = ""
10020 VL = 0: VR = 279: VB = 0: VT = 191
10030 RETURN

```

```

20000 REM <<<<< CLIP SUBROUTINE >>>>>
20010 GOSUB 20250: GOSUB 20350
20020 IF E1$ = "0000" AND E2$ = "0000" TH
EN OUT = 0: RETURN
20030 ET$ = LEFT$ (E1$,1): IF ET$ = LEFT
$ (E2$,1) AND ET$ < > "0" THEN OUT = 1: R
ETURN
20040 ET$ = MID$ (E1$,2,1): IF ET$ = MID
$ (E2$,2,1) AND ET$ < > "0" THEN OUT = 1:
RETURN
20050 ET$ = MID$ (E1$,3,1): IF ET$ = MID
$ (E2$,3,1) AND ET$ < > "0" THEN OUT = 1:
RETURN
20060 ET$ = RIGHT$ (E1$,1): IF ET$ = RIG
HT$ (E2$,1) AND ET$ < > "0" THEN OUT = 1:
RETURN
20070 EH$ = E1$: IF EH$ = "0000" THEN EH$
= E2$
20080 IF LEFT$ (EH$,1) = "0" THEN 20110
20090 Y3 = Y1 + (Y2 - Y1) * (VL - X1) / (X
2 - X1): X3 = VL
20100 GOTO 20190
20110 IF MID$ (EH$,2,1) = "0" THEN 20140
20120 Y3 = Y1 + (Y2 - Y1) * (VR - X1) / (X
2 - X1): X3 = VR
20130 GOTO 20190
20140 IF MID$ (EH$,3,1) = "0" THEN 20170
20150 X3 = X1 + (X2 - X1) * (VB - Y1) / (Y
2 - Y1): Y3 = VB
20160 GOTO 20190
20170 IF RIGHT$ (EH$,1) = "0" THEN 20190
20180 X3 = X1 + (X2 - X1) * (VT - Y1) / (Y
2 - Y1): Y3 = VT
20190 IF EH$ = E1$ THEN 20220
20200 X2 = X3: Y2 = Y3: GOSUB 20340
20210 GOTO 20020
20220 X1 = X3: Y1 = Y3: GOSUB 20250
20230 GOTO 20020

```

```

20240 REM ENDPOINT CODES X1,Y1
20250 E1$ = ""
20260 IF X1 < VL THEN E1$ = "1": GOTO 202
80
20270 E1$ = "0"
20280 IF X1 > VR THEN E1$ = E1$ + "1": GO
TO 20300
20290 E1$ = E1$ + "0"
20300 IF Y1 < VB THEN E1$ = E1$ + "1": GO
TO 20320
20310 E1$ = E1$ + "0"
20320 IF Y1 > VT THEN E1$ = E1$ + "1": RE
TURN
20330 E1$ = E1$ + "0": RETURN
20340 REM ENDPOINT CODES X2,Y2
20350 E2$ = ""
20360 IF X2 < VL THEN E2$ = "1": GOTO 203
80
20370 E2$ = "0"
20380 IF X2 > VR THEN E2$ = E2$ + "1": GO
TO 20400
20390 E2$ = E2$ + "0"
20400 IF Y2 < VB THEN E2$ = E2$ + "1": GO
TO 20420
20410 E2$ = E2$ + "0"
20420 IF Y2 > VT THEN E2$ = E2$ + "1": RE
TURN
20430 E2$ = E2$ + "0": RETURN
22222 REM <<<<< END COMPUTER DRAWING >>>
>>

```


ANALYTICAL COMPUTER ART

Joe Jacobson

675 E. Street Road, Apt. 1009
Warminster, PA 18974

Abstract

Analytical computer art, consisting of geometric designs based on explicit mathematical functions, is discussed in terms of 3 fundamental types of algorithms. An example is given of each type, with sample output patterns included.

Introduction

One of the oldest forms of visual computer art is the geometric design, or "line drawing;" pictures of this type have been generated ever since the first plotting peripherals became available. The field is still lively and today a great deal of experimentation and innovation has been done, in an attempt to exploit the potential of the high-speed, high-resolution graphics systems that are currently available. While most home computer CRT monitor systems do not fall under this category, it is possible to achieve high spatial resolution with a personal computer system if it includes a good mechanical plotter. In addition, an increasing number of artists are gaining access to high-performance laboratory-quality systems.

I create geometric designs on a Tektronix 4054 computer terminal, which contains a 16-bit micro-processor and can be used in "stand-alone" mode without involving a larger, remote computer. I use the Tektronix "PLOT 50" software system, which includes a version of BASIC language and some machine-specific graphics commands¹.

Types of Computer Art Programs

My computer art is "analytical" in the sense that it is based on explicit mathematical functions expressed in either rectangular or polar coordinates. These functions are typically algebraic, trigonometric, or exponential in nature. The resulting computer programs fall into one of three classes.

The first type of program is custom-designed to generate a specific, pre-conceived geometric pattern. The second type allows the user to prescribe values for a set of input parameters, with each specific combination of inputs resulting in a unique output pattern. The third type of program is a combination of the first two; a particular output pattern is selected that resulted from a program of the second type, and a new program (of the third type) is written to deliberately improve this pattern in some way. As an example, the visual contrast of the

picture may be enhanced.

Examples of outputs from all three types of programs will be presented here. BASIC listings will be given for two of the programs.

An Example of the First Type of Program

A program of the first type resulted in the pattern shown in figure 1. This picture is a 3-D perspective plot of the surface generated by a function of 2 variables. This function, which has the form $z = f(x, y)$, is given by the following expression:

$$z = 5xe^{-x} \left| \sin(60y) \right| + 0.9e^{-10(x-0.95)^2} \left[e^{-10(y-1.5)^2} + e^{-10(y-4.5)^2} \right] \quad (1)$$

where the argument of the sine function is interpreted in units of degrees (not radians) and the surface is plotted over the domain $0 \leq x \leq 6$, $0 \leq y \leq 6$. While this complicated function may seem somewhat abstract, the resulting pattern, shown in figure 1, will probably remind most viewers of a certain biological form. I "tailor-made" the function expressed in equation (1) to generate this pattern. The particular combination of elementary functions used, as well as the values of all constants, were selected accordingly.

In this program the numerical parameter values can, in fact, be varied slightly in order to effect desired changes in the pattern. An example would be selection of a value larger, or smaller, than 0.9 for the coefficient of the gaussian "fine structure" term, which can have an interesting psychological effect on male viewer response to the pattern. However, the overall pattern shown in figure 1 is basically the only output of the particular computer program used.

It might be noted in passing that this program did not use hidden-line suppression techniques; instead, the viewing angle was chosen to minimize the effects of hidden lines. The program was translated into BASIC from a low-level calculator language and a listing is not given here because it would take up several pages. However, if the 3-D perspective transformation equations had been coded directly into BASIC, the program would probably be no more than a tenth as long.

An Example of the Second Type of Program

An example of the second type of program is the "Composite Loop" program, for which a listing is included here.

```

100 REM COMPOSITE LOOP PROGRAM
110 PAGE
120 PRINT "ENTER L TO GENERATE A PARTICULAR PICTURE."
130 INPUT L
140 PRINT "ENTER D TO ADJUST SPACING"
150 PRINT "BETWEEN ANGULAR SWEEPS."
160 PRINT "SELECT A VALUE BETWEEN 30 AND 50 FOR D."
170 INPUT D
180 PRINT "ENTER C TO ADJUST ANGULAR INCREMENT."
190 INPUT C
200 PRINT "ENTER ANGULAR FREQUENCY K."
210 INPUT K
220 PRINT "ENTER INTERNAL WEIGHTS. U1,U2 ."
230 INPUT U1,U2
240 PRINT "ENTER EXTERNAL WEIGHTS. V1,V2 ."
250 INPUT V1,V2
260 PRINT "ENTER T1"
270 INPUT T1
280 PAGE
290 SET DEGREES
300 IF T1=2 THEN 380
310 VIEWPORT 22,100,14,100
320 GO TO 310
330 VIEWPORT 15,115,0,100
340 FOR B=0 TO 360 STEP D
350 FOR A=0 TO 360 STEP C
400 S=ATN(V1/V2)
410 T=360.01*(V1*SIN(S)+V2*COS(S))
420 WINDOW -T,T,-T,T
515 Z=K*(PI/2)*(U1*SIN(L*A)+U2*COS(L*A))
520 R=B*(V1*SIN(Z)+V2*COS(Z))
740 X=R*COS(A)
750 Y=R*SIN(A)
760 IF A>0 THEN 700
770 MOVE X,Y
780 GO TO 800
790 DRAW X,Y
800 NEXT A
810 NEXT B
820 IF T1=1 THEN 840
830 END
840 WINDOW 0,130,0,100
850 VIEWPORT 0,130,0,100
860 MOVE 0,10
870 PRINT "COMPOSITE LOOP PROGRAM PARAMETERS"
880 MOVE 0,0
890 PRINT "(L,D,C,K) = (":L:","D:","C:","K:");"
900 PRINT "(U1,U2,V1,V2) = (":U1:","U2:","V1:","V2:");"
910 INPUT J0
920 T1=2
930 GO TO 250

```

This program prompts the user to select values for eight input parameters. The output pattern is very sensitive to these input values and a large variety of pictures, of varying aesthetic appeal, can result from the use of this program. Once a program of this type is written, the artist tries to exploit its potential by repeated selection of sets of input values. This can be done at random, but better results are achieved if the user applies mathematical and artistic intuition tempered by a knowledge of how the program works. Some sample output patterns, with the corresponding input values displayed as annotations, are shown in figures 2 through 9.

Programs of this second type illustrate a practical paradox associated with computers. In principle, computers function in a totally deterministic manner; a given algorithm will, for a certain set of inputs, always yield the same output pattern. (The set of inputs may include

seed inputs to a random number generator; if all the inputs are known, the output is repeatable.) However, from a practical standpoint, the results can seem quite random and unpredictable; there is no way of predicting the exact output that will result from previously untried combinations of input parameters, except by cranking through the calculation manually—which in many cases would take a human lifetime to do. However, in using programs like Composite Loop, intuition has a definite place; it is a trans-logical process that can help the artist increase the frequency of appearance of aesthetically desirable outputs.

An Example of the Third Type of Program

A program of the third type is typified by the "Gear" algorithm, for which a listing is included here.

```

90 REM GEAR
91 REM (FILE 42)
100 PAGE
110 SET DEGREES
120 WINDOW -501.501,-501.501
130 VIEWPORT 15,115,0,100
140 PAGE
150 L=40
160 PAGE
170 FOR B=100 TO 400 STEP 6
180 FOR A=0 TO 360 STEP 5
190 COSUB 100
200 IF A>0 THEN 150
210 MOVE X,Y
220 GO TO 160
230 DRAW X,Y
240 NEXT A
250 NEXT B
260 GO TO 211
270 R=B*(1+0.25*ABS(SIN(L*A)))
280 X=R*COS(A)
290 Y=R*SIN(A)
300 RETURN
310 FOR N=0 TO L STEP 0.3
320 R=100
330 T=N*(100/L)
340 X=R*COS(T)
350 Y=R*SIN(T)
360 MOVE X,Y
370 X=R*COS(T+100)
380 Y=R*SIN(T+100)
390 DRAW X,Y
400 NEXT N
410 END

```

This routine had its origins in an earlier program, called "Sinusoidal Loop No. 2," which is of the second type. This program will be described in detail elsewhere². Selection of a particular set of input parameters with "Sinusoidal Loop No. 2" resulted in the pattern shown in figure 10. I then wrote the "Gear" routine to incorporate these inputs as program constants and to include modifications that would increase the visual contrast. The resulting pattern is shown in figure 11.

