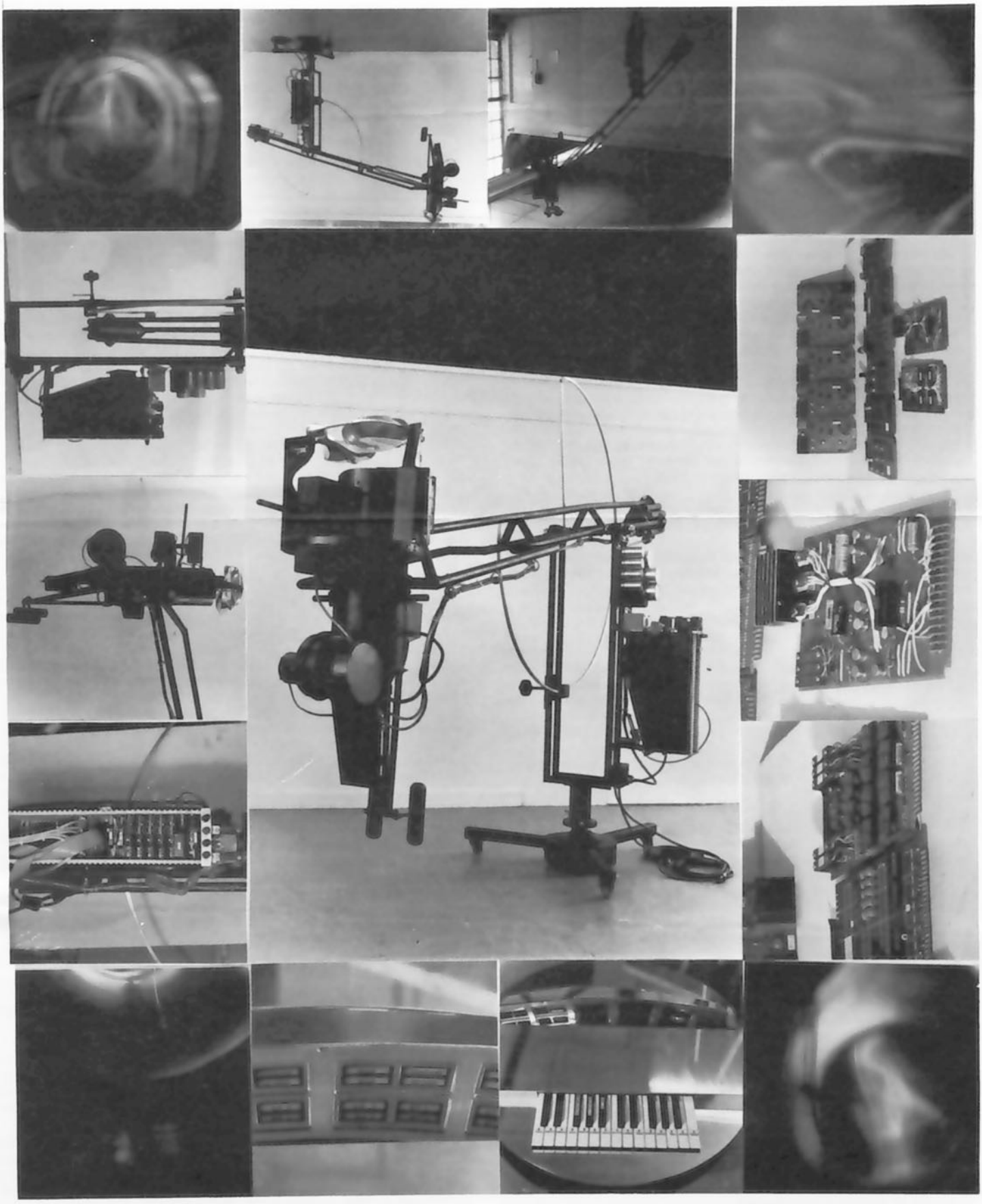
PAGE 36



OCLES UPHOFF

MINI-EXPLOR
A FORTRAN-Coded Version of the EXPLOR Language for Minicomputers

by
Ken Knowlton
Bell Laboratories
Murray Hill, New Jersey 07974

I. INTRODUCTION

The previously reported EXPLOR language generates two-dimensional patterns, designs and pictures from **Ex**plicitly provided 2-D **P**atterns, **L**ocal **O**perations and **R**andomness. It has proven effective in depicting results of simulations in natural (i.e., crystal growth) and hypothetical (e.g., cellular automata) situations, and for the production of a wide variety of designs. The language has been particularly valuable as a teaching aid for new students in computer graphics and computation in general — I have used it in courses and workshops at the University of California at Santa Cruz, Eastern Michigan University and Syracuse University; the "Santa Cruz" version has been exported to 30 other colleges and universities.

Mini-Explor is an abbreviated but powerful version of the system, coded in only 430 lines of that portable subset of American Standard FORTRAN called PFORT. It can run on most minicomputers having a 16-bit word length (or larger) and 8K to 16K of core storage and, of course, a FORTRAN compiler. Portability has been checked by the PFORT Verifier. Unless otherwise modified, output is by means of WRITE statements which cause up to 3-times-overprinted output on the machine's line printer or teletype — yielding 4 effective shades of grey scale. The internal image is retained in raster-scan format and consists of 140 lines of 140 spots each, packed as seven 2-bit picture cells per machine word. Most operations involve unpacking and repacking of sections of the total image; an implementer may find it desirable to recode in machine language the low level routines which perform these tasks. Other implementation hints appear in Appendix B.

II. PROGRAMMER'S DESCRIPTION OF MINI-EXPLOR

From the programmer's point of view, the system consists of these nine FORTRAN-callable functions and subroutines, subsequently described in detail:

FUNCTIONS

```
NUM     (x,y)
NE      (min,max)
```

SUBROUTINES

```
CALL SHØW   (x,y,w,h)
CALL PUT    (x,y,   n)
CALL PUT4   (x,y,   n)
CALL PUT16  )x,y,   n1,n2,n3,n4)
CALL CHANJ  (x,y,w,h,%                          rule)
CALL LØCØP  (x,y,w,h,%, many,nabors,these, rule)
CALL CØMBN  (x,y,w,h,%, xf,yf,orientation, ro,r1,r2,r3)
```

To use these routines effectively, the user needs to learn that part of FORTRAN dealing with the following:

- subroutine calls
- GØ TØ's
- integer variables

arithmetic assignment statements; operators + - * / **

- DØ loops
- logical IF statements; connectives .AND. .ØR. .NØT.
- functions, including built-in FORTRAN functions MINO, MAXO, MØD, IABS, ISIGN
- arrays: 1, 2, and 3-dimensional

All parameters used in subroutine calls are **integers** (i.e., with values 0,1,2,3,4 . . .); in most cases where it seems meaningful, their values may be negative. It is thus advisable to make the first line of every program, if accepted by the local FORTRAN compiler, the following:

IMPLICIT INTEGER (A-Z)

The user imagines the internally-storred picture as a 140x140 array of picture cells, each holding a digit 0,1,2, or 3 and addressed in terms of their x,y coordinates.

At the beginning of a program, all cells are filled with zeros. In the following discussion, subroutines dealing with rectangular areas have in their descriptions the dummy parameters

(x,y,w,h,%, . . .)

which have these meanings:

x is the x-coordinate of the center of the rectangle (or ½ cell left of center if width is an even number)

y is the y-coordinate of the center of the rectangle (or ½ cell below the center if height is even)

w is its width

h is its height, and

% is an integer 1 to 100 stating approximately the percentage of cells actually to be treated, on a psuedorandom basis. (100 means process all of the cells for certain.)

FUNCTIONS

NUM (x,y) has a value (0-3) of the **num**ber currently stored in cell x,y; if (x,y) is off of the internally represented surface, the value of NUM (x,y) is 4.