Acknowledgment

I would like to point out that the "Composite

Loop," Sinusoidal Loop No. 2" and "Gear" programs all make use of a procedure originated by a fellow computer artist and former co-worker, Woodrow Europa³. Woodrow conceived the idea of incrementing the radius by a constant for successive angular sweeps in a polar coordinate plot. He used this algorithm to generate iterated, "concentric" versions of traditional polar coordinate curves. Somewhat later, I realized that use of finite angular increments and high angular frequencies with this algorithm would make possible the generation, using various polar coordinate functions, of patterns radically different from classical polar coordinate curves. Such patterns include those displayed in figures 2 through 11.

Conclusion

I am currently exploring the possibilities inherent in the abovementioned process. A wide variety of polar functions can be formulated, and each yields a large set of possible output patterns (which result from different combinations of input parameter values). This is but one small instance of the current expansion of analytic computer art techniques into new mathematical and aesthetic domains. The near-term future should prove to be an exciting time in the development of geometric design as an art form, as more artists obtain access to high-performance computers and develop new techniques.

References

- (1) "PLOT 50: Introduction to Graphic Programming in BASIC," a computer software manual available from Tektronix, Inc.
- (2) "Computer Art for the Tektronix 4052," by J. Jacobson, Creative Computing (in publication).
- (3) W. Europa, unpublished computer program written in Hewlett-Packard "HPL" calculator language, 1978 (private communication).

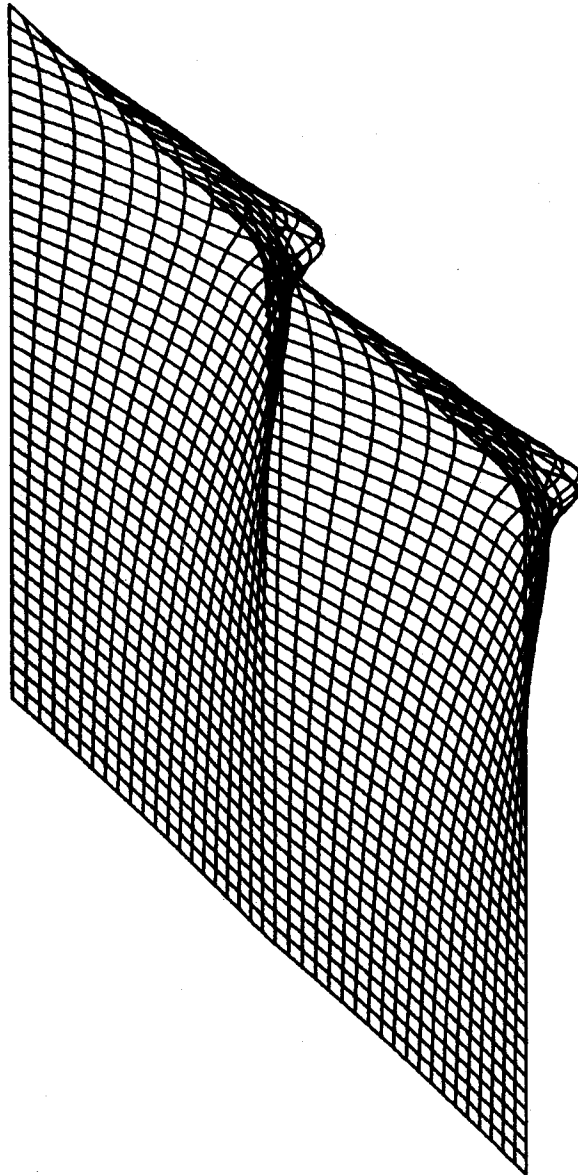
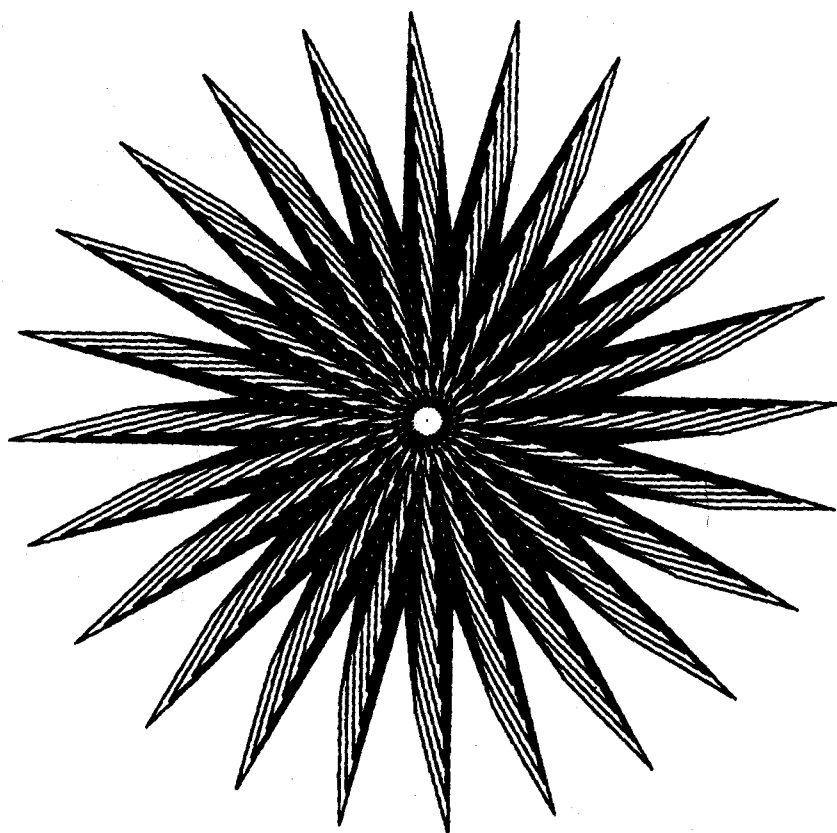
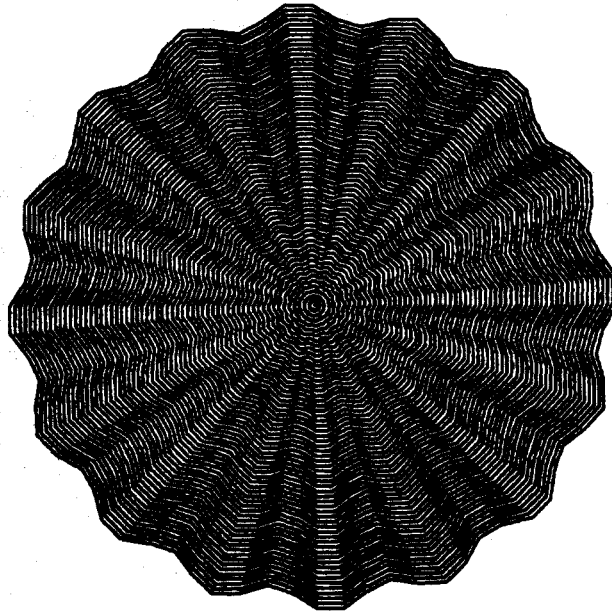


Figure 1



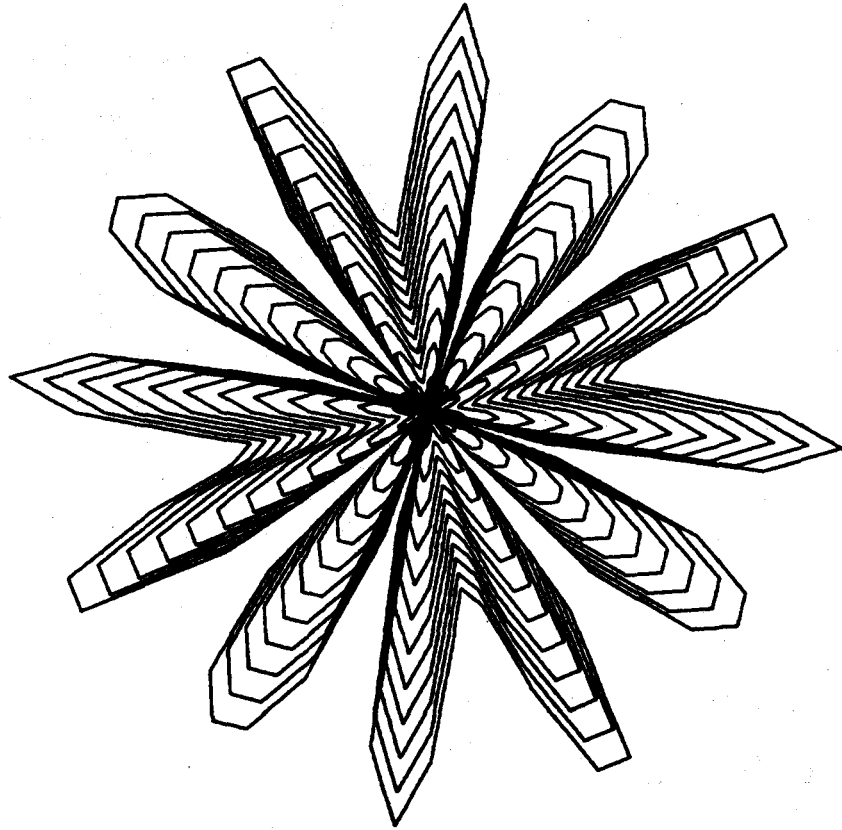
COMPOSITE LOOP PROGRAM PARAMETERS
(L,D,C,K) = (48,30,5,3.5)
(U1,U2,W1,W2) = (2,6,4,5)

Figure 2



COMPOSITE LOOP PROGRAM PARAMETERS
(L,D,C,K) = (16,6,5,1)
(U1,U2,V1,V2) = (1,1,1,1)

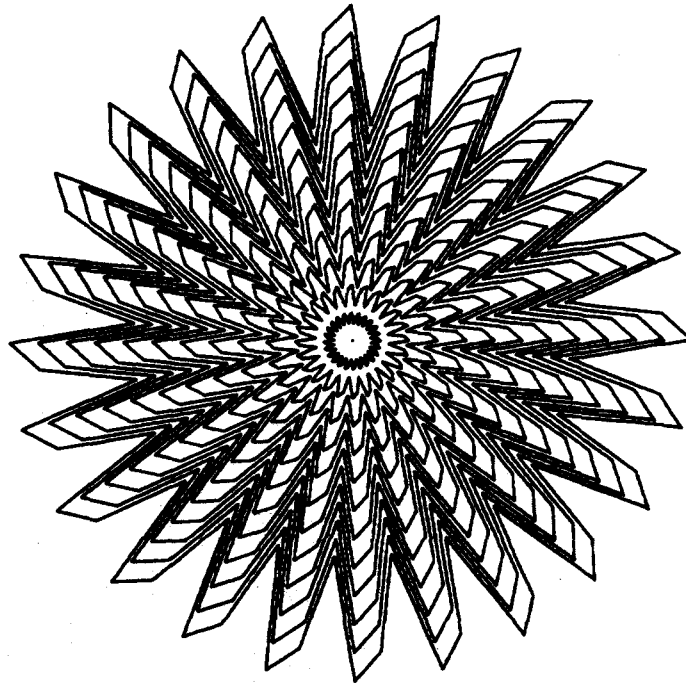
Figure 3



COMPOSITE LOOP PROGRAM PARAMETERS

(L,D,C,K) = (76,30,5,49.5)
(U1,U2,W1,W2) = (1,1,1,1)

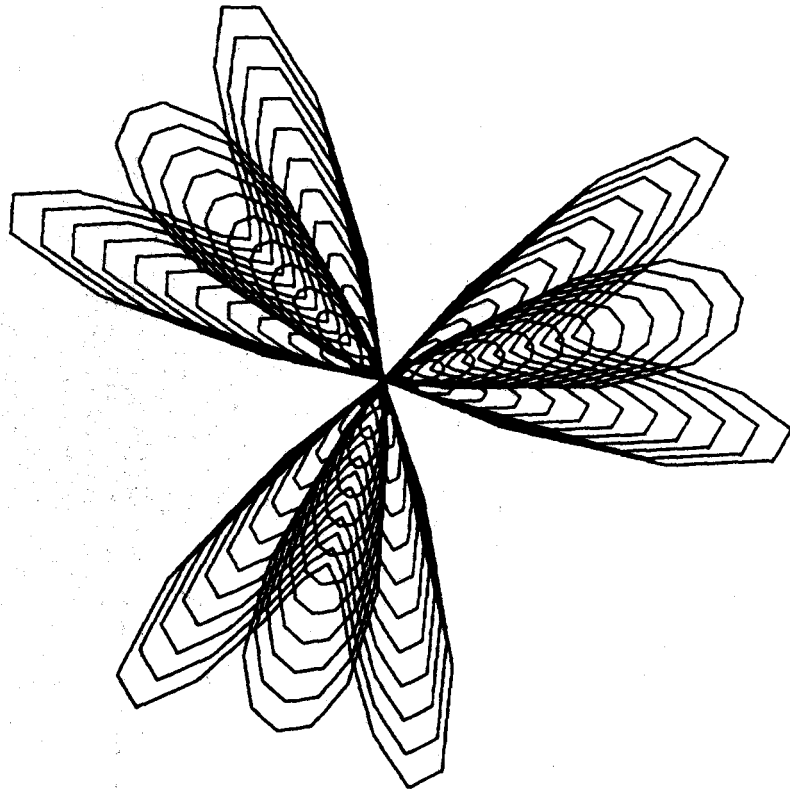
Figure 4



COMPOSITE LOOP PROGRAM PARAMETERS

(L,D,C,K) = (456,30,5,6.3)
(U1,U2,W1,W2) = (1,1,2,2)

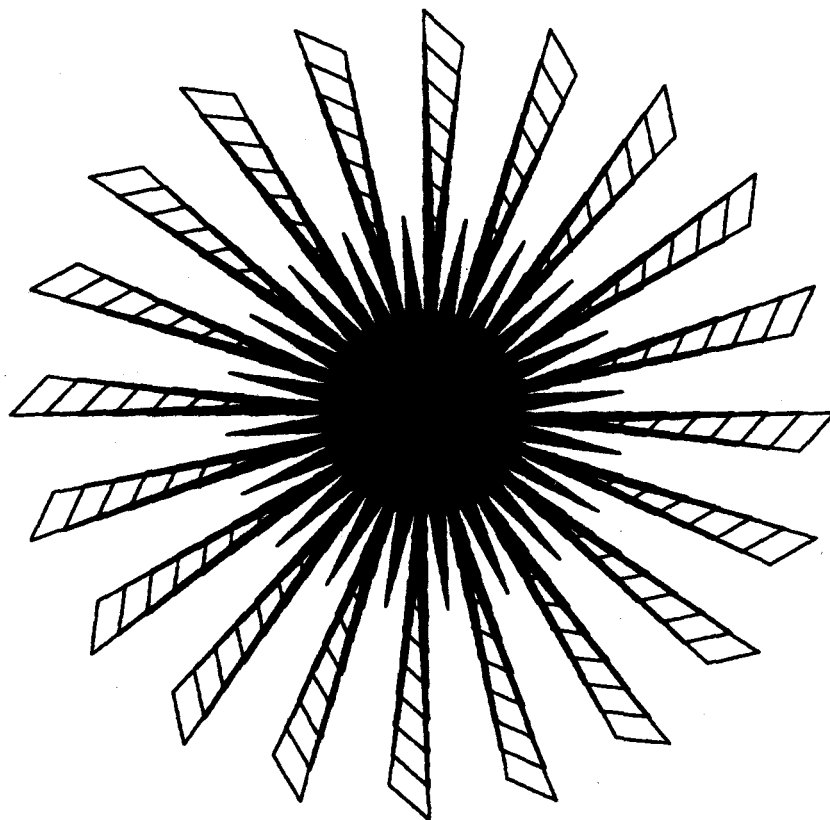
Figure 5



COMPOSITE LOOP PROGRAM PARAMETERS

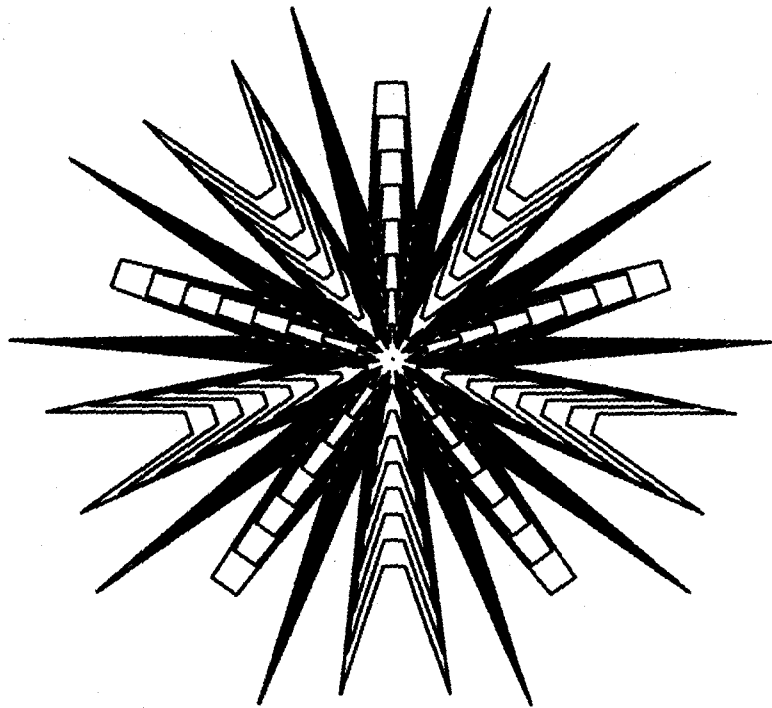
(L,D,G,K) = (75,30,5,48.3)
(U1,U2,W1,W2) = (1,1,1,1)

Figure 6



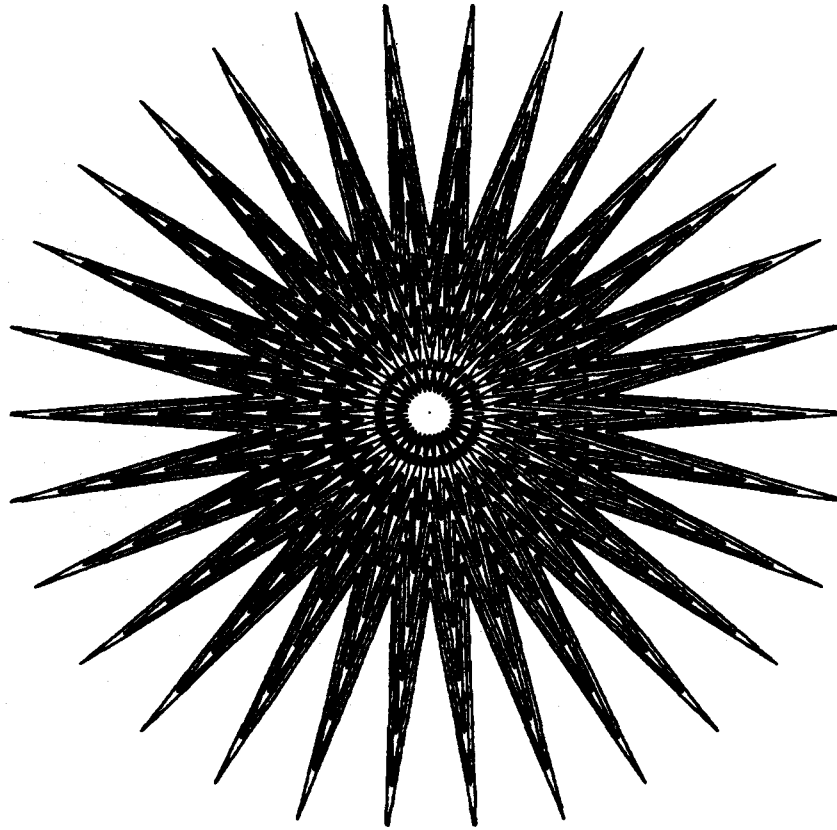
COMPOSITE LOOP PROGRAM
(L,D,G,K) = (1540,30,6,34.25)
(U1,U2,V1,V2) = (1,1,1,1)

Figure 7



COMPOSITE LOOP PROGRAM
(L,D,G,K) = (455,60,6,7,5)
(U1,U2,W1,W2) = (1,1,2,1)

Figure 8



COMPOSITE LOOP PROGRAM PARAMETERS

(L,D,G,K) = (450,75,6,7.5)
(U1,U2,W1,W2) = (1,1,2,1)

Figure 9

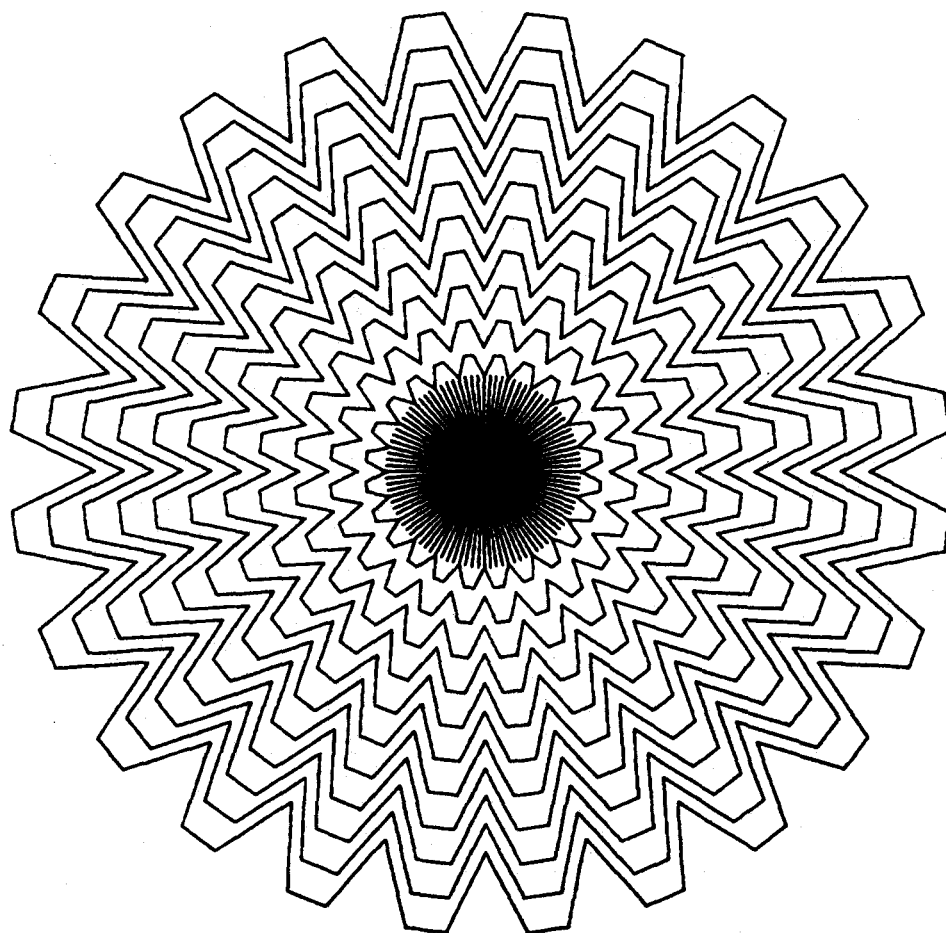


Figure 10

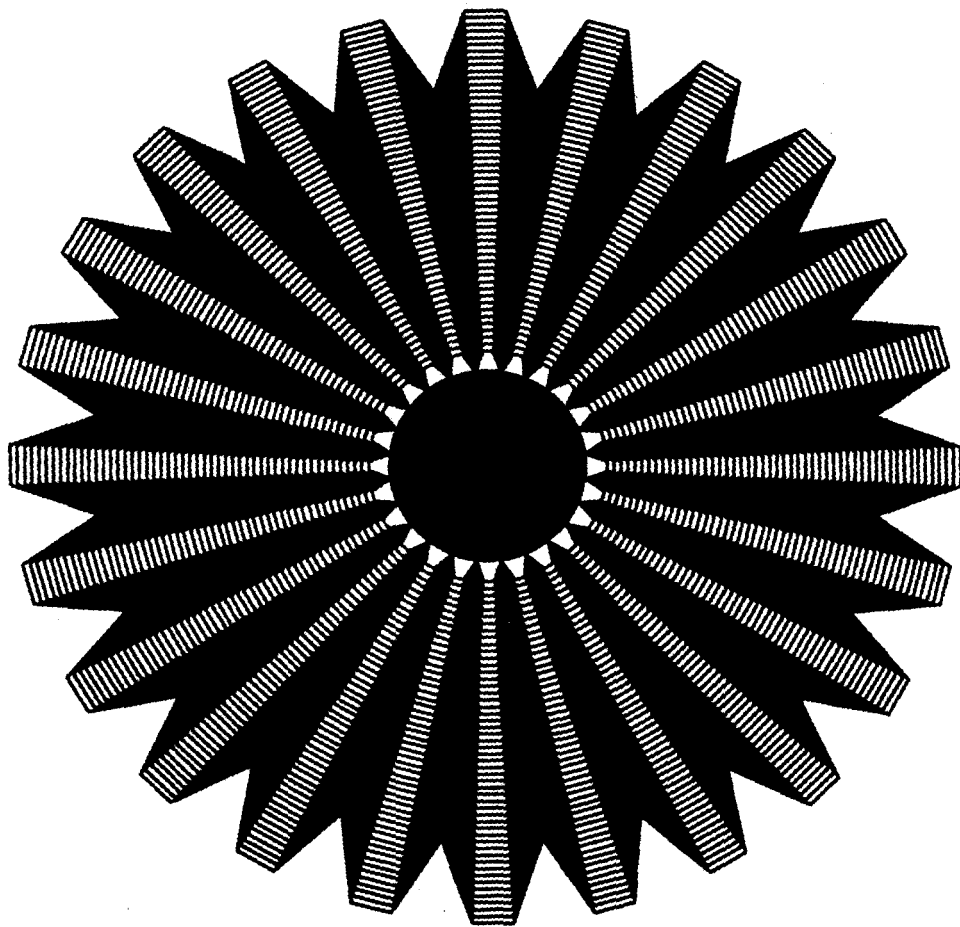


Figure 11

PANTOMATION A SYSTEM FOR POSITION TRACKING

by Tom DeWitt and Phil Edelstein

Electronic Body Arts, Inc., Albany, N.Y.

Pantomation is a tracking chroma key system. It is similar to a light pen interface for a computer but uses a color video source and chroma keyer to detect positional information. The system operates on the principle of tagging objects with colors of unique hue which can be masked out by chroma keying. Pantomation uses a computer and has an operating system which can process positional information in a variety of ways. Included are algorithms to remove spurious key occurrences, calculate velocity, record time of occurrence of tracked points, and output derived data as control voltages. As currently implemented, the system tracks up to four colors. It can be used to reposition juxtaposed images in a composite scene so that their movement is correlated. Research is underway to track the depth co-ordinate in the expectation of combining Pantomation with three dimensional computer graphics.

GLOSSARY TERMS: Blanking, Chroma Key, Key, NTSC color encoding, Raster, Re-entry Special Effects Generator, Vertical Interval, Wipe

Conception

Pantomation is a system for the corelation of scenes recorded by a camera with images created by electronic graphic synthesis. The need for such a tool became evident during productions by the authors in 1974 at the WNET TV Laboratory, a studio that housed video synthesis instruments and a re-entry switcher with chroma key. Significant production delays were encountered when foreground and background images had to be manually repositioned to provide a sense of realistic relationship between them. An example is illustrated by a sequence in which a mime creates the illusion of opening a box wipe (fig. 1). The scene had a background image of a stylized battlefield and a foreground image of the mime. The mime's movements were matched to the box wipe by manual controls on the wipe generator. Although the effect contributed to an award winning tape (1), the expense of producing such effects inhibited their further exploration.

Corelation between background and foreground images has been achieved by using computer controlled camera mounts (2). In these systems, camera movements are carefully controlled so that

separate scenes are recorded in registration for subsequent recombination as composite images. Typically, these systems impart movement to static objects such as miniature models by moving the camera. Complex scenes are created when a series of shots are combined in a motion picture optical printer.

Pantomation takes the approach of analyzing the picture itself to bring different images together in the proper positional relationship. Based on the principle of "tags" suggested by Leonard (3), small color chips are placed on objects in a scene to mark the positions of the objects. For the system to work, the color selected to identify each object must be unique to the tag. To facilitate precise color detection, a chroma keyer called the Window Keyer was designed. The Window Keyer discriminates colors precisely and can be set by computer to detect any of the tag colors being tracked. Chroma keyed video of a tagged object is converted from raster scan form to a set of digital words containing the Cartesean co-ordinates within the raster of each tag. Typical tags are small objects, and the co-ordinates describing them represent just a few



Fig. 1) A manually controlled box wipe synchronized with the movement of a mime.

points out of the video raster's quarter of a million pixels. As a result, the acquisition of data is compatible with computer processing speeds which are a fraction of the pixel to pixel data rate of the actual video. The computer has a bank of digital to analog convertors (DACs) as outputs which are patch programmed to voltage controlled audio and video synthesizers. When the system is operated the synthesizers are essentially controlled by the positions of the color tags.

System Description

Pantomation is achieved through the system illustrated in fig. 2. The head end consists of a color video source containing red, green, blue and encoded NTSC color signals. The R,G,B signals are routed to a voltage controlled window chroma keyer (fig. 3). The keyer consists of six comparators and associated logic to derive a combined key signal pulse. The "window" is created by setting limits on both the upper and lower levels of each of the three color signals. This requires a total of six reference voltages which are provided by the analog programmer. This device is a sequential voltage source typical of voltage controlled audio synthesizers. In the analog programmer, a matrix of potentiometers is preset to store reference voltages. Pulse lines from the computer set the sequencer to the preset values that define a color. In the current

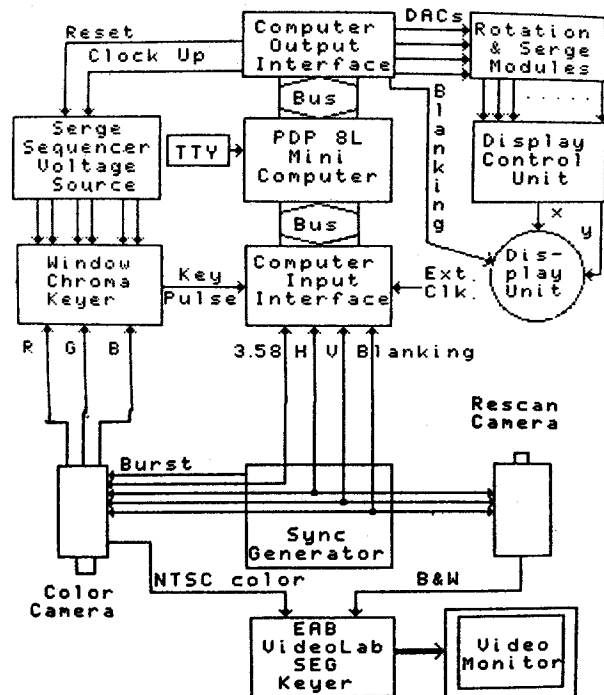


Fig. 2) Block diagram of the Pantomation system hardware

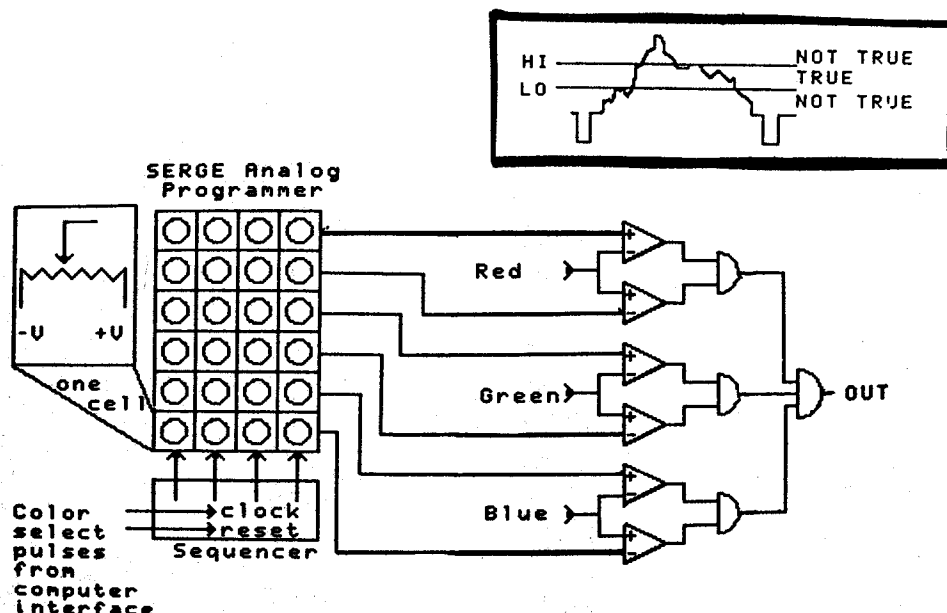


Fig. 3) Window chroma keyer. Insert in upper right shows an example of logic applied to each channel of video.

implementation of the keyer, four different colors can be tracked. In normal operation a different color is keyed each video field.

Key Positioner

The Key Positioner provides the means to extract horizontal and vertical position information from a key signal and make that information available to the computer system. This device (fig.4) resembles a light pen interface for a computer system. It reports the position of a tag in a fashion similar to a photo cell reporting the position of a light pen "hit". At the beginning of the vertical interval of video, the horizontal and vertical counters on the key positioner board are reset, and a one microsecond pulse is sent on the interrupt line of the computer bus. This event triggers a set of interrupt service routines which are described below in the description of the system software.

On the Key Positioner device, an interrupt is generated by the occurrence of the vertical interval or a key pulse. When a key pulse occurs, the current values of the horizontal and vertical counters are transferred to storage latches. In the Key Interrupt Service Routine that follows, the values in the latches are sent to the computer via the computer bus. The vertical counter is clocked by horizontal sync, and the horizontal counter is clocked by color subcarrier (3.58 MHz). The counters are reset to zero by vertical sync and horizontal sync respectively. Although eight bit counters are used, approximately 180 horizontal and 190 vertical lines are resolved due to limiting factors such as video blanking and tag size.

Central Processing Unit (CPU)

The CPU is used to provide computer software control, storage, and routing of the position information extracted from the tracked tag. The current implementation of the system is built around a PDP-8 mini-computer manufactured by Digital Equipment Corporation. The computer has an instruction cycle time of 1.5 microseconds and a memory size of 4096 twelve bit words of core. A teletype terminal is used as a console, and the computer's front panel switch registers are also used for operator communications.

Analog and Digital Input/Output (I/O)

The I/O system includes four digital to analog convertors (DACs) and three software programmable pulse outputs. The DACs are generalized in their use and can be patched to a variety of voltage controlled devices in the system. Of the programmable pulses, two are used to control the sequencer in the chroma keyer. There is a reset pulse that zeroes the sequencer at its first position and a clock pulse that steps the sequencer to the group of six preset voltages defining a particular color. This process takes no more than 200 microseconds and occurs during the vertical blanking interval. The

third programmable pulse is for display blanking. A final input device is an external clock input that synchronizes the computer to the timing of outside devices. Each cycle of the external clock triggers an interrupt service routine which transfers data to the DACs.

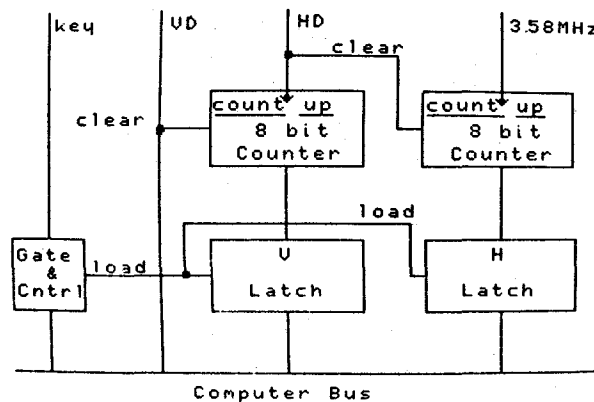


Fig. 4) Key positioner

Analog Synthesis Modules

The analog portion of the Pantomation system is used to create and modify video and audio signals. Two of the components in the block diagram in figure 2 are labelled "SERGE". These are combinations of analog devices manufactured by Serge Modular Music Systems of San Francisco, California. Pantomation includes the following modules: oscillators, amplifiers, filters, mixers, sequencers; comparators. These modules are voltage controlled and patch programmable. They are designed for synthesis and processing of waveforms in the audio frequency spectrum and are comparable to Moog, Buchla, and other audio synthesizer components (4). A VideoLab special effects generator designed by Bill Hearn of Electronic Associates of Berkeley, Ca. is also used in Pantomation. The VideoLab is used for processing functions such as re-entry keying, colorization, wipe and cursor synthesis. (5)

Output Subsystem

The output display system is based on a Rutt/Etra CRT and control unit manufactured by Rutt Electrophysics of New York City. The display controller has manual and voltage controlled inputs for altering the height, width, depth, rotation, intensity, and position of a video image. The display itself is a black and white CRT and is rescanned by a black and white television camera. The display yokes can deflect the CRT beam the entire screen width at frequencies up to 50 KHz. The CRT beam can be blanked at video bandwidth. The voltage control functions operate at approximately the same bandwidth as the SERGE modules, hence the selection of audio synthesis technology in a video synthesis application.

There are two different modes of operation for the output display: Raster and Stroke. A raster display consists of a television image taken from a composite video source. This image is displayed on the Rutt/Etra CRT in raster format, but is repositioned according to the position of the tracked tag. The rescanned version of the image is then available for insertion in a composite image through the re-entry special effects generator. In Stroke operation the display is used much like an oscilloscope to produce synthesized images. These are images created by movement of the CRT beam. Unlike raster based images which depend on changes in light amplitude, stroke displays can produce images with no change in beam brightness but simply with changes in beam position. In Pantomation, the only brightness change used during stroke display is a blanking pulse taken from the third programmable I/O pulse. It is used to turn off the display when it is not being refreshed. Stroke displays are used in Pantomation when the repositioned image can be synthesized from vector lists.

Software Operating System.

Much of Pantomation exists as computer programs. The Pantomation Operating System (POS) performs initialization during startup and services interrupts originating from the hardware interface. Other areas of memory are used to store positional information, system variables, and a program called Octal Debugging Technique (ODT).

Initialization Routine

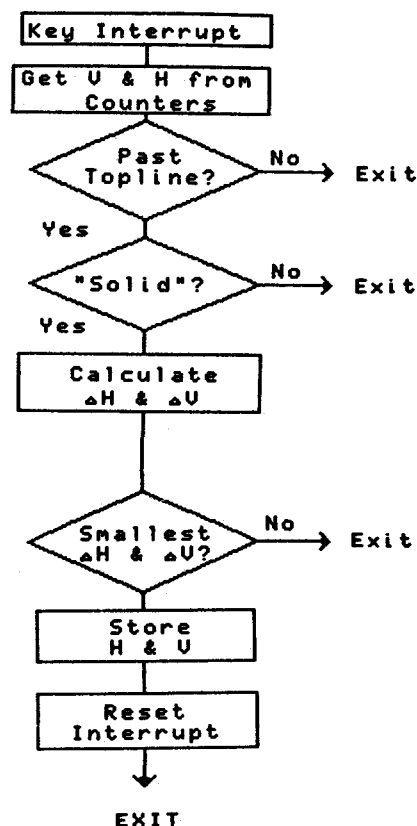
When POS first begins, the initialization program resets memory locations used for system variables, normals the chroma keyer to the first color in the sequence, enables all interrupt flags, and enters a wait state until an interrupt occurs. When an interrupt is asserted, the hardware interface is polled to determine which interrupt has occurred, and the appropriate interrupt service routine is entered. The program returns to the wait loop in the initialization routine after all requested interrupts are serviced.

Keyer Interrupt Service Routine

A flow chart of the key interrupt service routine appears in fig. 5. When the computer senses a key interrupt pulse on its bus, the most recent values stored from the horizontal and vertical counters on the Key Positioner are transferred to the CPU. If the vertical position is less than a variable called TOPLINE, the key interrupt routine aborts and the computer returns to its wait state. The TOPLINE software function was implemented to eliminate accidental or spurious key interrupts that occurred at the top of the frame from overhanging studio lights. If a key occurs below the TOPLINE variable, the horizontal and vertical position is stored in a

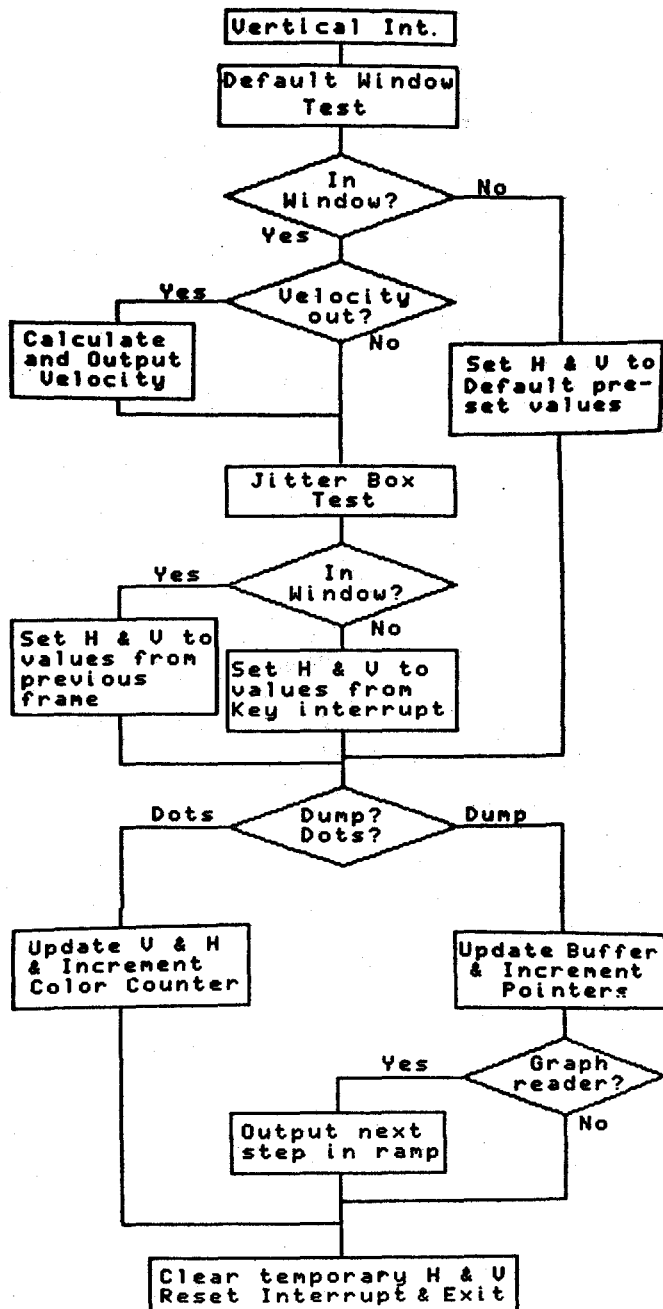
pair of words in memory.

If a second key occurs in the same video field, it is tested to determine how close it is vertically to the previous key in that field. This test, called SOLID, compares the vertical offset between keys by subtracting their vertical positions and comparing the results to a constant. This value is initialized as 7 vertical lines but can be changed by the operator. If the key has occurred no more than the tested number of lines from the previous key, it is assumed to be part of the same "solid" object. The purpose of separating "solid" keys from isolated key interrupts is to eliminate occasional key pulses that are artifacts of the video system such as color camera mis-alignments or signal noise. If the key pulses test as solid, the computer stores the vertical value of the first key occurrence and the horizontal value of the second key. This format of determining the position of the key source was found to be necessary because of large amounts of dither in the horizontal position of the first line of a keyed area.



Flowchart for key interrupt

Fig. 5



Flowchart for Vertical Interrupt

Fig. 6

The key interrupt routine also has a provision to reject spurious "solid" key sources as might occur if a color in the video is identical to a tag color. As keys occur during the field, their position is compared to the position of the tag during the last field during which that tag was sampled. The change in position between the samples is stored as the difference between current and past horizontal and vertical positions. If more than one "solid" key is detected within a frame, the key which was closest to the position of the previous key is accepted, and key points occurring elsewhere in the frame are ignored. Pantomation has been able to separately track two objects of identical color provided they did not touch. Running on the PDP-8, the key interrupt service routine is performed within 200 microseconds, allowing a new key to be tested every four video lines.

Vertical Interrupt Service Routine

At the beginning of the vertical interval, an interrupt triggers a service routine (fig. 6) which further processes the key position information obtained during the previous field. Two tests are run, both similar to TOPLINE. The first test, designated DEFAULT WINDOW, tests the position of the key to determine if it is within boundaries at the borders of the frame. These boundaries are TOP, BOTTOM, LEFT, and RIGHT (fig. 7). Their values are stored on page zero of POS.

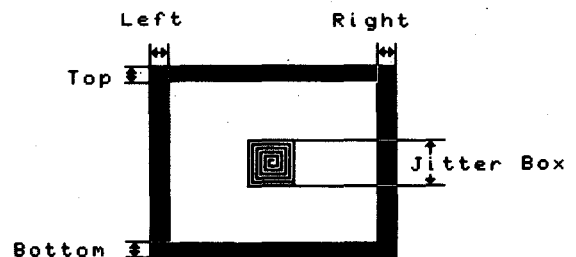


Fig. 7

If the key is in the DEFAULT WINDOW, the positional information from the actual tag is replaced by preset horizontal and vertical values, typically 0,0. The DEFAULT WINDOW is useful for creating the illusion that a tag carried to edge of the camera field has been carried out of view of the camera. Another test made during the vertical interrupt routine is called JITTER BOX. If the position of the latest sampled key position is within a pre-assigned distance from the position of the previously acquired key, the latest key position is ignored, and the position of the previous key is used. This inhibits occasional dither in values coming from the key positioner. The effect of this technique is observed when a tag is virtually stationary. As soon as the tag moves outside the JITTER BOX the full 8 bit resolution of the key positioner is used.

After the value of the position of the tag has been established, the information can be deposited in several locations in memory. In VECTOR DUMP mode, the data is stored as one entry in a software implemented First-In-First-Out (FIFO) buffer. This buffer can be likened to a software function generator that produces the position versus time function for a selected tag. This feature is useful for creating patterns from the path of movement of the tag (fig. 8). A special use of VECTOR DUMP is reading a formatted graph from a video camera into the FIFO buffer. This technique requires that the computer generate an analogue ramp wave from one of its DACs to move a voltage controlled cursor. As the cursor increments, it acquires the next value of a graph scanned by a video camera. The technique allows the operator to load handdrawn waveforms into the software function generator.

In DOTS mode, the position of the tracked tag is loaded into a pair of registers associated with the color being tracked. The number of colors corresponds to the preset values on the analog programmer and is set by the operator for each production situation. The POS software has been written to keep track of up to eight colors. Finally, the vertical interrupt service routine calculates the velocity of the tracked tag. This calculation averages the change in position of the tag over several fields to allow for the measurement of the velocity of slowly moving objects.

External Clock Service Routine

Most output functions of the computer are associated with the External Clock Interrupt Service Routine. This program is initiated whenever a pulse is sent to a patch programmable input on the I/O interface. Typically the pulse comes from an oscillator. Each pulse triggers an output cycle consisting of a transfer of data from the computer memory to the DACs. In DOTS mode DAC assignment for the routing of the data is determined by the operator before startup and establishes which tag positions will appear at which DACs. The VECTOR DUMP FIFO buffers are preassigned, one to each DAC, and they are enabled or disabled by computer console faceplate toggle switches.

Applications

Conventionally, a chroma key opens an area of a picture for insertion of another image. Such effects allow combinations of images by dividing a picture into foreground and background planes. Ordinarily, the background is keyed out of an image by placing the camera subject in a setting of unique and uniform monochromatic color such as blue or green. Pantomation alters an assumption of chroma key technique in that the tags it follows by chroma key masking are primarily foreground objects. Unlike the large areas of monochromatic background in the typical chroma



Fig. 8) "Drawing with Cheerios," a mime traces a space using a colored ball. 128 sequential positions are shown in Vector Dump mode.

key studio, these foreground objects are small. Conceptually they represent the center points of images that will be inserted over them. The foregrounds that will be inserted are not the same size or shape as the tags. The tags may never be seen in the final composite picture, because they are in the hidden portion of the background plane.

Pantomation, as currently implemented, can track up to four colors. There are conventional chroma key studios that use multiple chroma keys. As more than two images are combined, foreground and background distinctions must be extended to a series of planes that exist in a conceptual third dimension. Even in simple two plane chroma keys, depth information is important if the camera moves. If a pan over the foreground moves the foreground figure, the background being inserted must also move and at a rate which is proportional to its assumed distance from the foreground. Pantomation can be used to automatically move a background in synchronization with camera movement on the foreground object. As the position of a tag changes, the position of the second plane in the image is repositioned by a proportional amount. This technique does not actually determine the true depth of the tracked object.

Depth Detection

Some commercially available equipment can detect the size of a chroma key object (6). Size is proportional to the distance of the object from the camera, so this facility could be used to calculate the depth coordinate. In Pantomation this measurement is accomplished by placing tags at the edges of the object being tracked. The distance between the tags, calculated by subtracting the left tag position from the right

tag position, is proportional to the distance of the object from the camera. Another method for determining the depth plane position of tracked objects is to use more than one camera to detect position (fig. 9). A scene recording camera looks at the subject front-on, and another camera is placed at an orthogonal angle to the subject. The horizontal axis of the image in the orthogonal camera corresponds to the depth axis of the front-on camera. With multiple cameras set at orthogonal angles to a subject, Pantomation can derive the horizontal, vertical and depth co-ordinates of tracked tags. Depth detection would be useful in applying computer graphics to television production. Computers can manipulate images in three dimensions. Television systems interfaced to computer graphics systems will need methods for addressing three dimensional co-ordinate space.

Animation

One approach contemplated by the authors for television controlled computer graphics is based on the art of marionettes. A puppet is a three dimensional object which is manufactured a single time for subsequent manipulation in a performance environment. If a three dimensional model of an object is stored in a computer, it can be displayed in a wide variety of translational, rotational, and scaled configurations. Creating such models forms the analogy to making the puppet.

There are many methods being explored for building mathematical models of objects. Pantomation can be used in at least two approaches to the problem. Artists normally draw on a two dimensional surface such as paper. The Pantograph would allow an artist to draw "on air" in a three dimensional space. In this application, the artist handles a baton with a color tagged tip. If the baton tip is tracked in three dimensions, its spatial co-ordinates can be used to construct a computer graphic. An artist watching on a stereo-pair crt system (7) can draw a three dimensional object. Another approach, being developed conceptually, analyzes a real object for its three dimension co-ordinates. The system would record the positions at which an object is struck by a point of light from a laser beam which is deflected over the object. In Pantomation, the "hit" of the laser light would be used to tag the surface, allowing acquisition of all the points that intercept the light.

After a computer "puppet" is created, its movements can be imparted by Pantomation techniques. The puppet is partitioned into volumes that will be correlated to tags in a scene recorded by television cameras. In the analogy to marionettes, the tags correspond to marionette strings. During production, repositioning of the tags by the movement of actors or cameras will control the movements of the partitioned volumes. The proposed computer puppet process would benefit animation, because a character would only need a single rendering. The process would allow for much greater detail in designing the

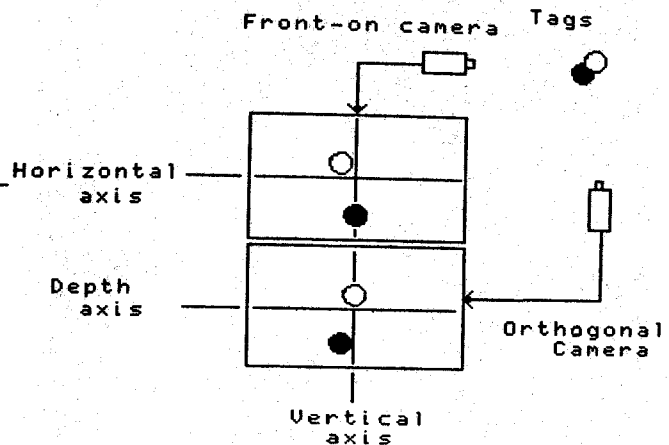


Fig. 9) Diagram of the set-up for three dimensional tracking

character than is permitted when many multiple drawings must be made.

In an experiment conducted with Pantomation, lip movements of an animated face were made by tracking the lips of the narrator who wore a color tag on his chin. This technique could free animators from the time consuming task of correlating lip movements with spoken dialogue. Another experiment with Pantomation created an animated wing for a dancer giving his interpretation of flight. Three points were tracked: the hand, the elbow, and the shoulder. This allowed for a realistic articulation of a synthetic wing made by video synthesis (fig. 10). The experiments are leading toward production techniques that divide control of an animated character among several artists such as a draughtsman for the image, a dancer for the body movement, and a orator for the voice.

Performing Arts

Other experiments with Pantomation have focussed on combining audio and video synthesis with music, dance and mime. A musical performance has been conducted by a bassist who used a tagged hand to control the pitch and rhythm of an audio synthesizer. He appeared to be playing an "invisible" instrument, but his videotaped image had the synthesized image of a vibrating bass string superimposed. The Pantomation system has been used to read a handwritten musical score. This technique uses a variant of the VECTOR DUMP program. The score is prepared according to the format of skyline notation, a form of bar graph, and each voice is written in a different chroma key color ink. The graphs are made on paper with punched registration holes, and are scanned by a video camera on an graphics stand. The graph is stored as a list of values in the VECTOR DUMP buffers and is output later through DACs under



Fig. 10) A corporeal mime articulates a synthetic "wing."

timing control of the external clock. The technique has the advantage that a composer can prepare an electronic music score away from the studio using the traditional approach of handwriting at the piano.

Pantomation was developed within a modern dance company, Electronic Body Arts, and has had immediate applications in videotaping dance and mime. As a tool for training and rehearsal, the system allows the teacher and choreographer to study the isolated tagged parts of the dancer. Audio cues can be generated by tracking position, giving the dancer an indication of important variables such as location on the stage or relationship to other dancers. Velocity detection is of use when comparing the current performance to performance standards. Notation for dance can be accomplished with videotaping, but some of the abstract ideas in dance can be understood by limiting the notation to key components of body movement. Laban notation has been implemented on computer graphics systems for this purpose (8). Pantomation dance notation has the advantage of simplicity, both in the recording and the reconstruction of dance movement. The system has produced videotapes in which graphic abstraction itself is combined with the dance (fig. 11) (9).



Fig. 11) A dancer holding two tags of different color manipulates pompoms created by electronic image synthesis.

SUMMARY

A tracking chroma key process called Pantomation has been developed. The system incorporates a mini-computer and operates on the principle of color tagging. Positional information about subjects or scenes is derived from computer analysis of color coded information taken from images scanned by television cameras. These positions are marked in the scene being televised by placing color chips called tags on the points to be tracked. The tags are detected by chroma keying techniques, and a special interface to the computer detects their position

in the raster. The computer processes, stores and outputs the positional data under software control. The output of the computer is connected to production components such as video and audio synthesizers. Pantomation is similar in construction to light pen interfaces common to computer graphics, but a camera is used to sense light instead of a single photo cell. The system is interactive and designed for real time operations.

Pantomation was developed by a theatrical company that produces works in dance, mime and music and is intended to advance the integration of these arts with film and television. Some of the applications for the system are in matching movement of chroma key backgrounds with foregrounds, tracking and displaying trajectories of objects in motion, and controlling electronic graphics and sounds. Uses under investigation include three dimensional tracking for interfacing video to computer graphics systems.

ACKNOWLEDGEMENTS

Pantomation was developed by a team of artists and engineers. Analog electronics were designed by George Kindler and the operating system software was written by Roger Meyers. The display and special effects generator came by way of the efforts of Bill Etra. Video artist, Vibeke Sorensen, and dancer, Maude Baum, made production contributions. The authors are indebted to John Godfrey of the WNET TV Laboratory for his foresighted endorsement of the project, and to Aaron Heller, Dean Winkler, Dave Powell, and the people of WRPI radio who gave Pantomation a home. The apparatus was constructed on a grant from the New York State Council on the Arts to WMHT TV and took place at the Electronic Music Studio of the State University of New York at Albany. Research, conceptualization and exercise of Pantomation was supported by grants from the Media Arts program of the National Endowment for the Arts, the John Simon Guggenheim Memorial Foundation, and the Creative Artists Public Service Program (CAPS). Pantomation is a project of Electronic Body Arts, Inc. of Albany, New York and the system resides at the Image Processing Lab of Rensselaer Polytechnic Institute.