NE (min,max), pronounced "any," has, on each usage, a new randomly-selected value from **min** thru **max**; max may be less than min and either or both may be negative but the difference [max-min] must be less than 199.

SUBROUTINES

CALL SHØW (x,y,w,h) will cause a printout showing the contents of the specified rectangle. The specified area will be truncated if it exceeds the area actually represented in the machine or if it is too wide for the device used for output. Digits 0, 1, 2, 3 appear as a grey scale: , ', x, and X, respectively.

CALL PUT (x,y, n) will put at coordinates (x,y) the number n (i.e., overwrite the previous contents). If n is larger than 3, the cell remains unchanged.

CALL PUT4 (x,y, n) where n is a 4-digit number will cause the left-most digit to be "put" at (x,y), the next to be put at (x + 1,y) etc. If any digit is larger than 3, the corresponding cell is not changed.

CALL PUT16 (x,y, n1,n2,n3,n4) where n1 through n4 are each 4-digit numbers, "puts" or writes the 16 digits into 16 successive cells (x,y), (x + 1,y) etc. Note that a series of calls to PUT16 with decreasing y values can serve to place an explicit 2-dimensional pattern onto the internal grid:

```
CALL PUT16 (50,40, 3311, 3333, 3311, 3311)
CALL PUT16 (50,39, 3311, 3311, 3311, 3311)
CALL PUT16 (50,38, 3311, 3311, 3311, 3311)
CALL PUT16 (50,37, 3333, 3311, 3333, 3311)
```

CALL CHANJ (x,y,w,h,%, rule) pronounced "change" — causes the contents of the specified rectangular area to be changed according to the specified rule: rule is a 4-digit number saying, rom left to right what the digits 0, 1, 2 and 3 are to be changed into. Thus the rule 1033 says that **0's** become 1's, **1's** are to become 0's, **2's** become 3's, and **3's** remain unchanged (become 3's).

CALL LØCØP (x,y,w,h,%, OK-counts, neighbors, these, rule) is a local operation, causing certain of the cells in the specified region to be changed according to the indicated **rule**: Those changed are the ones with acceptable counts of the designated adjacent cells holding appropriate numbers:

OK-counts indicates up to 4 permissible numbers of neighbors which, satisfying the test, permit the cell to be changed by the rule. If zero is a permissible count, it must be last.

neighbors is a 3-digit number specifying a set of neighbors, made up by summing the corresponding numbers from this chart:

400	200	100
40		10
4	2	1

these are up to 4 values that individual neighbors must have to satisfy the test. If zero is one of them, it must be last.

For Example:

CALL LØCØP (x,y,w,h,50, 350,707,120, rule) says "in the area **x,y,w,h**, change, according to the given **rule**, half (50%) of those cells where 3, 5, or none of the following six neighbors: diagonally adjacent cells (400 + 100 + 4 + 1) plus cells directly above and below (+ 200 + 2 = 707) contains 1's, 2's, or 0's."

The routine works in such a way that effects do not propagate during a single call. For example, a single layer of 3's could be placed around existing 3's without producing unlimited streamers of 3's. The routine uniformly does **not** process cells on the edge of the represented area, regardless of the neighborhood specified.

CALL CØMBN (x,y,w,h,%, xf,yf,orient, r0,r1,r2,r3) read "combine" — causes contents of the indicated percentage of x,y,w,h to be changed by one of four rules, depending on contents of a corresponding cell in an area centered at (xf,yf). The result is thus a simple or complicated "combination" of two picture area, and is imagined to come about as follows: a copy of the neighborhood of the "from" area centered at (xf,yf) is picked up, (re) oriented according to the value 1-8 or **orient**:

1 as is
2 rotate 90° clockwise
3 rotate 180°
4 rotate 90° counterclockwise
5 flip right-left
6 flip r-l and rotate 90° clockwise
7 flip r-l and rotate 180°
8 flip r-l and rotate 90° counterclockwise

and repositioned so that the central (xf,yf) cell of the "from" area is over (x,y) — the center of the area to be changed. Each affected cell is then processed by one of four translation rules: which rule is determined cell-by-cell by the contents of the "from area cell" directly over it. Of the many possible sets of four translation rules, four examples are here given:

		0	1	2	3 ← "from" area cell contents
(a)	. . . ,	0000,	1111,	2222,	3333)
(b)	. . . ,	0123,	1123,	2223,	3333)
(c)	. . . ,	0123,	1111,	2222,	3333)
(d)	. . . ,	0123,	1230,	2301,	3012)

In example (a) if a cell to be processed has a 0 above it (in the corresponding "from-area" cell) then whether it be a 0, 1, 2 or 3 it is changed to a 0; likewise if there is a 1 above it, it becomes a 1 regardless of what it was, etc. - the total effect of rule set (a) is that it is a **copy** operation in which, if the associated probability is 100, a copy of the "from" area replaces the "to" area.

In example (b) the larger of the two cell contents remains after the operation — e.g., 0's remain only where there were o's in both "from" and "to" cells, 3's result if either was a 3, etc. In (c) the "from" area is a pattern copied into x,y,w,h except that 0 is a "don't copy" number — i.e., where there are 0's in the from area, the original x,y,w,h contents remain. Example (d) leaves the sum, mod 4, of the two cell contents.

In instances where the "from" cell is off the represented surface, no action is taken for the corresponding "to" cell. If "from" and "to" areas overlap, let the programmer beware of undesired effects resulting from the order in which the subroutine treats the cells! The order is: leftmost column of affected area first, from bottom to top.

The six photographs enclosed in this issue of PAGE are produced by very simple programs and represent a variety of possible graphic results using MINI-EXPLOR: (1) Game of Life, Starting with Pi; (2) Contour Plot of Superimposed Octagonal Pyramids; (3) Contour Plot of a Mathematical Function; (4) Hemisphere of Small Cubes) (5) Growth of Nuclei by Randomly Selected Rules; (6) Randomly Oriented Juxtaposed Modules.

ISMUS
Iowa State Computerized Music System

Stefan M. Silverston, Computer Science
Terry A. Smay, Electrical Engineering
Gary C. White, Music

Iowa State University
Ames, Iowa 50010

THE PROJECTED SYSTEM

ISMUS will be a studio for the generation and processing of sound which incorporated digital and analog equipment. The system will be flexible and sophisticated enough to be useful for serious compositional activity as well as demonstration of and research into the properties of sound.

The system will contain a PDP11-20 computer with 8K or more of core plus disk storage. Magnetic and paper tape will be available for permanent storage of software and finished compositions when the system is completed. Communication with PDP11 will be through a teletype of CRT terminal, A Buchla Model 218 keyboard, and other manual controllers. Analog signal generation and processing will be accomplished by a Buchla series 200 modular electronic music studio. The PDP11 will communicate with the analog studio through a switching matrix which allows the computer to control the interconnection of the analog equipment, and a digital-to-analog processing system which generates control voltages for the Buchla studio. The output of the Buchla system may be monitored using headphones or speakers and recorded on tape.

When completed, the system will allow the composer to control the sofeware in the computer with an input code, with manual controllers (keyboards, switches, etc.) or with a combination of these. The user will be able to move smoothly from one type of control to another. Interconnection of modules (patching) will be possible either by hand (patch cords) or by computer control. Control voltages will be producible by manual controllers and by computer output. The user will be able to apply combinations of these control voltages to any input in the studio. Timing of events will be controlled either manually (performance) or by a clock resident in the interface. The real power of this system lies in the flexible combinations of manual and automatic control which permit optimal collaboration between the composer and the system.

ISMUS OPERATION AND USER INTERACTION

ISMUS is, in essence, a system for the control of electronic sound-generating modules. The primitive element of a composition is an **instruction** to control a sound-generating module. The user communicates with the ISMUS system via a teletype and a Buchla digital keyboard.