(1) Tom DeWitt, "Zierot in 'War Walls'", CATHODE RAY THEATER, distributed by Electronic Arts Intermix, New York City

(2) John Dykstra, "Miniature and Mechanical Special Effects for Star Wars", The American Cinematographer, V. 58, July '76, p.702; Joseph L. Matza and John Crale, "Magicam - The Process and Production Techniques", SMPTE Journal, V. 86, Oct. '77, p. 728; Pennfield Jensen, "Battlestar Galactica", Filmmakers Newsletter, V. 11, #12, Oct. '78, p. 20

(3) Eugene Leonard, "Considerations Regarding the Use of Digital Data to Generate Video Backgrounds", SMPTE Journal, V. 87, Aug. '78

(4) Allen Strange, ELECTRONIC MUSIC, Wm. C. Brown Co., 1972; Thomas Wells, THE TECHNIQUE OF ELECTRONIC MUSIC, Schirmer Books, 1981

(5) Peter Caranicas, "Video Hardware Gets Smart", Videography Magazine, V. 1, #3, June '76. p. 28

(6) Michael D. Patten, "The Digital Video Effects (DVE) System", SMPTE Journal, V. 87, April '78, p. 217

(7) J.N. England, "A System for Interactive Modeling of Physical Curved Surface Objects", ACM Siggraph 1978 Proceedings, p. 336

(8) Norman I. Badler and Stephen W. Smoliar, "Digital Representations of Human Movement", ACM Computing Surveys, Vol 11, No. 1, March '79, p 29

(9) Tom DeWitt, "Pantomation", SIGGRAPH Video Review, Issue #2, section 9, Sept. '81, distributed by UICC/NFE, Box 4348, Chicago, IL.

GLOSSARY

Blanking: The period during the video signal when the crt beam is turned off such as during the vertical interval.

Chroma Key: A key created by analyzing a color video signal for red, green, and blue components.

Key: A comparison of the video signal to a reference brightness level. A keyer creates a bit mask based on the outcome of the brightness test.

NTSC color encoding: A method for transmission of color information using a 3.58 Mhz phase modulated sine wave which is added to the brightness level of a black and white television signal.

Raster: The Cartesian grid of a television picture, for example, North American television has a raster of 525 lines vertically, each with several hundred pixels horizontally.

Re-entry special effects generator: A television production component that combines several signals by keying, wiping and mixing. Sometimes called a "switcher."

Vertical Interval: A synchronizing period which occurs between fields of brightness information. In North American television the vertical interval is about one millisecond and occurs each 1/60th of a second.

Wipe: A video special effect which produces a simple geometric mask such as a box.

REAL TIME ANIMATION TECHNIQUES WITH MICROCOMPUTERS

Frank Dietrich

Pixel Creations
731 West 18th Street
Chicago, Illinois 60616

Animation movies are back in business. For a long time this genre was almost identical with Walt Disney's cutely drawn animals and fairy tale characters. Now it is the Disney Studios again making the breakthrough into the animated space age of computers with a new creation TRON. The computer generated special effects of many other current films make headlines in the movie critiques. The type of computer animation possible with low-cost machines with less resolution, less colors, less speed, ... the list of deficiencies could easily be extended, are a far cry from these expensive and very sophisticated visual innovations. Nevertheless, as will be demonstrated throughout this article, the small graphics computers have strong features for animation used by industrial and educational video productions as well as in the exploding world of cable TV.

TRADITIONAL & COMPUTER ANIMATION

Animation is the art of changing images in time. This is faster said than done. Traditional film animation requires many tedious and time consuming steps to produce the numerous drawings necessary for sophisticated motion effects. Computer animation can save labor by automating some of these tasks like cel inking or inbetweening. Computer Assisted Imagery (CAI) can even produce complete movies. But most computers do not produce real-time animation. Instead they generate one still image at a time. The illusion of motion is invoked later, as in traditional animation, when complete series of stills have been filmed and are projected at 24 frames per second.

Real-time animation has been possible in the past only with vector refresh displays, mainly because the computer could calculate the new endpoints of relatively few vectors forming the image during the refresh cycle of the display device. Much less picture information has to be processed in such vector systems

than in a raster system, where the entire screen memory containing up to millions of pixels (picture elements) has to be rescanned for display. Changing parts of this amount of such large memory has been until now too slow to allow for real-time animation. Only modern high speed computers with dedicated image processors can accomplish this task in the short time of a 30th of a second.

In this respect it is a surprise that microprocessor-based video game machines are highly interactive and capable of real-time animation. Their main limitation, low resolution, becomes a feature: less pictorial information has to be moved around. Even if this motion appears to be somewhat crude, it is nevertheless real-time animation and serves the interactive purpose.

The animation techniques discussed here were executed on a low-cost micro-graphics system, the Datamax UV-1, initially developed from a video arcade game computer. Its RAM was expanded to 32K, and 256K of screen memory were added, yielding a 320*200 pixel resolution with 16 2-bit 'thin' framebuffers. ZGRASS is the hi-level graphics programming language of the system, ready to go in 32 K of ROM. An internally generated standard NTSC signal puts the imagery from the computer right into the heart of the video world. Even though all this accounts for a specially configured micrographics system, many of the animation techniques are commonly used and thus serve as general examples for getting a maximum of motion out of small computers.

ADDITIVE CHANGES

Two different ways exist to change an image, one, by drawing new graphics (or erasing existing ones for that matter), the other by changing previously created and stored images. Of course the later which can be either a change of color or a

switch to another image plane happens much faster and therefore is better suited for real-time animation. In terms of how much information is changed by drawing, the least demanding animation technique simply keeps writing one graphic element over the previous ones, thus building up the image. This amounts to a gradual change of the image, where the order of introduction of the picture objects significantly contributes to the meaning produced. In some respect this is like story telling with pictures as an ongoing process. And, as in stories, one new element can drastically turn the already known facts around.

There are two flaws with additive changes. This technique can only produce gradual changes of an image and the viewer has to watch the drawing of the new graphic, which can divert the attention to a very insignificant action like filling in an outlined shape. Computer systems featuring at least two image planes offer help: With a technique called "Double Buffering" the immediate action of drawing can be hidden by always showing the image in the other buffer, where no drawing takes place. Once the drawing is finished the program switches to the plane displaying the new image and continues to draw onto the plane which is now not shown. In this manner the animation effect is concentrated on the images changing in time without having to show how the images are being drawn.

```
SHOW plane1; DRAW to plane2
SHOW plane2; DRAW to plane1
```

COLORMAP ANIMATION

The fastest method of changing a raster image is known as colormap animation. Since no drawing is done, the bitmap, storing the digital information of the screen, is not touched at all. Instead, only the representation of color for each given combination of numbers is altered. Since this is a minimal computation it can be quickly accomplished. Just consider the difference between changing only four color values to calculating up to 320*200 pixels. A simple but effective trick like the blinking of a particular object is done by switching back and forth between its color and the background color. Continuous motion (e.g. bloodstream, waterfall) can rather easily be visualized, if a series of colors is switched.

```
TEMPCOLOR = COLOR 1; COLOR1=COLOR2;
COLOR2=COLORn; COLORn=TEMPCOLOR
```

A more elaborate version of blinking is the creation of multiple frames in one image plane. Essentially it is a trade-off between color resolution against multiple images which are initially hidden in the background and then consecutively switched on to create the illusion of motion.

```
COLOR1...COLORn=BACKGROUND
COLOR1=FOREGROUND;COLOR1=BACKGROUND
...
COLORn=FOREGROUND;COLORn=BACKGROUND
```

SNAP ANIMATION

Numerous animation techniques are possible given the ability to store parts of the screen in a special image array. The DATAMAX UV-1 features a custom designed chip to handle the activity of storing and displaying images up to a quarter of the total size of the screen. Such a memorized image is called a SNAPSHOT or SNAP for short. The same effect can be done with other systems by storing images into another frame buffer for instantaneous retrieval.

The simplest method of animation is to make a SNAP and then continuously display this image along a computed path.

```
SNAP name,xcenter,ycenter,
width,height
1MOVE DISPLAY name, xcenter, ycenter,
displayoption
xcenter=xcenter+offset;
ycenter=ycenter+offset;
IF Xcenter AND ycenter<
screenborder,GOTO 1MOVE
```

The display option defines how the SNAP information is logically combined with the existing values in that particular part of the screenmemory. A total of 150 different display options is available: PLOP, OR, XOR, AND, etc. are the choices for the BOOLEAN operation to be performed. Additional color filters can be applied to determine which colors would be effective. For instance, one useful application of appropriate display options would be to let a red car drive behind a green house, disappear and become visible again.

A special trick called the "Difference Snap" relies exclusively on these display options. The difference snap deals with the problem of having to execute two drawing commands to move an object once: first the image has to be erased at the old location and then it must be displayed at the new location. Not only does this take more time, it also causes a disturbing flashing visually interrupting the movement.

The "Difference Snap" combines erasing and displaying into one single action. Its name describes what it is: the visual difference between a snap at location X1 and X2, created by XORing the snap onto itself while it was offset by exactly the amount of pixels it should move. The visual result becomes the difference snap and is stored. Now the movement can be achieved by displaying the original image only once and then continuing to XOR the difference snap with the pre-defined offset. Memory considerations restrict this technique, since two snaps are needed and the direction of movement is not variable.

SEQUENCING

By now we know how to move static images across the screen. In order to change the image itself the snaps have to be sequenced. This technique consists of pre-storing and playing back a series of snaps. First a number of snaps are created, each slightly different than the previous one. These images are labeled in series PIC1, PIC2, ..., PICn. Again, memory constraints allow only a limited number of snaps of a limited size to be stored. The animation takes place as a simple sequencing through the snaps much like flipping through a flipbook. In this case the PLOP display option is used, in order to completely cover the old image information with the next one.

```
DISPLAY PIC1, x,y,PLOP
DISPLAY PIC2, x,y,PLOP
...
DISPLAY PICn, x,y,PLOP
```

The particular effect achieved depends on the chosen approach. One approach is a sequence in the tradition of Disney animation where two keyframes (or extremes) for each cartoon character are drawn first. Then a number of inbetweens are drawn to create a smooth transistion from one extreme to the next.

A pseudo rotation around either the X or Y axis can easily be created by scaling down either the width or the height of the snap until only a thin line remains. Along this scale subsequent new snaps are made and later played back in sequence. The net result of this technique is a rotation without any time consuming computation of the sine and cosine functions which are usually employed.

MULTI-PLANE ANIMATION

Similarly, the 16 planes of the UV-1 can be utilized for full screen animation

by cycling though a series of previously designed images. This is an important extension of displaying graphic objects in snaps both in terms of the size of the image changed and the decrease in time this takes. The switching from one plane to the next (memory bank switching) is so fast, that the eye can not detect the switch. In addition to multi-plane animation, the UV-1 hardware has two other important features: video-digitizing as a source of input and programmable arrangements of 16 screens either into a panorama of X by Y planes or into a stack of 16 planes, one behind the other.

INTERPOLATION

Since the very beginning of computer animation various mathematical forms of interpolation were established mainly to produce keyframe animation of cartoon characters. Basically the interpolation algorithms are set up to divide a range formed by two given numerical poles into a specified number of intervals, thus producing intermediate values in between. The distribution of the intervals and the numbers of interpolated values depends on the function (e.g. linear, sinusoidal, etc.) and the step factor used. Both determine the number of intervals and whether they are constant (linear) or not (sinusoidal). Interpolation algorithms as such are so general that they can be applied to almost any element of the animation sequence: distance, size, angle, time, etc.

Interpolation techniques played a prominent role in a recent computer installation entitled DO-IT YOURSELF MACHINE ART by Joanne Culver, Zsuzsa Molnar and myself. The DATAMAX UV-1 was programmed in such a way that two participants from the audience could create instant animation. Each of them drew one keyframe into plane 1 and plane 16 respectively. Then the computer took over, interpolating the in betweens necessary to transform the picture drawn by the first player into the picture by the second one. After finishing the drawing job the computer cycled through the planes creating a short animation sequence.

FIELD MOVES

Most animation techniques discussed so far share one commonality which can easily turn into a major problem. They require memory space set aside for the storage of images. Fortunately another technique exists that does not need one single bit more image memory to move fields as large as the entire screen.

ZGRASS provides two graphic commands facilitating this welcome option, SCROLL AND WRAP. Both enable the move of an entire section of the bitmap to a different location (memory block move).

Both commands require the specification of X and Y location, width and height of the field, X and Y direction of the move and finally the display option. The main difference between the two is that SCROLL simply moves the designated field to a new location, automatically clipping it along the screen boundaries if necessary. WRAP performs a wrap-around function to the image information contained within the area, but displays it again at the same location.

Both techniques lend themselves very well to preparing animation sequences for later playback as well as copying large areas. But the drawing of these techniques is such a visible interruption, that it should be hidden by double buffering (as described earlier) to successfully utilize these field moves. Sometimes an artifact of WRAP can be turned into a visually interesting result. Since WRAP simultaneously executes both READ and WRITE functions, the image that appears during the WRAP as seen as though through venetian blinds in motion.

TIMING AND CONTROL

Finally the question of timing and rhythm must be addressed, since a high degree of control over time is absolutely necessary for animation. Unfortunately the option of greater speed is limited. Only two possibilities presently exist: to compile a program or to streamline it. The latter can be done, for instance by calculating lookup tables of moves defined by complex functions thus stripping down the computation during the animation to the bare bones of retrieving those values.

Slowing down the animation is much easier and offers more options. A WAIT command counts in seconds helpfully slowing down the action. System timers which need to be set only once and then count down to zero become helpful for triggering a new event after completion of a previous one.

For choreographing a number of simultaneous events parallel processing comes in handy. ZGRASS offers three different execution modes: normal, foreground and background. The normal mode executes a program only once and then returns. Background and foreground modes execute the program until explicitly stopped. The difference between them is

that programs running in background mode are executed one line of code at a time whereas macros in foreground mode are executing a line of code each 60th of a second.

Features like this make use of the extendability of the graphics language to allow the development of libraries of special purpose routines. They also support the task of combining a handful of different events into one complex animation. A controller program for such a sequence could look like this:

```
READJOYSTICK.FOREGROUND  
MOVESHIP.FOREGROUND  
CHANGECOLORS.BACKGROUND  
MAKESOUND.BACKGROUND  
DISPLAYSCORE. BACKGROUND
```

SUMMARY

The variety of animation techniques available for small computers and the rich palette of examples already produced, will convince even skeptics that powerful tools are ready to enhance video and cable TV productions. Even the micro graphics computers of today's generation are far superior to the titlers and character generators widely used in audio-visual production.

By now the excuse that computer animation is so difficult that it can only be handled by programming staff can no longer be made. The software is so advanced that it allows the animator to concentrate on the graphic design without bothering about the internals of the assisting computer. What it does require however is the courage to develop new techniques for new machines.

An Application of a Microcomputer as an Interactive Art Medium

George K. Shortess

Department of Psychology
Lehigh University
Bethlehem, PA 18015

Abstract

This paper describes the way that I am currently using a microcomputer as the medium for expressing some art ideas. Specifically, it allows me to incorporate real time interaction between the art piece and the viewer, who thus becomes part of the work.

Introduction

In one sense, all traditional art forms can be considered to involve interactive or participatory activity. Certainly in the visual arts, the viewer typically moves around in the gallery space to gain different perceptions of the painting or sculpture. In addition, the active scanning processes of the eyes change the particular focal areas and the sequences of eye fixations. In both ways, the viewer is interacting with the piece and, while not changing the physical properties of the art object, he/she is changing the nature of the stimulation reaching his/her eyes.

A higher order of participation would be achieved if the viewer could actually manipulate the piece and change it to suit her/his needs, desires or whims. The artist, of course, has always had this experience in the design and execution phases of the work. In recent history, getting viewers more fully involved began in the late 1950's and 1960's with the advent of Happenings and various forms of Conceptual Art.¹ However, with the availability of powerful microcomputers, and their ability to operate in real time, the possibilities of participatory art pieces expanded tremendously. While a number of artists are using microcomputers in interactive ways, I will describe only my own applications.

An Application

The art ideas behind my work have been based on the functioning nervous system. That is, I have selected properties of the nervous system and attempted to give them artistic form, much as a landscape painter selects features of the physical environment and gives them artistic form. Among the various features of the nervous system that I have used are (1) the structure of the neural network, (2) the electrical impulse activity of nerve cells and (3) the interactive quality of the nervous system. These features have been incorporated into various constructed sculptural pieces. They are networks made from 1-1/2 inch square aluminum tubing which simulate, in a formal way, the structure of the nervous system. These structures vary in size from 2 to 20 feet in length and width, and up to 7 feet in height. Small aluminum boxes, varying in size from 2-3 inches, have been attached at the points where the pieces of square tubing join together. Covering these boxes are photographs of images, which were digitized using a television camera input to an Apple II Plus microcomputer, equipped with a Digisector card (Figure 1).

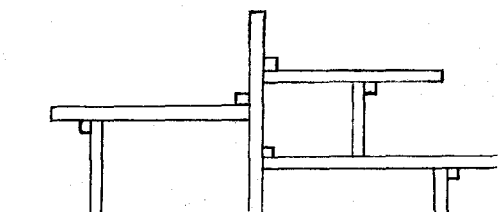


Figure 1. A drawing of a section of one of the sculptural pieces, showing the network of tubing with the small boxes at the junctions (not to scale).

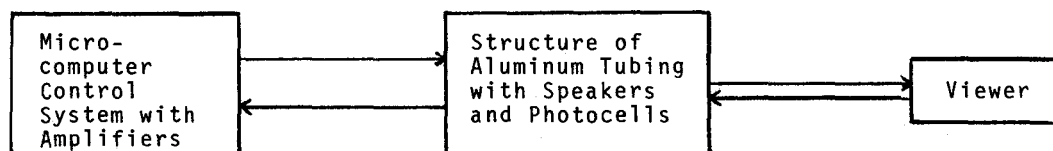


Figure 2. Schematic of Interactive Art Work

In addition, these pieces emit sounds, simulating nerve cell impulses. A simplified schematic is shown in Figure 2. Embedded within the aluminum structures are (1) speakers which emit the sounds and (2) photocells which sense the light from the environment. The microcomputer (an Apple II Plus) is programmed in BASIC to read the resistances of the photocells and to generate the electrical signals which are amplified and fed into the speakers. In its "resting state," with no viewers around, the computer, with the program generates a low level of sound activity in the speakers and continually samples the resistance values of the photocells.

As viewers move to the nearby space, they change the light level reaching the photocells. This causes the micro-computer system to change the particular group of speakers that is activated and to increase the sound activity of these speakers. In this way the viewer can, in a very natural manner, change the physical qualities of the piece and select the kinds of stimuli to be experienced.

The basic program design is relatively straightforward. It consists of several loops and sub-routines. There is a main loop through which the computer "idles" when there are no viewers in the area. It continually reads the resistances of the photocells and generates a low level of sound in several speakers.

When there is a change in the light level falling on a photocell, the program senses this and shifts to another loop, depending upon the particular photocell activated. This loop causes increases in the sound activity in the speakers near the photocell that had "experienced" a change in light level. Then, depending upon further changes in the light to that photocell or to others, the sound patterns change in terms of both how and which speakers are activated.

The sound simulates a form of electrical activity generated by nerve cells. They are called nerve impulses (Figure 3) and consist of a sequence of brief (several milliseconds) changes in the electrical potential occurring across the membrane of the nerve cell. The amplitude (voltage) of each impulse is the same (approximately 100 millivolts). With an appropriate sensor inserted in an active nervous system, these voltage changes can be detected, amplified and played through a speaker. The result is a sequence of click-like sounds with each click (impulse) of the same loudness. (For more details of these neural processes, see Carlson.²)

The sounds generated by the computer and amplified for the speakers in my art work are similar. The rate of "impulse" generation changes in response to changes in the viewer's behavior, much as most nervous systems do. In this way, I have created computer controlled art works

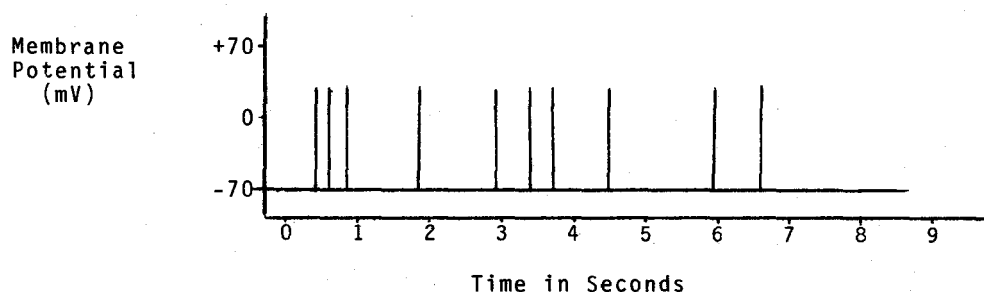


Figure 3. An example of impulse patterns. Each vertical line represents one impulse.

which incorporate certain features of the nervous system. There is a fixed structure which responds interactively with a viewer by changing its pattern of simulated nerve impulses.

A video tape of a viewer interacting with one piece, "Network," will be presented and discussed.

References

1. Arnason, H. H. History of Modern Art. New York: Harry N. Abrams, Inc., 1977.
2. Carlson, N. R. Physiology of Behavior (2nd ed.). Boston: Allyn and Bacon, 1980.

AN ARTIST'S COMPUTER SYSTEM

Walter Wright

Digital Image Corporation

ABSTRACT

Digital Image Corporation makes computer generated 35mm slides for audio visual producers and graphic artists. We aspire to be better and less expensive than the competition. Therefore, we use relatively inexpensive hardware and we write our own software. I will list briefly our hardware and then describe in more detail our graphics software.

HARDWARE

Our hardware includes a Cromemco Z2D microprocessor, video terminal, dual 5 $\frac{1}{4}$ " floppy disks, 64 K bytes of random access memory, the SDI graphics controller and two 48 K image buffers or pages. As inputs we use a Houston Instruments Hipad digitizer and a Via Video digitizer. This latter device consists of a b and w video camera and an interface to digitize and store the camera image in either image buffer. As outputs we have an Aydin Controls RGB color monitor and a Matrix Instruments Camera system modified to accept a pin registered Nikon 35mm camera body. The Matrix camera takes the separated

red, green and blue video images from the computer, displays them, in turn, on its own high resolution b and w monitor and exposes each image through the appropriate color filter onto a single frame of 35mm film. See Figure 1.

Worth approximately \$35,000 this system isn't for the average hobbyist but it is within the grasp of enterprising individuals like ourselves.

SOFTWARE

Our software is designed to make commercial slides. This means word copy slides. Slides for speaker support, to augment training manuals, title slides, tables of numbers and so on. We make charts, graphs and special effects slides. Posterization, digitizing a video image and adding color with the computer, is a hot item.

Beginning at the beginning, our software includes image making programs such as FONT, a program which produces word copy or text in a variety of typefaces or fonts. Next, image manipulating or "cut and paste" programs such as MENU. Bitpad driven, this is our most powerful program, the heart of the system. More about MENU later. Finally SHOOT retrieves completed images and outputs them to the camera. See Figure 2.

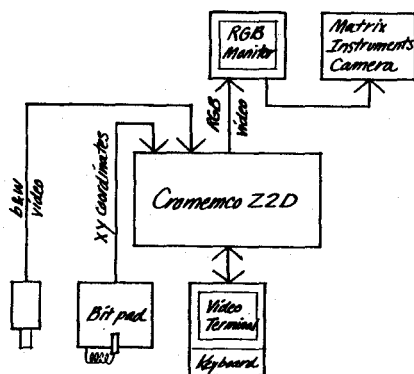


Figure 1

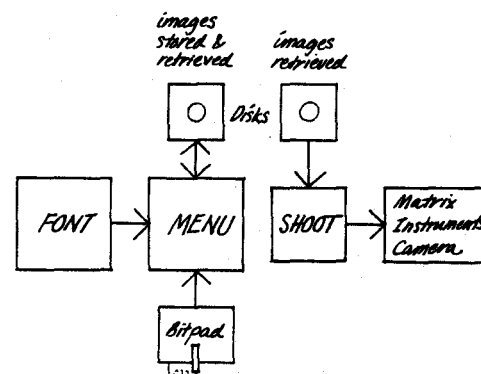


Figure 2

DISK	RES	CLR		DISK	DISK	DISK		CAM	CAM		PAGE			RUN	RPT	BEGIN	END	DEL	ESC	STOP
ADD	SUB	AND	OR	XOR	REPL						ALT	LB	FF			PUT	GET			
AV	FUN	COPY	KEY	PUZZ		SHIFT	JUST	ADJ			1-15	-	=	CE						
						↑		↑			12	13	14	15		+I	+R	+G	+B	
SEED	LOOP	JUMP				←	Y	→			8	9	10	11		-I	-R	-G	-B	
						↔		↓			4	5	6	7		RND				
ASSIGN	FIND	HILITE	BOX	BOX						BOX	0	1	2	3	COLOR	SET	SHADE	HILITE	COLOR	COLOR

Figure 3

MENU

At the creative center of our system sits MENU. The user communicates with MENU using the bitpad and occasionally the keyboard. MENU communicates with the user on the video terminal screen. Looking at the bitpad, the upper portion is an array of 21 x 7 function keys. These keys are activated by depressing the bitpad stylus within the key boundary. See Figure 3.

The top line of keys control basic program operations such as input/output, page selection, resolution, macros and program termination. Near the center of the array is a number pad. On the bottom left of this number pad is a key labelled BOX; keys to the left of the number pad control boxes. Boxes are defined using the ASSIGN, FIND and BOX=BOX keys. Boxes are moved and aligned using the SHIFT, JUSTIFY, ADJUST and directional arrow keys. The YES key confirms possibly dangerous operations. Various operations can be applied to boxes, groups of boxes or images with the AVERAGE, FUNCTION, COPY, KEY and FUZZ keys. These operations can be applied in ADD, SUBTRACT, AND, OR, XOR or REPLACE function modes. On the bottom right of the number pad is a key labelled COLOR; keys to the right of the number pad control color. Colors are set using the SET RGB, RANDOM COLOR, SHADE and COLOR=COLOR keys. Color intensity and red, green and blue components can be fine tuned using the I, R, G and B keys. Color maps can be saved and retrieved with the PUT MAP and GET MAP keys.

In summary, the user controls 16 boxes per page and 4096 colors. He/She can recall any number of images from disk. Function modes include add, subtract, and, or, xor, replace. These modes can be invoked for any disk load, camera load (digitizing an image from the video camera), average, copy, key and fuzz. These operations apply to image loads and to boxes. Individual boxes or groups of boxes can be filled, inverted, cleared, shifted,

justified, adjusted, equivalenced, assigned and found. Colors are set, shaded, raised or lowered in intensity, separated into red, green and blue components, equivalenced and transliterated. Color codes are used by the function modes to produce effects such as smoothing, edge detection, surface textures, etc. MENU allows the user to define macros which become higher-level operations and can be used to define grids, shading, animation effects etc.

ART

Artistic expression is difficult to achieve in any medium. Computer art generally emphasizes technique over expression. Artifice is not art. Artistic expression is akin to personal expression, an artist develops his/her own style. Technique is important as a starting point and as a foundation to build upon. MENU demands this technical skill and, in turn, offers the potential for developing individual style. To illustrate I will describe one aspect of my own work in more detail.

FEEDBACK

Abstract expressionism is my favorite school of painting. The subject or content is spiritual, the soul of the artist. It recognizes feelings and emotions; defines structure in terms of balance, pattern, rhythm, harmony, contrast, counterpoint, etc; and defines beauty as appropriateness of form (content plus structure).

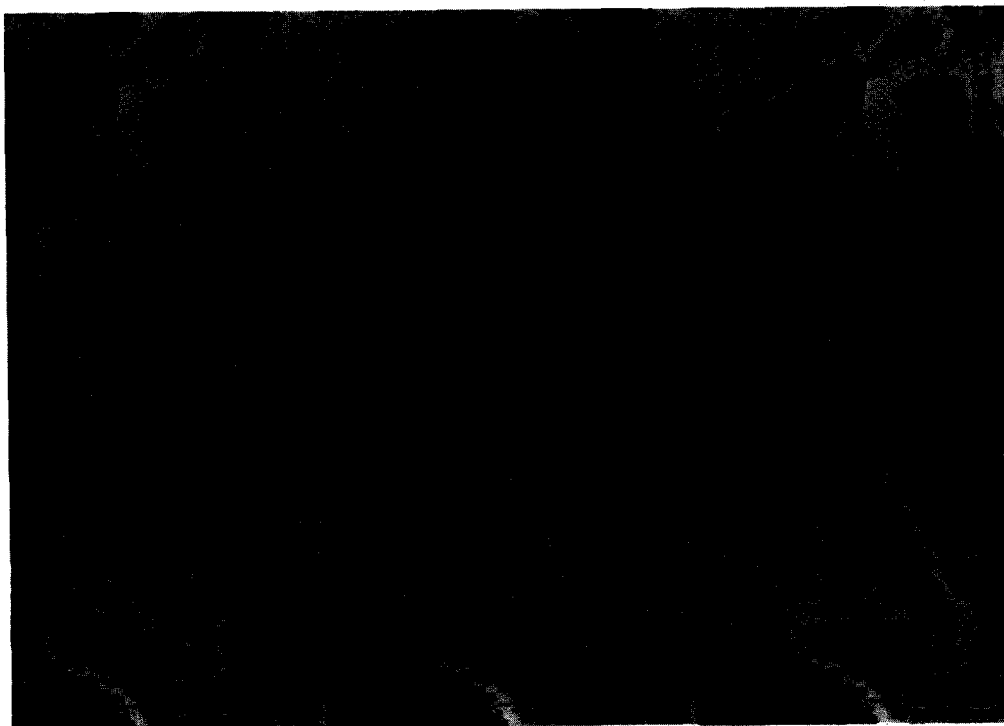
Video feedback has interested many artists in recent years. It produces detailed and dynamic images which symbolize, for many, cybernetics, the computer revolution and so on. Its basic structure is recursive, an important programming technique of some philosophical interest. A feedback image is produced through a combination of controlled and random elements. Again, for many, this symbolizes the electronic age.

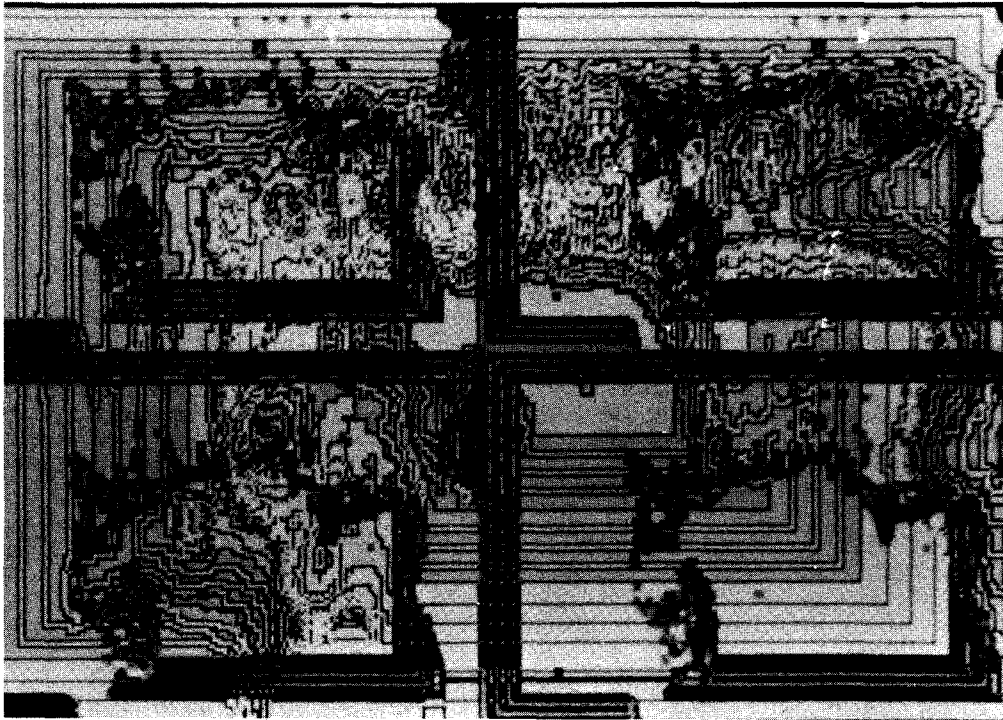
My feedback images are created by pointing the video camera at its own monitor. Noise and sources of light reflected on the monitor screen are repeated within the loop, like a hall of mirrors. This conventional feedback loop is modified by our slow column by column scanning process. The image is controlled by varying camera position, lens settings and monitor brightness and contrast. After considerable tinkering acceptable images are selected and stored.

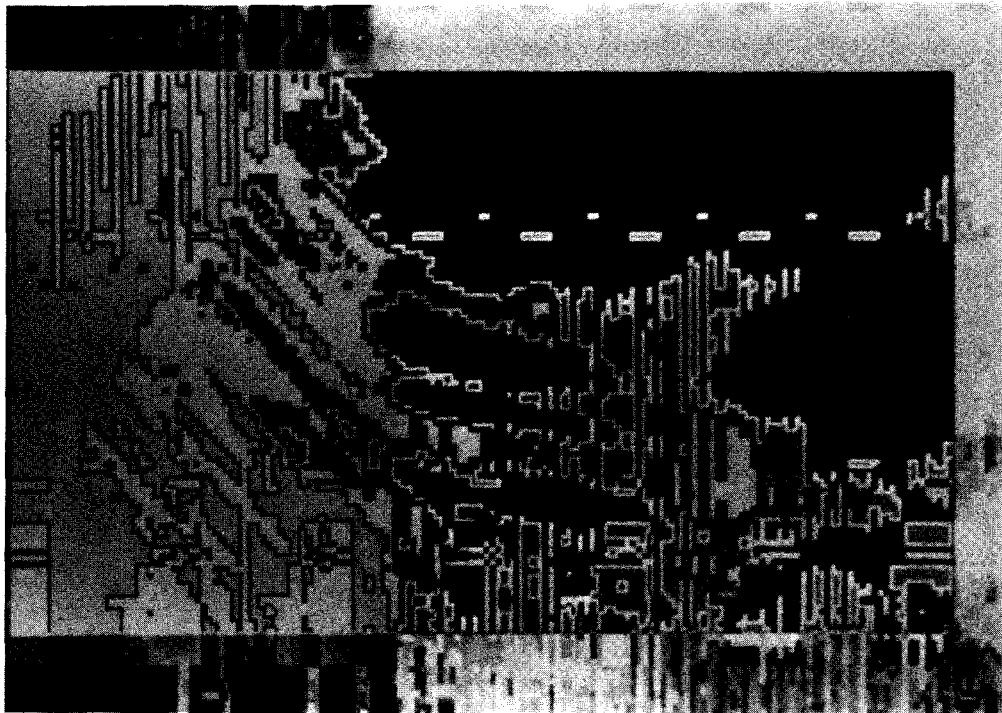
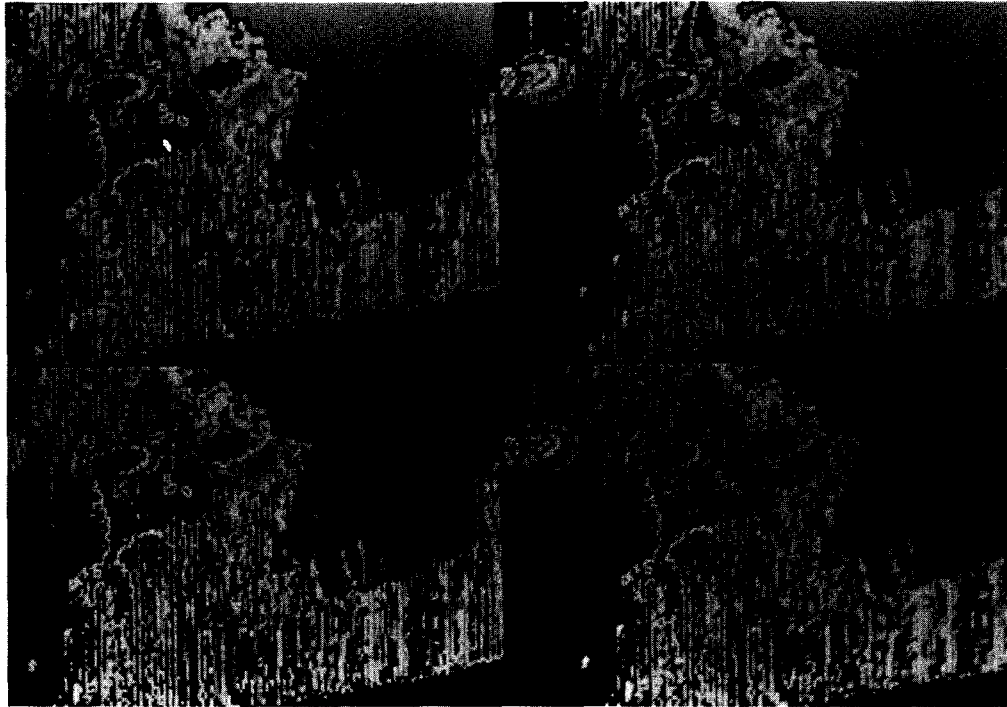
I use a panoply of techniques to develop these basic b & w images. Fuzzing with the replace function smooths the image. Fuzzing with the subtract or xor function produces an edge detected outline. This outline can be averaged back into the original image in various modes. Fuzzing with the add function then loading the original image with the and function produces cutouts. These can be used as mattes for collaging. Sections of the image can be assign-

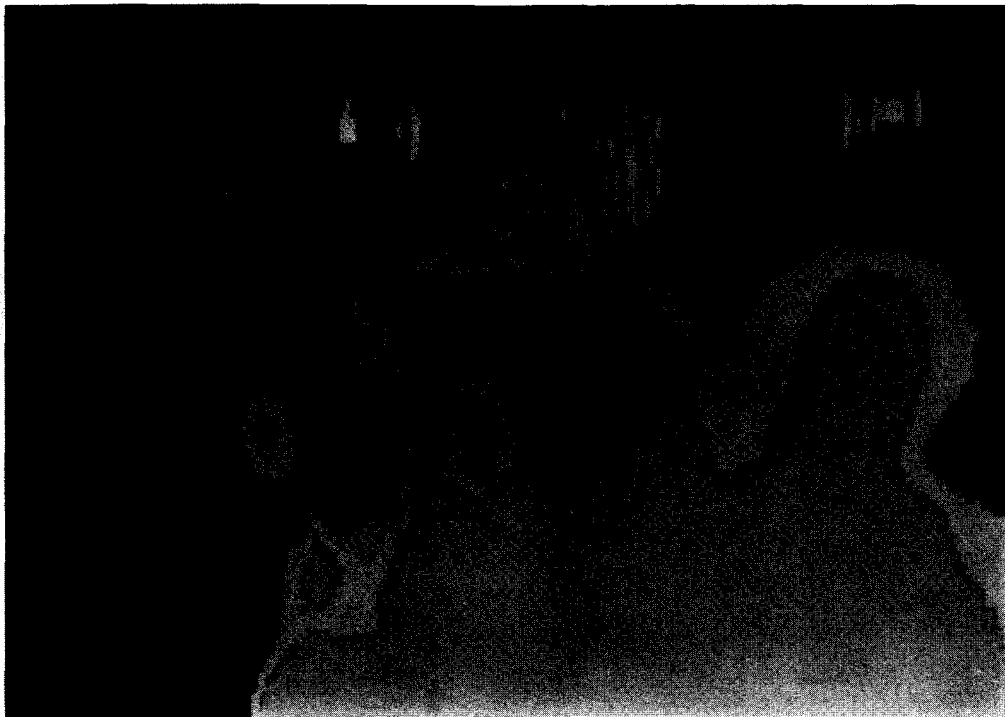
ed to boxes and these boxes keyed over the full frame to produce foreground or background patterns. Colors can be transliterated. I clean up a complex image by setting colors 1 - 3 to 0, 5 - 7 to 4, 9 - 11 to 8 and 13 - 15 to 12. Selecting colors can be approached in many ways. Often, I randomly set colors 0 and 15, then shade the map to produce an overall tone, and finally randomly select only the odd or even colors. The combined results of these and many more related strategies are illustrated in the photos following this article.

For me, using MENU in this manner parallels the process of painting. I am directed by my intuition. I respond to the image as it develops on the color monitor. I can store an image I'm having trouble with and return to it later. I can make false starts, backup and start again. I can develop several images simultaneously. The computer is my studio.











VIDEOTEXT AS PERSONAL EXPRESSION

WENDY RICHMOND AND SUSAN RUBIN

ABSTRACT

"Videotext as Personal Expression" is a description of two artists' participation in the creation of a videotext electronic art gallery. This project was funded by the National Endowment for the Arts through a grant awarded to the Alternate Media Center at New York University. The first section is a description of the graphic workings of the videotext system. The second section is a discussion of Susan Rubin's interactive electronic calendar, created for the project. It encompasses an evaluation of the database for the calendar, as well as her methodology of working with the medium. The third section describes Wendy Richmond's project, an exploration which emphasizes the search for an appropriate visual language and imagery for the videotext medium. The conclusion addresses the definition of an electronic art gallery in the home.

Videotext is the generic name for the transmission of computerized information and services to the home television screen. It is delivered via cable or telephone line, and is accessed by the home viewer using a hand-held decoder or computer keyboard. The videotext displays are static pictures, or "pages" which are called up individually by viewer request. These pages can be either text or graphic information, or both. Videotext is interactive, in that the viewer selects the pages she wishes to see.

The Telidon videotext system, which was selected because of its enhanced graphic capabilities, provides a palette of six colors--red, blue, yellow, green, cyan, and purple--as well as six shades of gray from black to white.

Its drawing system is based on Picture Drawing Instructions, or PDI's, which enable the user to create points, lines, arcs, circles, rectangles, and polygons using a compact set of commands. These shapes can be drawn in any specific location using a keyboard, joy stick, or electronic stylus to position the cursor.

Telidon offers textures in the form of grids, diagonal lines, dot matrices, and stripes, all in changeable widths. These textures can be used to draw and fill, and can be overlayed to create variations in pattern and tonal value.

Each page is created by issuing a sequence of commands. When the image is displayed, it appears to draw itself in sequence for the viewer. This step-by-step picture creation offers the potential for limited movement and animation.

In addition, the Telidon system enables the artist to edit her pages, providing commands to change colors, move forms, rotate and scale shapes, erase specific areas, and combine part of one page onto another.

Generally, videotext services consist of news, financial data, sports scores, community activities, weather reports, travel information, and the like. In the fall of 1981, Martin Nisenholtz of the Alternate Media Center at New York University received a grant from the National Endowment for the Arts to create an electronic art gallery using Telidon. The grant was awarded to artists selected for their abilities to explore the graphic potential of the medium, and to extend its application as an art form.

This paper is a collaboration written by two of the participating artists, Susan Rubin and Wendy Richmond. They come to the project with complementary yet different backgrounds and attitudes. Ms. Rubin is an animator with a persistent interest in calendars. Her approach lies in translating her interest into a new medium. Ms. Richmond is a graphic designer whose work focuses on exploring the visual language with which a given medium communicates. Her primary objective in this project is to discover the appropriate imagery to be viewed on this medium, and the appropriate style in which to portray the chosen imagery. For Ms. Rubin, videotext is yet another graphic tool for creating personal imagery. For Ms. Richmond, it is an extension of her research and evaluations of the uses of media.

INTERACTIVE 1983 CALENDAR Susan Rubin

I was invited to participate in the project based on my work as an animator. The calendar has been a theme in my work for the past six years. Four of my films are animated months. I also designed a calendar for 1982 using xerox, collage, and rubber stamping. So for the Telidon project, I proposed to create an electronic interactive calendar, with an emphasis on its graphics. My approach was to extend my interest in calendars to a new graphic medium, and to find the appropriate structure, imagery, and vocabulary within that medium.

The database for the calendar is the twelve months, each created in a different graphic style using the elements of the system. For each month, there is a selection of four additional pages, each keeping within the graphic theme and "look" of the month of which it is part. There is a holiday page, depicting reference to a holiday celebrated during the month. There is a recipe page, offering a different recipe for each month. The order of the recipes throughout the year follow the order of a meal, from appetizer through dessert. There is a quotation page, which presents quotes from poetry, prose, and songs. And there is a page for each month which is one installment of a twelve-part story which progresses through the year.

The calendar is designed to offer the user an initial choice of any of the twelve months. Once the selected month has been displayed, the viewer is asked to move on to the next menu. At this point, she can select either a new month, or a choice of holiday, recipe, quotation, or story. If she selects, for example, the recipe page, the recipe for the month will be displayed. At this stage, she is given the option to view other recipes, or to go back to the month's menu and select either its holiday, quotation, or story. She is also offered the opportunity to return back to the cover menu, from which she can select another month. (See Figure 1)

The interactivity of the calendar in this medium allows the viewer to travel multi-dimensionally through time and content, month by month, through the holidays, recipes, and quotations one by one, and through the story chapter by chapter. She can also elect to view each full month, with all of its options, one at a time. As previously explained, all options for each month are graphically similar, whereas moving "horizontally" through the months, holidays, recipes, quotations, and story results in twelve different graphic styles.

It took several weeks for me to become accustomed to the system and to produce images that I found acceptable. Initially, I attempted to sketch certain ideas on paper and translate them onto the monitor. The results were frustrating. I found it necessary to spend several sessions playing loosely with the system in order to develop a feel for the kinds of images I was looking for. Eventually, my sketching as well as my final pages were created solely on the system.

As I became more familiar with creating these pictures, I developed certain preferences and styles of working. For instance, when the program is booted, the screen is black. At first, I drew my shapes against the black background. It didn't occur to me for several sessions that I could change the color and texture of the background. In the final calendar, a good number of my pages were drawn on other-than-black backgrounds.

I also found that I preferred working in certain modes. I discovered the polygon mode much more useful to me. And I began to know which fill patterns I wanted.

I developed the habit of storing successive stages of pages as I was creating them, and of saving as much as I could, even if I was not intending to use it. As it turned out, a number of early sketches were eventually developed and incorporated into the final calendar.

Essentially, I found the system a satisfactory vehicle for my personal creative expression. As the calendar evolved, I discovered a language with which to produce imagery which was aesthetically gratifying and exciting.

CULTURAL PATTERNS Wendy Richmond

As a graphic designer, my approach to creating electronic videotext art consisted of an exploration of the medium itself. What are its limitations? What are its advantages? How will the audience view the images? What is the appropriate content for this medium, and what is the appropriate style in which to display this content?

The limitations include extremely low resolution, limited choice of colors, small screen size, and slow display time. Each image is always displayed as it has been constructed: piece by piece. The viewer must "request" each image. It is impossible to represent highly realistic, photographic images.

Conversely, each limitation can be a strong advantage. The extremely low resolution can be used to exhibit rich, patterned textures -- electronic weavings, mosaics, tapestries created with electron beam threads. The colors are emitted rather than reflective, and are bright, strong, highly graphic. As the image builds up, piece by piece, its form unfolds. From mysterious disconnected shapes, it evolves, like a story, into a final, fully constructed image. The viewer can request any image, and construct her own sequence. And finally, the images are not of our reality, but characters of an electronic world.

The key, then, is to analyze the limitations/advantages, and use them appropriately.

Information displayed on a CRT screen is never literally static. The picture is constantly being redrawn by an electronic beam which traverses the screen left to right, top to bottom, completing an entire frame every thirtieth of a second. Although the picture may not actually move, it is never totally stable. One of the most important requirements of textual typography is that it be extremely stable, allowing the reader to progress smoothly through the text without encountering visual breaks and inconsistencies. Videotext type, because of the medium's extremely low resolution, is made up of very few pixels, thus causing the type to be jagged and poorly articulated (not unlike the type that you are reading at this very moment, that of a low resolution dot matrix printer).

Therefore, I decided that the frames should rely primarily on graphics, and text should be confined to short, caption-like material. All graphics themselves should be strong and free from miniscule detail which does not "read" in such low resolution.

My electronic imagery consists of masks, figures, and tapestries of primitive art. Each of the three categories is intended to support the physical nature of the videotext medium in its own manner.

The masks of primitive art are strong and highly graphic. They have immediate emotional impact. In translating these images to video, I strove to retain this visual and emotional impact, creating large, looming "faces", full of benevolence, anger, or fear, with an imposing stature. As elements build on the screen, the mask emerges, and its emotions unfold. It floats in space, with no landscape.

The figures that I chose to translate into electronic form offer a different kind of appropriateness to the medium; they are geometric, with sharp edges and angles. They are bright and cartoon-like, and have a surprisingly (or not so surprisingly) strong resemblance to the figures of video games.

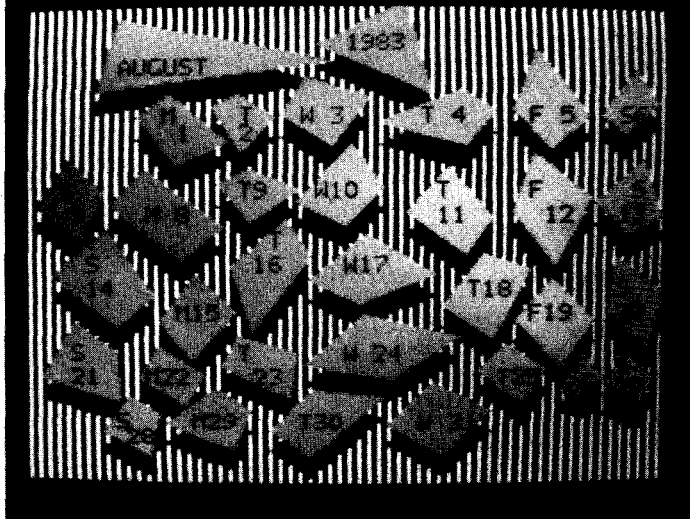
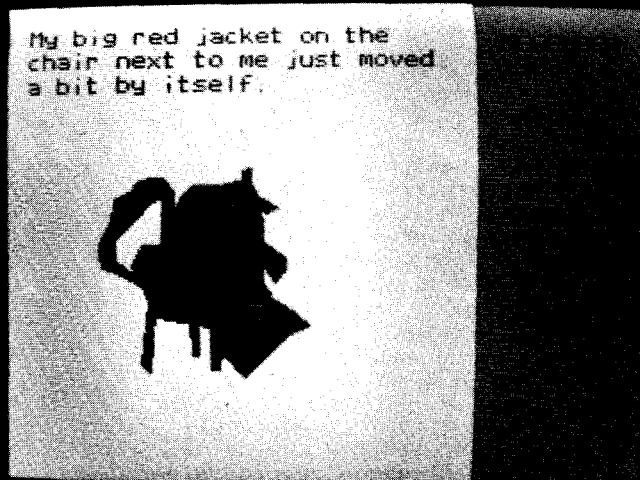
The tapestries have the most obvious and concrete appropriateness to low resolution computer graphics. They can be translated almost literally from their original form, where the patterns and textures provided on the Telidon system are extremely similar to patterns and textures of weaving.

The viewer has a choice of two paths in which to travel through the database. She can view a sequence of masks, figures, and tapestries from a specific geographic origin, such as Africa, Peru, or the Pacific Northwest. Or, she can choose to view a sequence of either masks, figures, or tapestries from a variety of geographical origins.

In conclusion, a note regarding the illustrations. Because this is a "paper", a hard copy, black and white, primarily verbal description of a soft copy, color, primarily graphic set of images, I am compelled to allude to the irony of displaying the artwork in this paper. Therefore, I am including additional images more appropriate to the medium at hand, that is, hard copy sketches.

CONCLUSION: WHAT IS AN ELECTRONIC ART GALLERY?

An electronic art gallery is one which is viewed at home, on a television monitor. It can be viewed at any time, at any speed, in any sequence. It differs greatly from the art gallery as we know it. Think of the way in which you normally view art. You allot a Sunday afternoon, most likely only several per year, outfit yourself accordingly, leave your home, and drive or take the subway to the chosen location. The purpose of your trip is to view art. Anything that happens during your visit that does not involve looking at art is secondary. Think of viewing an art gallery at home, on your television set. Are you eating dinner at the same time? Doing the dishes? Waiting for a phone call? Why not?



Susan Rubin

INTERACTIVE VIDEOTEXT CALENDAR
Susan Rubin

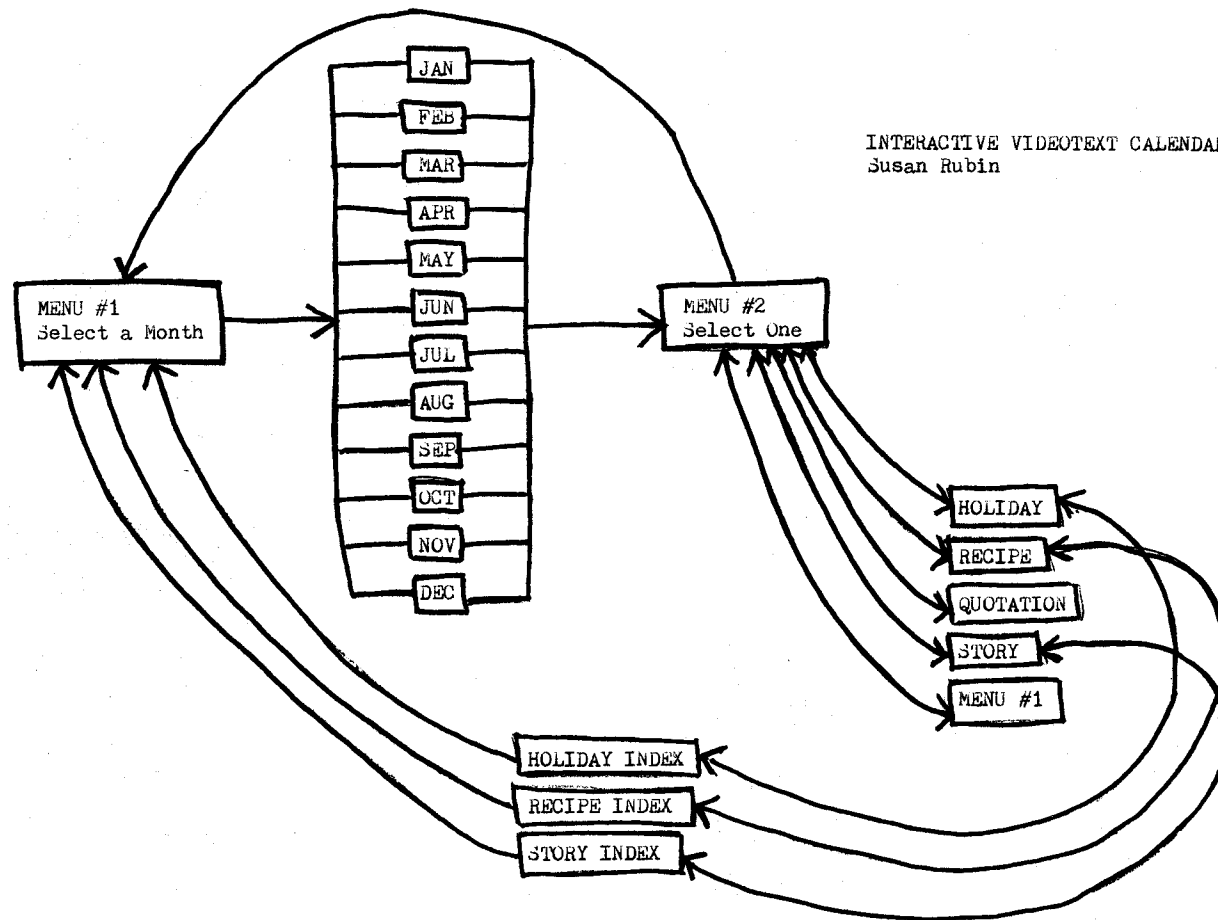
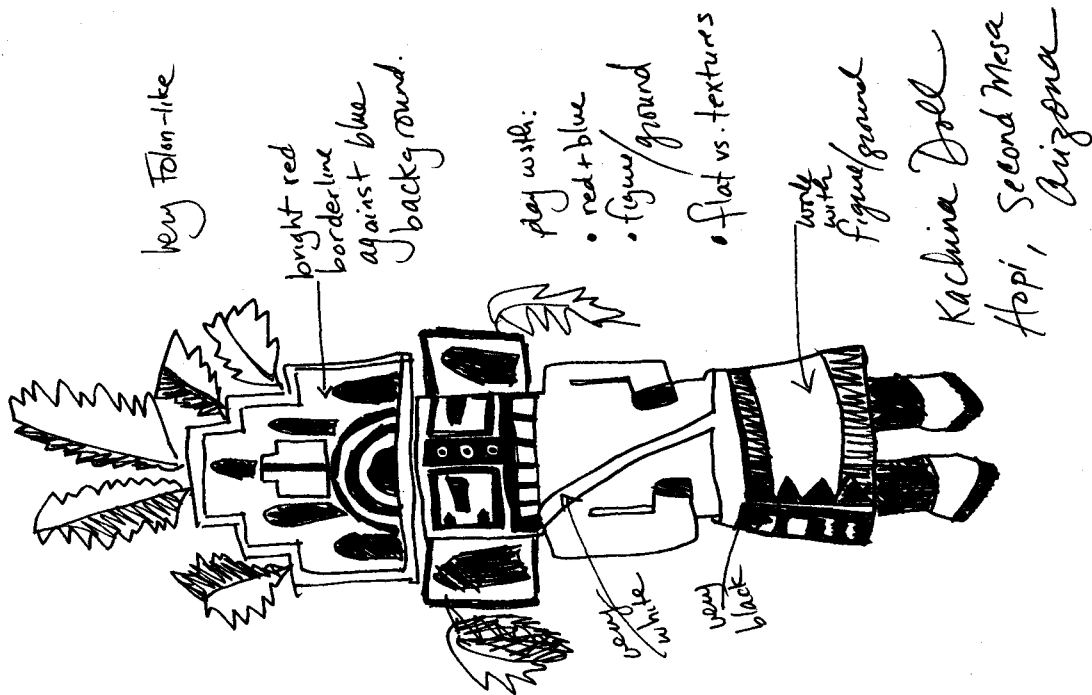
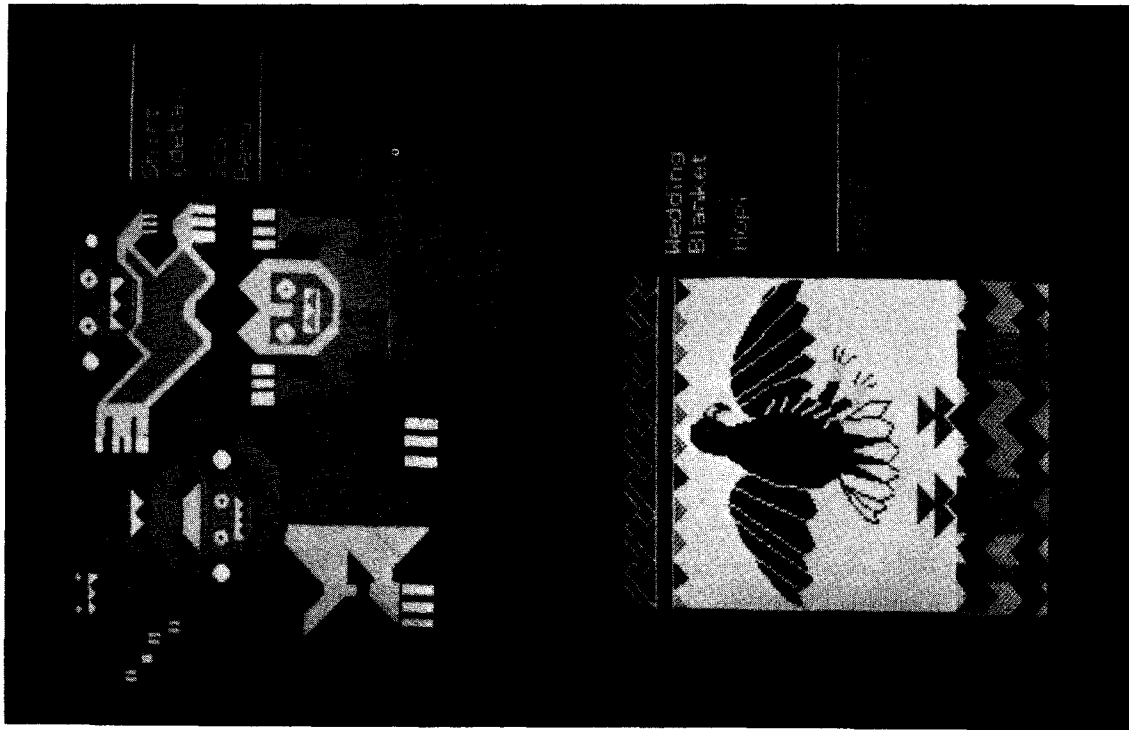
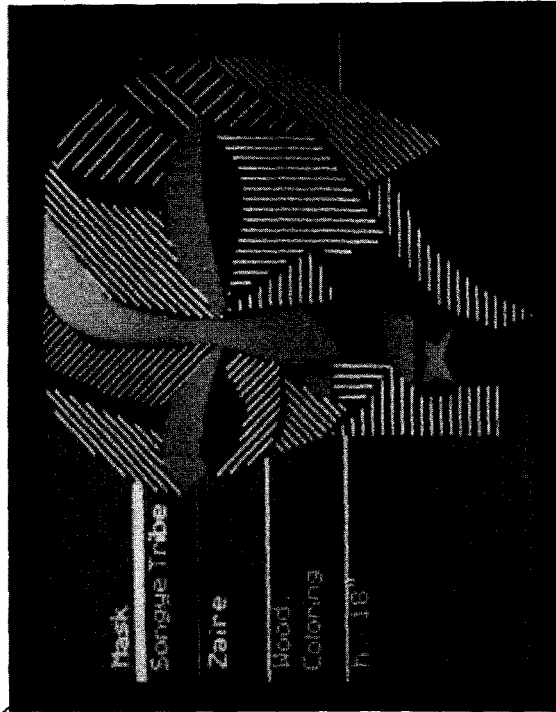


FIGURE 1





Wendy Richmond

COMPUTER PAINTING WITH RODIN

by Monique Nahas <*> and Herve Huitric <*>

Production Automation Project
University of Rochester
Rochester, New York 14627

<*> On leave from the University PARIS VII, Paris, France.
<*> On leave from the University PARIS VIII, Paris, France.

ABSTRACT

This paper presents computer generated artistic images produced by a system called Rodin, which we designed and implemented. Rodin utilizes the following techniques.

- Surface design with bicubic B-Splines
- Elimination of hidden parts
- Generation of shadows
- Surface modulation
- Blending or concatenation of surfaces
- Texture generation
- Recursive subdivision
- Reflection, refraction, and transparency simulation
- 3D interpolation

SURFACE DESIGN

We model objects by using bicubic B-Spline surfaces because these are well suited for constructing smooth 3-D objects. A B-Spline surface is defined by a set of "control points", with each point represented by its x,y,z coordinates. B-Splines are not interpolating surfaces, i.e., they do not pass exactly through their defining control points.

The control points to produce a desired 3-D surface can be estimated by a number of different ad hoc methods. We illustrate the use of the methods through several examples.

1. For the head in Picture 1 we used $20 \times 40 = 800$ control points. An initial set of points was selected by hand drawing two views of a face (front and profile). This technique is similar to the data acquisition procedure described in [1]. We spent several days manipulating manually the control points to achieve a satisfactory result. For an easier manipulation, we implemented algorithms based on the so-called "Oslo algorithm" [2] for the local non-uniform introduction of new control points. This provides a means for modifying objects locally in the regions where the objects are richer in details.

2. For the dinosaur in Picture 2, we started with a wood skeleton and measured by hand coordinates of points along the spinal column and various cross-sections ("slices"). Initial estimates for control points were computed by positioning the slices appropriately with respect to the spine. This spine/slices combination has the advantage of being rapidly adaptable for an animation sequence: this only requires changing the spinal column position and the orientation of slices.
3. The control points of a surface can also be generated algorithmically. For example, the control points for the mountains of Picture 1 were generated via fractal curve algorithms [3]. The control points for the lake of Picture 3 were defined by superposing several waves generated algorithmically.

Improving the methodology of data acquisition for complex scenes is of fundamental importance to our overall work. Consider, for example, the construction of a pseudo-realistic movie, and think of how many different simulated objects and actors are needed to keep an audience from being bored after a few minutes.

OBJECT VIEWING

Depth Buffering

Rodin calculates visible surfaces and shadows by using algorithms based on depth buffering [4] which are similar to those described in [5]. These algorithms were chosen for ease of programming.

Visibility calculations are performed as follows. We step along the u,v parameters which define the surface in equal increments and generate the corresponding points P(x,y,z) on the surface. Each point P is projected onto a discretized "screen" to determine a corresponding pixel. The depth (i.e., the distance VP, where V is the viewpoint) is

computed and compared to the depth stored in a Z-Buffer for the appropriate pixel. If P's depth is smaller than that in the Z-Buffer, then P's depth is stored in the buffer.

Shadow calculation is similar to visibility calculation. We use an S-Buffer in which we store distances SP between the (point) light source S and surface points P. The S-Buffer corresponds to an "auxiliary screen" approximately orthogonal to the "average" light direction. The Z-Buffer and the S-Buffer are computed in a single u,v scan of the surface.

To produce a display with shadows we proceed as follows. We traverse the surface again. For each point P we determine visibility by comparing the depth VP with the appropriate entry in the Z-Buffer. If P is visible, we project it onto the "auxiliary screen" and find if P is illuminated (or if it is obscured by another point) by comparing the distance SP with the contents of the S-Buffer. If P is not illuminated then it belongs to a shadow and is painted black. If P is illuminated then we compute its corresponding intensity by using Lambert's law or different formulas as described in [4], and display it. Surface normals are computed by finding the partial derivatives with respect to u and v by using de Boor's formulas [6], and then evaluating the cross product of these derivatives. We have encountered the two following problems with the above method:

1. When two neighboring points P and P' on the surface project onto the same S-Buffer "pixel", the point which is farthest from S will be assigned to the shadow, and the object will appear with some rather strange "self shadows". This problem is easy to solve by introducing a tolerance in the distance comparisons. Specifically, if the distances SP and SP' are approximately equal, both P and P' are considered illuminated.
2. The surface often is "under-sampled", i.e., not enough points on the surface are generated to fill completely and correctly the Z- and S-buffers. The effects of under-sampling are quite visible in the shadow cast by one object upon another. For example, a shadow point P may be found by the algorithm to be illuminated simply because a portion of surface which obscures P was under-sampled and therefore was not taken into consideration in the calculation of the S-buffer. Under sampling effects can be avoided by using a very large number of points, which is an expensive solution, or by interpolation.

Interpolation method

To reduce computing time we use the

following interpolation technique. First we compute the points of the surface corresponding to two successive discrete values of v and to all the discrete values of u. After projection on the screen, two sequences of 2-D points are thus obtained. Then we interpolate each polygon of four points as shown in Figure 1.

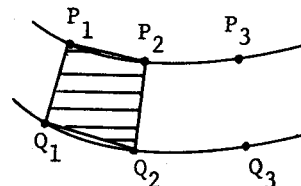


Figure 1

To each corner of the 2-D polygon being considered, we associate a set of values to be interpolated simultaneously: (a) brightness value, (b) the contents of the Z-Buffer, (c) the values of the two indices of the S-Buffer, (d) the contents of the S-buffer itself, (e) the depth, and (f) the S-depth, i.e., the distance between the point and the light source. During the interpolation the shadow and the hidden parts are evaluated for each point. Note that a Gouraud shading [7] is included automatically.

Brightness

The brightness of a point can be computed by using Lambert's law or several other formulas [4]. Highlight effects are used in Picture 10 to enhance the rocks in the foreground. Fog effects are used in Picture 7; fog is created by adding a diffuse light whose intensity is proportional to the distance to the eye. Other effects can be created by modulating the light according to different parameters (e.g., the altitude). All these effects seek realism. In addition, certain promising effects can be produced by random modulation of the light around an average computed value.

SURFACE MANIPULATION

Local Surface Modulation

We modulate a surface by perturbing its defining control points. First the Oslo algorithm [2] is used to add the desired number of control points to a selected region of the surface. Then the control points are moved according to a specific rule. For example, in Picture 4 control-point perturbations were sinusoidal, with variable frequencies.

A local modulation is obtained by controlling the displacement of a subset of a

surface's defining points. When the point perturbation in a region has zero amplitude, the initial surface appears without deformation in a subregion because of the local properties of B-Spline surfaces. Local modulation was used, for example, to construct the head of a faun in Picture 5.

Concatenation of B-Spline Surfaces

Blending two or more B-Spline surfaces together is required to obtain more complex shapes such as a body with limbs or a tree with branches.

We describe our concatenation process for the specific example of the dinosaur shown in Picture 6. First, subdivision algorithms [2,8] are used to cut the surface into two B-spline surfaces, with two sets of corresponding control points. Then the control points of the foreleg are juxtaposed with those of the first part, and the corresponding surface is displayed as a whole (Picture 6). The process continues by cutting the rest of the body, juxtaposing the new leg and displaying it as shown in Picture 6. Finally the last part, the tail, is displayed alone.

The trees of Picture 7 are made in a similar way, the sole difference being in this case that the process of cutting and attaching a branch is recursive. The tree is made with two B-spline surfaces: a trunk and a branch, and the branch is automatically translated before each concatenation.

3-D Interpolation

By equalizing the number of control points in two B-Spline surfaces, it is possible to combine the surfaces through interpolation in various ways. The combination can be local by controlling spatially the coefficients of the interpolation. The head emerging from a mountain of Picture 8 was constructed in this way, and can be imagined as a part of an animated sequence.

Texture

In Rodin, we are using the ideas of [9]. The texture is produced by modifying a point in the direction of its surface normal. We use different kinds of texture, either by giving a pattern (e.g., the ground in Picture 9), or by using random functions (e.g., the mountains, ground, trees in Pictures 7, 2, and 1).

Many different textures can be obtained by employing stochastic functions or by locally controlling their variations. The texture can be altered in coordination with different parameters of the Picture, the characteristics

of the objects, etc...

For our artistic goals, this kind of stochastic method is very important. It gives some life to the uniformly metallic aspect of the B-spline surfaces and thus we are pursuing the stochastic approach in all the imaginable directions trying to produce hair and skin textures, and stochastic movements [10].

Recursive Subdivision

Using the ideas in [11], we combine Z-Buffer techniques and recursive subdivision to get another way of displaying a B-SPLINE surface. The subdivision is obtained by cutting the given surface into four B-Spline surfaces using [2] and [8]. It is then possible to modify the control points of the subsurfaces in a way analogous to that described in [3]. According to the number of recursions, the amplitude, and the direction of each deformation we obtain different perturbed surfaces as shown in Picture 10.

Reflection and refraction

We simulate reflection and refraction of light on B-Spline surfaces by using the raytracing techniques described in [12]. We traverse the reflecting or refracting surface, and for each point on it we determine the ray going from that point to the eye. Then the brightness of the point is determined by optical geometry: light can come from a ray symmetric with respect to the normal in case of reflection, or from a ray determined by the index of refraction in case of refraction.

To find the intensity along these rays, we intersect them with the other objects of the scene. Then the same process is repeated recursively at each point of intersection. We refer the reader to [12] for a full description of the algorithm. In Pictures 11 and 12 we stopped the process at the first intersection (point A in Figure 2a) in order to produce the reflection of one opaque object -- the body -- in the lake-mirror.

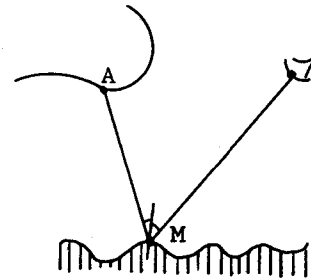


Figure 2a

Reflection

In Picture 1 we stopped the process at the second intersection, i.e., point B in Figure 2b, to compute a refraction through thick glass. (The two sides were obtained by a small translation of an initial vase.) The illumination of A (respectively B) is then computed in the usual way and used as part of the brightness of the point M of the mirror. This method uses certain approximations: for example the influence of the reflected light source, an important factor, is neglected.

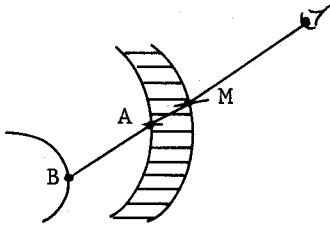


Figure 2b

Refraction

In any case, we must compute the intersection of a ray with a B-Spline surface. We use a recursive subdivision algorithm [2] and [8]. We cut the B-spline into four subsurfaces, and determine if the ray can intersect each of the subsurfaces. If a potential intersection with a subsurface is detected, we subdivide again, and so on. Intersection tests are made by examining the largest cube that encloses the control points of a subsurface. Since B-Splines have the convex hull property, a ray which does not intersect an enclosing cube cannot intersect the surface itself.

Texture can be added to the mirror without difficulties. Note, however, that some problems involving the reflection of a textured object are still not solved.

Transparency

By using a ray tracing procedure it is possible to combine the intensities of different points on the same ray to compute transparency. When there are no reflections or refractions we compute transparency by simple depth buffering techniques using multiple Z-buffers as in [13].

CONCLUSION

Rodin was implemented in Fortran on a VAX 11/780 under VMS at the University of Rochester. The system extends to 3-D scenes many of the ideas used in our earlier work [14], and exploits the specific capabilities of 3-D computer graphics.

In order to address the artistic questions

that constitute our fundamental concern, we must always use the state of the art in computer graphics; this prevents Rodin from ever being a closed system.

The choice of B-Spline surfaces reflects our own preference for smooth surfaces, even if we try to perturb them most of the time. Obviously the choice of tools is a personal affair and is not, and does not have to be the same for all artists.

ACKNOWLEDGMENTS

We are grateful to the Production Automation Project of the University of Rochester for facilitating our artistic research by hosting us during 1981-1982. We would especially like to thank Dr. Herbert Voelcker, Professor and Director of the P.A.P., and Dr. Aristides Requicha, Associate Director, for their kind efforts in our behalf.

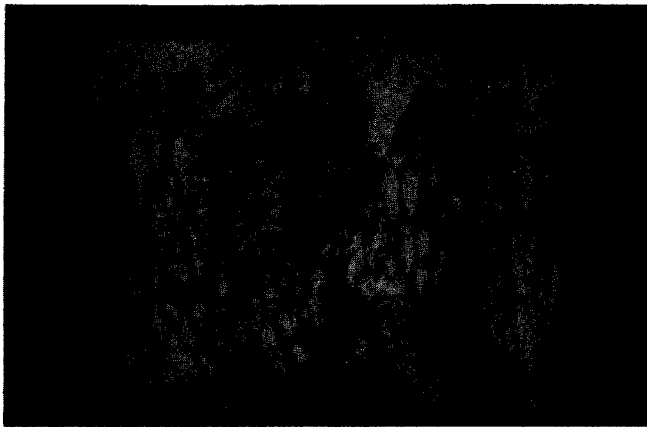
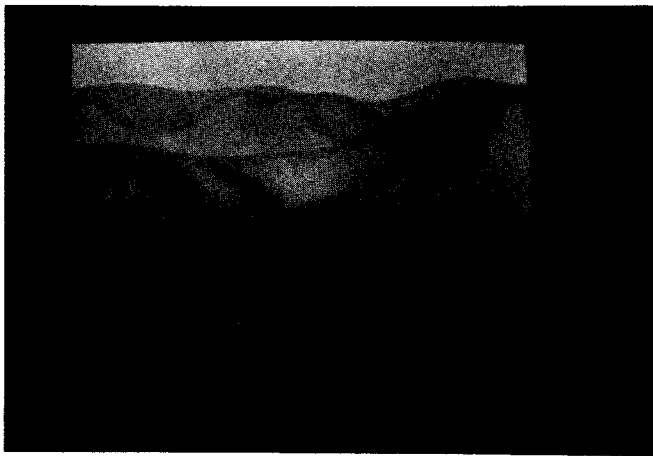
REFERENCES

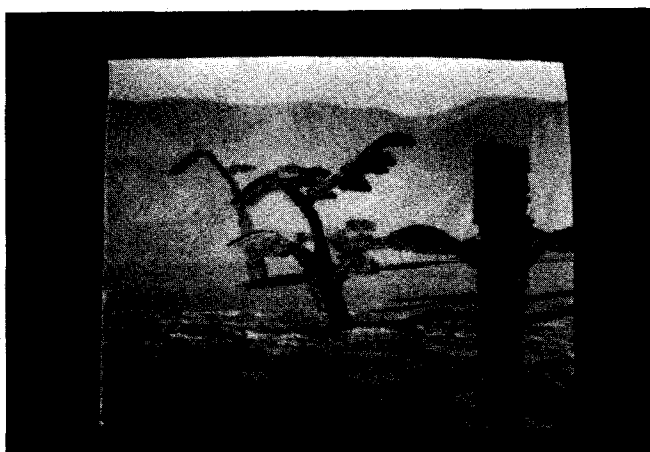
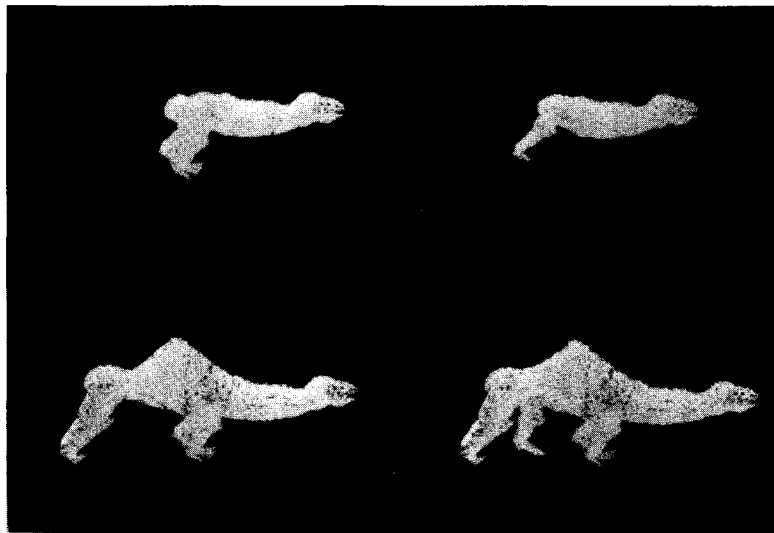
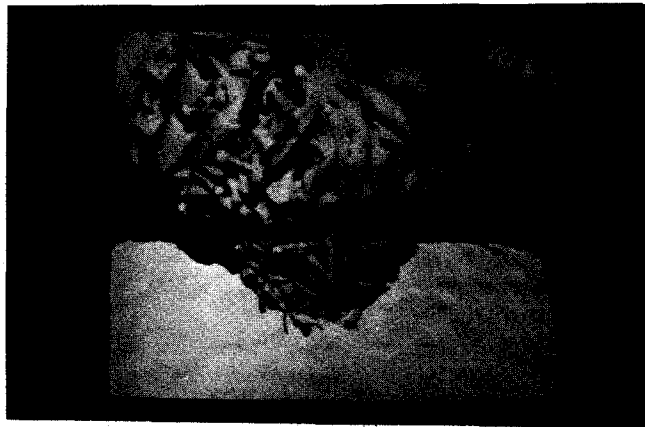
Remark: The references given here are a very small part of all the literature existing about B-Spline and other topics considered in this paper. We list only those papers which are directly relevant to our system. For a very interesting, complete, but mathematical presentation of B-Splines, we add the reference [15].

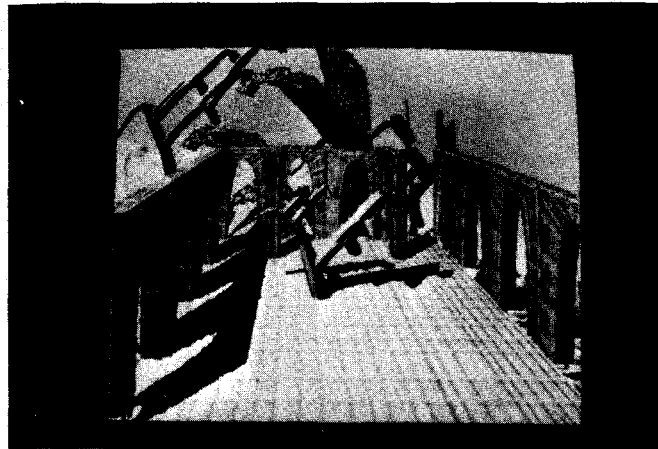
1. F. I. Parke, "Computer generated animation of faces", in H. Freeman, Ed., *Interactive Computer Graphics*. New York: IEEE Computer Society, 1980, pp. 357-363.
2. E. Cohen, T. Lyche, and Richard Riesenfeld, "Discrete b-splines and subdivision techniques in computer-aided geometric design and computer graphics", *Computer Graphics and Image Processing*, no. 14, pp. 87-111, 1980.
3. L. C. Carpenter, "Rendering of fractal curves and surfaces", *ACM Computer Graphics*, vol. 14, no. 3, p. 109. July 1980.
4. W. M. Newman and R. F. Sproull, *Principles of Interactive Computer Graphics*. New York: McGraw-Hill Book Co., 2nd ed., 1979.
5. L. Williams, "casting curved shadows on curved surfaces", *ACM Computer Graphics*, vol. 12, no. 3, pp. 270-274, August 1978.
6. C. de Boor, "On calculating with B-Splines", *J. Approximation Theory*, vol. 6, no. 1, pp. 50-62, July 1972.

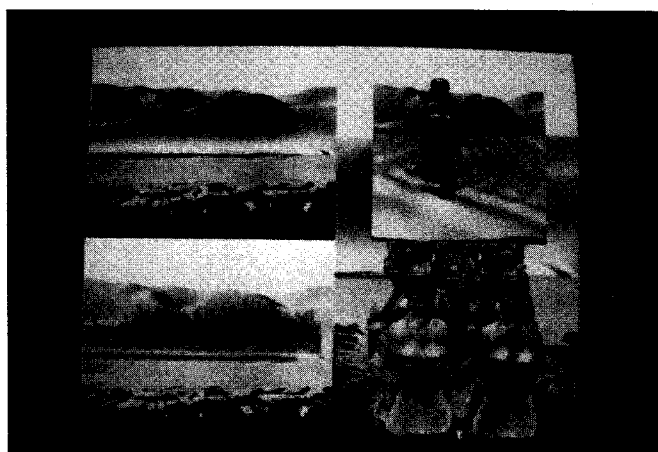
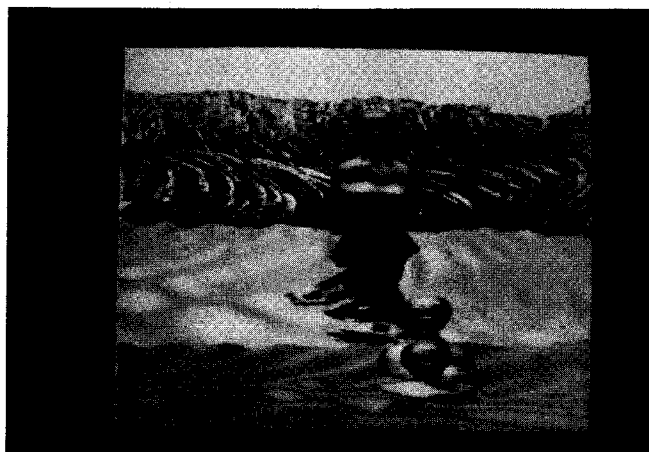
7. H. Gouraud, "Continuous shading of curved surfaces", IEEE Transactions on Computers, vol. C-20, no. 6, pp. 623-628, June 1971.
8. W. Boehm, "Inserting new knots into B-Spline curves", Computer-Aided Design, vol. 12, no. 4, pp. 199-201, July 1980.
9. J. F. Blinn, "Simulation of wrinkled surfaces", ACM Computer Graphics, vol. 12, no. 3, pp. 286-292, August 1978.
10. A. Fournier, D. Fussell, and L. Carpenter, "Computer rendering of stochastic models", Communications of the ACM, vol. 25, no. 6, pp. 371-384, June 1982.
11. E. Catmull, "A subdivision algorithm for computer display of curved surfaces", Tech. Report No. UTEC-CSC-74-133, University of Utah, December 1974.
12. T. Whitted, "An improved illumination model for shaded display", Communications of the ACM, vol. 23, no. 6, pp. 343-349, June 1980.
13. P. R. Atherton, "A method of interactive visualisation of CAD surface models on a color video display", ACM Computer Graphics, vol. 15, no. 3, pp. 279-287, August 1981.
14. H. Huitric and M. Nahas, "Computer art experiments of the last ten years", Proceedings of IEEE Symposium on Small Computers in the Arts, IEEE catalog no. 81CH1721-0, pp. 87-89, November 1981.
15. C. de Boor, "A practical guide to b-splines", Applied Mathematical Science, vol. 27. New York: Springer-Verlag, 1978.











Computer Graphics and Woven Fabric Design

Laura Giannitrapani

Personal Computer Arts Group
Box 1954, Philadelphia, Pa. 19105

ABSTRACT

The nature of the weaving process lends itself to using computer graphics in the design stages. As a designer and hand weaver, I have used computer graphics in generating decorative and graphic designs. I intend to use the computer as a design tool in weaving. In this paper I will describe ways that small computers can assist the weaver.

Computers are used as design aids in a variety of fields both functional and aesthetic. One area that lends itself particularly well to computer assistance is woven fabric design. Both commercially mass produced fabrics and individually hand woven fabrics can be enhanced by the use of the computer in the design stage.

The structure of woven fabric is composed of two sets of interlocking threads called the warp and weft. The warp extends the entire length of a piece of fabric and is held under tension on a loom for the duration of the weaving process. Alternating warp threads are lifted and depressed by harnesses and treadles creating a shed for a weft thread wound on a shuttle to pass through. This process ranges in complexity from the simple methods (American Indian backstrap loom) to the most complex (Dobby and Jacquard looms).

The design of woven fabrics involves determining which warp threads to raise for each shed. The treadle sequence for a plain (or tabby) weave requires lifting every other warp thread for each shed. More complicated designs are achieved by varying the treadle sequence. A point by point graph is made of the design illustrating which threads will appear on the completed surface. The columns and

rows are assigned numbers. Threading and treadling sequences are generated from them.

A computer can be programmed to simulate patterns of warp sequences. The treadling sequences can be entered in any combination and the graphics monitor can display the rendered patterns. The weaver may experiment and perfect a pattern and color combinations before she/he undertakes the time consuming tasks of threading a loom and weaving.

Granted, experimenting and planning with the fiber itself reveals invaluable information about how a fabric can and will look, while the computer's rendering is presently rather plastic. Nonetheless, the computer can be used in the planning stages when the designer establishes desired constants such as texture, number of colors, and complexity of pattern. Here the computer is an invaluable aid in experimenting with the other elements and in finalizing a design. The plotter and printer can then draw and list the threading and treadling sequences.

More than accelerating the design process, the computer can help to generate designs. For example, a design sampler could be generated by giving the computer a threading sequence and the number of harnesses. The computer would generate every possible treadling sequence and render the resulting patterns. The element of color is always a variable which can be controlled either by the designer or the computer.

A unique approach to designing with the computer would be to use mathematical formulae to make harmonic transitions in designs. Given a formula, number of harnesses, warp and weft threads, and colors, the computer could generate an entire plan for a piece of fabric without

pattern repetitions as is common in conventional weaving. Naturally, the number of harnesses and treadles contribute to the complexity of a design.

Microcomputers are limited in their inability to render the unlimited variety of colors available in yarn dyes. The micro's relatively low screen resolution also inhibits accurate rendition of fabric texture. However, their low cost and ability to generate and manipulate patterns quickly and interactively makes them both accessible to and desirable for an independent handweaver. Software packages that include some of the features described above are currently available; interested weavers need not worry about having to program the computer themselves. However, after establishing familiarity with the computer, the designer may wish to explore the many possibilities available through programming.

COMPUTERS AND CHOREOGRAPHY

Cathy M. Stadler

The Spence School
22 East 91st Street
New York City, N.Y.

ABSTRACT

I am a dancer and teacher of dance who up to a year ago had no exposure to computers. In this paper I will describe where my year-old interest in using a computer to choreograph and replay dances has taken me.

My first encounter with computers was seeing a child's exhibit called Graphics Theatre running on an Atari 800 micro-computer. The program was designed to show the animation capacity of a small computer and how it could be controlled interactively. It enabled a child to manipulate the color, path and speed of three different objects, a rocket, a tree and a man. As a dancer/choreographer, I was immediately captivated by the man who appeared to be dancing. I didn't care about changing his color or speed, I wanted to change how he moved. I turned to the creator of the program and asked, "How do you change his movements? How many positions can he strike? How many objects can interact on the screen simultaneously?"

While my friend answered my questions, my mind raced on. I considered how difficult it was to express movement ideas with dance notation systems like Labanotation. While Labanotation and other dance notation systems provide the positional information of a movement, they don't show the stylistic qualities that help a choreographer visualize the spatial flow of the dance. However, here before my eyes was a means to solve these problems. The computer could help a dancer create, preserve, and animate her movement ideas.

After this experience I was hooked. I was convinced the computer could serve as

a tool to help me create dance combinations and view their spatial design. I thought that if I could learn some programming skills, I would be able to direct an army of animated dancers with a single keystroke. I even envisioned a system that would enable me to see both the positional relationships and the stylistic qualities of each movement. Or so I thought....I had lots of dreams.

I plunged in and began to learn Basic and Logo on a friend's Apple II computer. I experimented with a number of animation packages trying to create figures that could do individual dance steps and combinations. In many instances I never moved beyond creating the figure because the system couldn't accommodate the size or shape I wanted. When I did manage to create the desired image it took forever because I had to key in seven discrete body positions to compose the one movement phrase I wanted! It took an hour to create a figure leaping. After spending days designing the figure, compiling the paths, and generating the actors, I discovered that the movement could only follow a horizontal or diagonal path. And even worse, it was impossible to show a figure jumping or turning.

Discouraged by my own inability to program and frustrated by the limitations of animation packages, I turned to research. What I discovered was that together programmers and dancers at universities here and in Canada were grappling with similar issues, albeit on larger computers. I discovered computer dance programs that used keyboard equivalents of Labanotation to animate a figure. Some programs were sophisticated enough to generate an image that could articulate 21 body joints. Using this system one could program leaps, runs, and complex movements like a forearm extending or shoulders rotating. Other systems used goniometers, devices which measure angles, and transmit that movement information to the computer. The

computer could then produce a Labanotation score or an animation directly from the dancer's movement. Another system reproduced a dance from video frames grabbed at specific time intervals from a video camera.

* (Labanotation is the most thorough system for symbolically describing movements. There are notations for movements as small as a tiny hand gesture.)

While most of these systems produced an animated figure that moved in a humanlike manner, there still remained the major issues of body orientation and notation. Many of the systems generated a turning figure but one couldn't tell whether the shape was turning clockwise or counter clockwise. Nor was it clear whether the image was moving forward or backward.

None of the systems I investigated could completely serve my needs. There were limitations with each either in the cost or the design. I didn't see myself buying a goniometer any time soon and I needed a system that could let me create a dance that included turning movements.

So after months working on the computer what had I achieved? I still hadn't found an easy way to choreograph a dance. What the research taught me was that every system had its limitations. But most importantly, the time I spent working made me realize that the computer could operate as a tool to serve my artistic ends only if I knew what I wanted. The months of work forced me to take my dreams and translate them into ideas that could be adapted to the capabilities of a computer I could afford.

A software package integrating the following eight features would suit my needs when choreographing a dance. While I believe that such a package could be built on a microcomputer, I also realize now that the programming skills involved would be far in excess of my own.

Draw Mode: For drawing a series of two dimensional stick figures directly on the screen. These figures would be composed of six or seven lines to represent the relative position of the arms, legs, head, and torso as they exist in an individual dance pose. A carrot shaped line could be used for the head and when pointed either left or right would allow me to see the direction of the movement.

Save Screen: I would want to draw a specific number of figures on one screen to constitute one movement phrase. This screen could be saved and recalled

subsequently for viewing and editing as needed.

Interpolation Function: When I draw a dance phrase I use a shorthand notation of my own. I don't draw every discrete body change. I draw a sequence of gross positional stances (see Figure 1). The interpolation function would encode the relative limb and body positions of the figures on the screen and fill in the interval movements to create a humanlike animated movement that would complete the movement phrase.



Color Function: A means to use color to determine the body direction, focus and orientation. This would enable me to use a two dimensional screen and still be able to work with angular and turning movements. For example, a movement phrase such as a pirouette would be represented by displaying the initial forward stance in white, the side view of the leg lifted in red, and the ending stance with back faced upstage in green.

Repeat Function: A means to create repetitive movements or positions on the screen without having to draw them each time.

Path Drawing: Once a movement phrase were drawn, it would be essential to establish its spatial and directional design. The ability to draw a path along which the movement would be executed would satisfy this need.

Edit, Grab and Insert Functions: I would need to be able to take a movement phrase and either edit a segment of it or pick up a segment and place it in another part of a phrase. This would allow me to easily resequence the movement phrases.

Animating Sequences: I would need a means of specifying and sequencing the movement phrases in order to create the full dance. When this ordered sequence was recalled from the disk, I would see the dance performed.

If all of these features were integrated in a single program, I could draw a figure, create the steps, experiment with their order and direction, animate them, view the steps and their spatial design, and finally hook them all together to create a movement score.

This system would reflect my own individual style of choreographing. However, I feel that these are the basic elements any dancer would need to create and view a dance. Once given this, a dancer could watch a movement score without the choreographer being present. The interpretation of the movements would come later, after the dancer had learned the work. If such a package were fully designed and programmed, it would provide a system that preserves movement ideas as well as animating them on the screen for others to learn from and adapt.

DESIGNING A RECURSIVE FRAMEWORK FOR EVOLUTIVE POETRY

by Ned J. Davison

University of Utah, Salt Lake City

Abstract

The personal computer offers new solutions to the sequential limitations imposed on creative writers by traditional printed forms. Temporal experiences can now be manipulated, re-ordered, and juxtaposed in new and endlessly varied combinations. This essay explores the literary implications of the new creative freedom and suggests ways in which the computer may expand our literary imagination in the future.

Since the first time a text was cut into segments and the resulting pages broke the sequential tyranny of the papyrus roll, writers have searched for greater temporal freedoms. Late nineteenth-century experiments with the displacement of action and the spacial disposition of words, phrases, and letters brought critical incomprehension and cries of "degeneracy" and "decadence." The modern artist's sensibility was no longer served by the orderly forms of the past. His world view was indeed "dis-integrating," and new technical needs arose. Mallarmé, Joyce, Dos Passos, Jorge Luis Borges, Julio Cortázar, or a William Burroughs, all inherited the marvellous flexibility of the separate leaves of a book, susceptible to instant leaps from middle to back to beginning, but they were still obliged to struggle against the confinement of the imprisoned pages and the unalterable printed letters.

The personal computer offers new solutions, freedoms, and perhaps even new forms. Here are some musings on one of them.

I recall reading, I believe in Creative Computing, of an Argentine poet who was working on a program that would allow the user to make literary decisions. Since then, notations of narrative or filmic forms that invite the spectator to choose the ending or simulate open-ended dialogue have proliferated. Recent experiments with adapting the video-disc to new game-development are, in a way, a variant of the same impulse to add new dimensions to our play and aesthetic activities. In one way or another these are all manifestations of the evolution of a participatory art in which the artist and the

spectator, or the author and the reader, are merged and share the creative act. In a curious sense this evolution, or quest of modern artists, prefigures in aesthetic terms the emergence of Alvin Toffler's economic "prosumer," the social archetype of the high-tech society. (The parallel is not fortuitous, and further speculation on the socio-psychological implications of the relationship might be worthy of consideration in some future essay.)

At any rate, it is very possible that recollection of the arrival of the computer in the arts might arouse, in coming generations of writers, a sense of appreciation analogous to what others must have felt when first provided with the freedom of separate manuscript leaves bound together in easily managed pages.

If we limit ourselves to only letters, words, and phrases--leaving aside for the moment the bewildering promise of graphics and sound--we can quickly see that the almost instantaneous variation of sequence that pages provide is extended by the computer to the simplest element of written language, the printed or displayed alpha or numeric character. Passing from the letter to the syllable, word, phrase, line, stanza, or paragraph is a simple and natural progression in the liberation of verbal art from the bonds of fixed sequence.

Future literature that takes advantage of this exceptional feature of the computer is very difficult to envision with any clarity. It will be the artist who will uncover the new forms, though certain characteristics are already discernable. One is the obvious realization that all materials of composition, potential structures, and forms, will be stored in a format that is essentially latent and unfixed. Latency in art has always existed, of course, but in a simpler way--the book, dormant, waiting to be opened and the action and characters put in motion; the film, on the point of being projected; the record or tape, ready to be played. Even the sonnet awaits a reader to put its structure in motion, like a mobile anticipating the touch of a finger or a thrust of air to bring its planes and light to life. All of these, in their way, share a latent expressivity. But the electronic file holds ready a sensory pattern of experiences, feelings, and sensations that are uniquely unfixed and

mutable. The pathways that weave them together offer ever-renewable roads and combinations. Each presentation, or playing, or reading of the electronic work may be made to be inevitably individual, unique, and not repeatable. The spiritual variety that we feel with each reading of a traditional work is extended and embellished with the variety and unpredictability of the computer driven work. The creative management of traditional ingredients, time, characters, place--or physical, emotional and sensorial environment--is made more flexible and complex with the power to "program" chance, to invite it in and then put limits and borders on it. Random occurrence is given an opportunity to entwine itself with the will of the author and to combat his choices. The popular idea of the poet/god, the magus, takes on new meaning with a fictional world of such structured chance, guided but unpredictable even to its maker.

An example is in order. Let's take the archetypal notion of a "trip." It could be a voyage through space, that is to say, geographical, or through our bodily perceptions (sensory). The basic structure is the same in either case--a matrix of meeting places like a Borgean network of spacial and temporal junctures. The links are varied in form and intensities. As in a railroad network, some lead to dense nuclei out of which spread new routes that terminate in "minor" modules. The switchman is a "double," a multiple personality. One part is the programmer/author who creates the nuclei and modules--as a formal act of structuring and composition--and who fills those modules with life experiences, metaphors and images. The other half of this "double" is, naturally, the reader/spectator. Using a controller, a keyboard, a stick, oral commands, or other means, the "reader" chooses, along the journey, a direction, a turn, or a destination. Let's imagine, in order to make things more concrete, a trip through Europe, by train.

The matrix, or cohesive framework, is a stylized map, a blueprint, expressed in terms of the cardinal points of the compass. Pushing the stick forward sends us northward. The author may have dropped us, at the beginning, in a predetermined city (nucleus) or have left the locale to chance, according to his creative instincts, or we might have been given a choice. All possibilities can be latent within the microEurope of the program branches.

We leave from, say Paris, and the screen and air fill with shapes, colors, and sounds psychically correlative to the French-Belgian countryside. (Pictures if you wish. I'd prefer visual metaphors, abstractions.) Then there appear, one after another, phrases, free images, combined randomly out of a limited set of fixed and carefully conceived lines that share a common

emotional and thematic base.* When we reach a nucleus/city, we elect to remain or not. We choose a place or an atmosphere--a cafe, club, museum, pension. With each choice, visual images and sounds attend the decision, accompanied by a flow of poetic lines appropriate to the module chosen. They play on, unfolding, turning back on themselves in continually varying repetitions until we decide to move on, and we continue our journey toward some new geographic or affective module. Every so often, chance makes us take the wrong train, and then we awaken surprised, in an unexpected world. The profile of such a poetic network might suggest, perhaps, something of the angular and cosmic geometry of Borgean chance.

This outline is not entirely imaginary. Computer-generated music and graphic arts are commonplace and already part of the history of our times. The verbal arts, however, are still waiting in the wings. Early and generally unsuccessful efforts to produce poetry generated by computer led us somewhat astray and inclined us to overlook the real possibilities. With the arrival of the personal computer, writers now have at their complete and exclusive discretion a tool that combines, on one level, pen, brush, canvas and tape, and on another much broader scale, pages/book, screen/sound and the promise of their varied and perpetual regeneration.

VERSE SET by Alan Davison

100 ? "the room is full of smoke":GOTO 23
 101 ? "a policeman walks back and forth":GOTO 23
 102 ? "two drunks sleep in the corner":GOTO 23
 103 ? "stale bread for lunch":GOTO 23
 104 ? "the smell of oil":GOTO 23
 105 ? "four days without a bath":GOTO 23
 106 ? "you look around.":GOTO 23
 107 ? "a Frenchman cleans his nails":GOTO 23
 108 ? "a Soanlard lights a cigarette":GOTO 23
 109 ? "stomach growls":GOTO 23
 110 ? "you have gas":GOTO 23
 111 ? "three hours til the next train":GOTO 23
 112 ? "you look at your watch":GOTO 23
 113 ? "a lady enters with four kids":GOTO 23
 114 ? "the benches are full":GOTO 23
 115 ? "you try to sleep":GOTO 23
 116 ? "you rearrange your backpack":GOTO 23
 117 ? "you sit down on the floor":GOTO 23
 118 ? "the loudspeaker rattles":GOTO 23
 119 ? "Tiene usted la hora?":GOTO 23
 120 ? "the man next to you asks a question":GOTO 23
 121 ? "close your eyes":GOTO 23
 122 ? "you shrug and look away":GOTO 23
 123 ? "Tiene usted la hora?":GOTO 23
 124 ? "a cigarette butt lands on the floor":GOTO 23
 125 ? "you write in your journal":GOTO 23
 126 ? "you think about home":GOTO 23
 127 ? "been waiting two hours":GOTO 23
 128 ? "So this is Europe?":GOTO 23

* See my "Verse Weaving . . ." in the July '82 issue of *Creative Computing*, pp. 166-172, as an example of an affective module, and also the sample verse-set and randomly generated stanzas at the end of this essay.

129 ? "" :GOTO 23
 130 ? "" :GOTO 23
 131 ? "" :GOTO 23
 132 ? "" :GOTO 23
 133 ? "you look at your watch":GOTO 23
 134 ? "time passes":GOTO 23
 135 ? "time passes":GOTO 23
 136 ? "an ant crawls on your shoe":GOTO 23
 137 ? "indifferent":GOTO 23
 138 ? "tired":GOTO 23
 139 ? "slowly":GOTO 23
 140 ? "uncomfortable":GOTO 23
 141 ? "without thinking":GOTO 23
 142 ? "self conscious":GOTO 23
 143 ? "you wait...":GOTO 23
 144 ? "the loudspeaker rattles":GOTO 23
 145 ? "nothing changes":GOTO 23
 146 ? "" :GOTO 23
 147 ? "" :GOTO 23
 148 ? "Quelle heure est-il?":GOTO 23
 149 ? "an old man coughs":GOTO 23
 150 ? "a beggar approaches":GOTO 23
 151 ? "you pretend not to notice":GOTO 23
 152 ? "voices...":GOTO 23
 153 ? "feverish":GOTO 23
 154 ? "you yawn":GOTO 23

time passes
 you sit down on the floor
 been waiting two hours,
 stomach growls

a Spaniard lights a cigarette
 the benches are full
 slowly
 you look at your watch
 indifferent
 you sit down on the floor
 the room is full of smoke
 the man next to you asks a question
 the loudspeaker rattles
 voices...
 indifferent
 you wait...

stale bread for lunch
 the loudspeaker rattles
 you think about home
 you look at your watch

 been waiting two hours,
 voices...
 indifferent
 you shrug and look away
 you rearrange your backpack
 you have gas
 the room is full of smoke
 three hours til the next train
 you wait...
 indifferent
 the loudspeaker rattles
 a cigarette butt lands on the floor

you look around.
 the benches are full
 slowly
 stomach growls
 indifferent

an old man coughs
 you look around.
 'So this is Europe?'
 you rearrange your backpack
 the room is full of smoke
 the loudspeaker rattles
 the benches are full
 the room is full of smoke
 stomach growls

stomach growls

feverish
 time passes
 a policeman walks back and forth
 the man next to you asks a question
 you look around.
 'Quelle heure est-il?'
 self conscious
 you look at your watch
 two drunks sleep in the corner
 the room is full of smoke
 you try to sleep
 you shrug and look away
 you look at your watch
 a lady enters with four kids
 time passes
 stomach growls
 you wait...

LASER SHOWS AND THE USE OF COMPUTERS

by Alan Jackson

Buffalo State Planetarium, Buffalo, New York

Abstract

Laser shows are described as they have developed over the last 20 years from the invention of the laser to present day machines. They may evolve to incorporate video projectors and their use of computers will be greatly expanded.

Laser shows came about because of the special properties of lasers. Before the laser, there was no comparable medium which could create an image as fluid or dynamic live and in real time. With the arrival of the microcomputer, these images could convey more precise and complex information according to artists' ideas.

An artist strives for control of his or her medium. The computer provides the graphics artist with an extension of his or her own mental process that allows control over a much greater amount of information than would be possible if each image were individually made by hand.

For a laser show, the artist gives a live performance drawing patterns according to his or her interpretation of the music that accompanies the show.

The beam of a laser is useful because it is so concentrated and does not diverge as it passes through the air. A laser (Light Amplification by Stimulated Emitted Radiation) consists of a partially reflecting and a fully reflecting mirror which are exactly parallel, with a tube filled with an inert gas mounted between them. Atoms of the gas may be in various states of excitation depending on what orbital level the electrons of the atoms may take. When an electron absorbs energy, it jumps to a higher energy level. When an electron jumps from a high to a lower energy level, due to a collision with a photon, it gives up energy in the

form of a photon. That Emitted photon plus the original photon are an Amplification of the original photon. Light that is exactly perpendicular to the mirrors is reflected many times and each reflection is amplified. A small amount of this light passes through the partially reflecting mirror as a pencil thin laser beam.

The invention of the laser supplied the world with the solution to a "thousand unasked questions," although the laser soon found applications in many places in research, industry and entertainment. The first lasers shows were crude; usually the laser projector was a mirror glued to a piece of balloon that was attached to a loud speaker. When music was played the mirror would vibrate randomly. A laser beam reflected by the mirror onto a screen created a pattern that would pulsate to the beat of the music.

The next innovation was the use of scanners specifically made to deflect a laser beam. A scanner is made of a small piece of iron and a mirror attached to a shaft mounted on instrument quality bearings. The iron is made to rotate by coils of wire surrounding it and is spring loaded so that it returns to a center position when no current is applied. A small variable capacitor mounted on the shaft feeds back the position of the mirror to a controller circuit.

The scanner made possible the sophisticated laser shows that first appeared in the early seventies. These shows use a synthesizer similar to a music synthesizer to create various frequencies and waveforms. These signals are fed to pairs of scanners, the mirror of the first scanner deflecting the beam horizontally and the second scanner deflecting the beam vertically. The shape of the patterns produced will depend on the two signals given respectively to each scanner and the relationship of the signals to each other. The arrangement is an electromechanical version of an oscilloscope set up to produce Lissajous figures. For example, two sine waves which are of the same phase and amplitude will cause the scanners to trace a straight line. If they

drift out of phase, the pattern will seem to be a slanted circle that slowly rotates.

The major laser shows incorporate a wide variety of color and special effects. Most use an Argon-Krypton laser as a light source. The beam of this laser is split into different wavelengths by prism. Each of the most powerful of the resulting beams (usually green, blue, yellow and red) is sent through a modulator and then to a pair of scanners. The modulator is able to turn the laser beam on and off at high speed to achieve a "spaghetti effect" in the patterns. It can also allow the scanners to draw separate patterns without a connecting trace.

The shows are controlled live by an operator or by a multitrack tape deck. Pre-recording a show allows a tape to be made at a remote computer graphics facility where simple words and figures can be composed at a terminal. Now that the cost of computers is coming down, some shows are using them as part of the live performance. Using the available input devices, such as joysticks and bit pads, the artist can draw words and figures quickly before a show or generate abstract patterns during a show which are not possible using a synthesizer.

The computer generates patterns as a series of one to several hundred points connected by straight lines. It uses a stored program plus data supplied to it through joysticks and switches to calculate the coordinates of each point in the pattern.

After the coordinates of each point in the pattern is calculated they are sent to an external buffer memory as successful 8 bit bytes, each (x,y) coordinate pair occupying two bytes of memory. Data in the buffer memory is presented to digital to analog converters and the output of the D/A's is sent to the scanners through the scanner controller amplifiers. In order to relieve the computer of excessive I/O data handling, the buffer continuously cycles through to present the scanners with the same pattern thirty times per second. At this speed, the human eye cannot detect a single moving point of laser light but sees only a non-flickering continuous pattern.

Although computer controlled laser shows are in their infancy, they have potentially an enormous advantage over hardwired devices because creating patterns is as simple as supplying data to a program.

Future advances will allow television images to be presented to large audiences as easily as films. Although relatively inexpen-

sive conventional light source television projectors are readily available, their image size and quality is too limited to be truly useful. High quality conventional and laser light source projectors exist but are prohibitively expensive. With the rapid development taking place, it seems only a matter of a few years before an economical high quality projector comes into common usage.

A system with great promise is the large area display which uses a laser beam to create a raster similar to the raster created by the electron beam in a TV. The beam from an Argon-Krypton laser is split into the four primary colors, passed through modulators and recombined into a single beam. That beam is scanned in the horizontal direction by an octagonal mirror rotating at one hundred thousand rpm on an air bearing. The vertical is provided by another slower mirror whose axis of rotation is perpendicular to the first. It seems reasonable to assume that television images with quality comparable to present day film will be possible with such a machine.

Right around the corner may follow the showing of prerecorded tapes or live presentations with special effects not possible in film. The actor(s)/artist(s) may act out scenes and then through the use of video processing the images may be changed for greater effect.

Chroma-keying is one popular means of enhancing video. In one scene, a particular color (usually blue) is the key. Then another video image can be substituted wherever blue appears in that scene. For example, the weather map on the six o'clock news are viewed by separate cameras. The forecaster stands in front of a blank blue background and the map is superimposed on the blue by a videoprocessor. This allows any map to be shown at will including moving satellite films which would otherwise have to be shown behind the forecaster using rear screen projection.

The movie Tron made extensive use of Chroma-keying. The actors were filmed on blank white stages and the rest of the scenery was filled in by computer graphics. The graphics in Tron were generated one frame at a time each composed of well over 10 million pixels and sometimes taking hours to finish; for comparison, a TV frame contains only a quarter million pixels.

As computers become faster, it will be possible to generate these graphics in "real time".

Plays could be presented with instantaneous set changes. Scenery could move about the stage. Or, using a variation on chroma-keying, a computer could be programmed to

respond to a person's motions. It would search a scene for "Key"color or "key" shape such as a person's hand. It would extrapolate geometric data from the object and use it to generate figures. The actor/artist could drive a car, step off a plane, perform magician's tricks, move an animated puppet, all by the "wave of the hand". The actor/artist will be able to spontaneously communicate with an audience, through a visual medium made possible by computer.

Author Index

ASHCRAFT, A.C.	5	KEITH, M.	29
COVITZ, F.H.	5	KOLOMYJEC, W.J.	39
DAVISON, N.J.	111	LORD, R.H.	1
DEWIT, T.	61	MAUCHLY, J.W.	23
DEWITT, T.	61	NAHAS, M.	95
DIETRICH, F.	71	RICHMOND, W.	87
EDELSTEIN, P.	61	RUBIN, S.	87
FLYNN, A.C.	35	SHORTESS, G.K.	75
GIANNITRAPANI, L.	105	SPIEGEL, L.	15
HUITRIC, H.	95	SPROUL, K.A.	5
JACKSON, A.	115	STADLER, C.M.	107
JACOBSON, J.	47	WRIGHT, W.	79

REPORT

1	1000	1000	1000
2	1000	1000	1000
3	1000	1000	1000
4	1000	1000	1000
5	1000	1000	1000
6	1000	1000	1000
7	1000	1000	1000
8	1000	1000	1000
9	1000	1000	1000
10	1000	1000	1000
11	1000	1000	1000
12	1000	1000	1000
13	1000	1000	1000
14	1000	1000	1000
15	1000	1000	1000
16	1000	1000	1000
17	1000	1000	1000
18	1000	1000	1000
19	1000	1000	1000
20	1000	1000	1000
21	1000	1000	1000
22	1000	1000	1000
23	1000	1000	1000
24	1000	1000	1000
25	1000	1000	1000
26	1000	1000	1000
27	1000	1000	1000
28	1000	1000	1000
29	1000	1000	1000
30	1000	1000	1000
31	1000	1000	1000
32	1000	1000	1000
33	1000	1000	1000
34	1000	1000	1000
35	1000	1000	1000
36	1000	1000	1000
37	1000	1000	1000
38	1000	1000	1000
39	1000	1000	1000
40	1000	1000	1000
41	1000	1000	1000
42	1000	1000	1000
43	1000	1000	1000
44	1000	1000	1000
45	1000	1000	1000
46	1000	1000	1000
47	1000	1000	1000
48	1000	1000	1000
49	1000	1000	1000
50	1000	1000	1000
51	1000	1000	1000
52	1000	1000	1000
53	1000	1000	1000
54	1000	1000	1000
55	1000	1000	1000
56	1000	1000	1000
57	1000	1000	1000
58	1000	1000	1000
59	1000	1000	1000
60	1000	1000	1000
61	1000	1000	1000
62	1000	1000	1000
63	1000	1000	1000
64	1000	1000	1000
65	1000	1000	1000
66	1000	1000	1000
67	1000	1000	1000
68	1000	1000	1000
69	1000	1000	1000
70	1000	1000	1000
71	1000	1000	1000
72	1000	1000	1000
73	1000	1000	1000
74	1000	1000	1000
75	1000	1000	1000
76	1000	1000	1000
77	1000	1000	1000
78	1000	1000	1000
79	1000	1000	1000
80	1000	1000	1000
81	1000	1000	1000
82	1000	1000	1000
83	1000	1000	1000
84	1000	1000	1000
85	1000	1000	1000
86	1000	1000	1000
87	1000	1000	1000
88	1000	1000	1000
89	1000	1000	1000
90	1000	1000	1000
91	1000	1000	1000
92	1000	1000	1000
93	1000	1000	1000
94	1000	1000	1000
95	1000	1000	1000
96	1000	1000	1000
97	1000	1000	1000
98	1000	1000	1000
99	1000	1000	1000
100	1000	1000	1000

PROCEEDINGS

**symposium on small
computers in the arts**

IEEE CATALOG NO. 82CH1831-7
LIBRARY OF CONGRESS NO. 81-84728
IEEE COMPUTER SOCIETY CATALOG NO. 455

COMPUTER
SOCIETY
PRESS