The basic structure in ISMUS is the layer. The layer is a list of instructions assembled by the ISMUS compiler, each consisting of a delta time (increment on time), an address and two values. At the time indicated, the values are sent to the address indicated.

A layer may be assembled from any combination of several inputs. In performance the output which plays the synthesizer is the combination of one or more layers. All layers may have been previously stored. Or, one of the layers may come from real-time (performance) input. In the case of a real-time input, the operator has the option of storing this input if he desires.

TELETYPE INPUT

An **ISMUS Language** has been developed which consists of commands to control the system. Commands are communicated to the ISMUS system via teletype. These commands either generate instructions, later to be transmitted to the synthesizer, or perform auxiliary tasks.

Examples of commands are: PATCH, which sets up interconnections among the sound-generating modules; LIST, which provides listings of stored compositions; DEFINE, which establishes the musical meaning of the Buchla keyboard; INSERT, which facilitates editing of compositions; TUNE, which allows the user to "tune" the synthesizer to obtain the desired sound (and records in the computer the final configuration reached); SCORE, which allows for coded input either from the teletype keyboard or from an external source; and PLAY, which performs a composition on the synthesizer.

KEYBOARD INPUT

Each time a status change occurs on the keyboard (either depression or release of a key), a corresponding layer (see above) is invoked. Which layer is invoked for a particular key depends upon the keyboard definition set by the user via the DEFINE command (see above). A standard default keyboard is also available.

Whenever a key is depressed or released, the event is recorded in the computer. Sequences of keyboard events are thus available for later playback, editing, and inclusion in layers and compositions.

HARDWARE

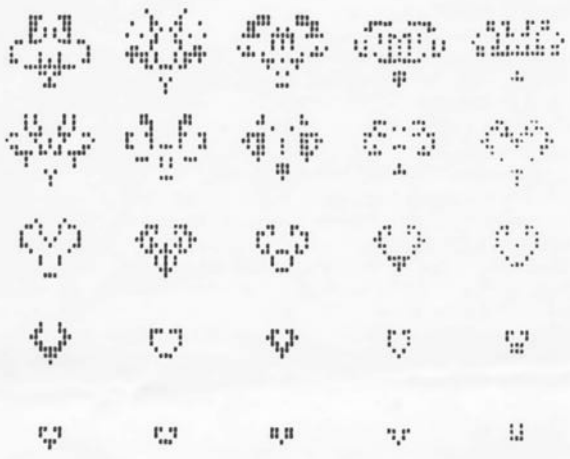
A block diagram of the hardware configuration is shown in Figure 1 on the next page.

THE BUCHLA SYNTHESIZER AND KEYBOARD

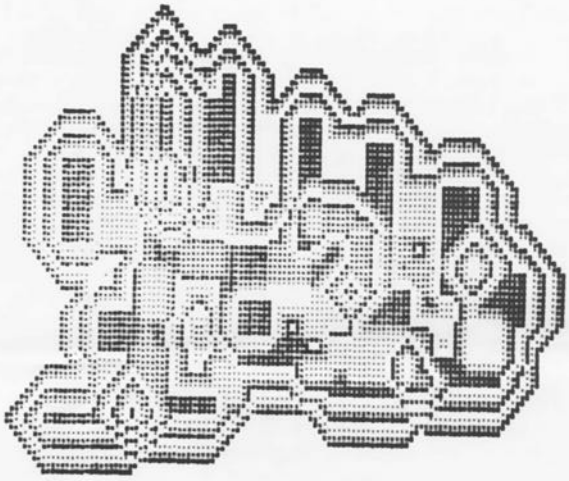
The system which is presently being developed employs Buchla series 200 modules and a digital output keyboard recently designed by Donald Buchla. Modules included in the system are: 206 Dual Mixer, 218 Keyboard, 258 Dual Oscillator, 265 Source of Uncertainty, 275 Reverberation Unit, 291 Voltage Controlled Filter, 292 Gate, and 295 Ten Channel Filter. Envelopes, and other time varying functions will be generated by units in the interface.

THE PDP11-20 AND TELETYPE

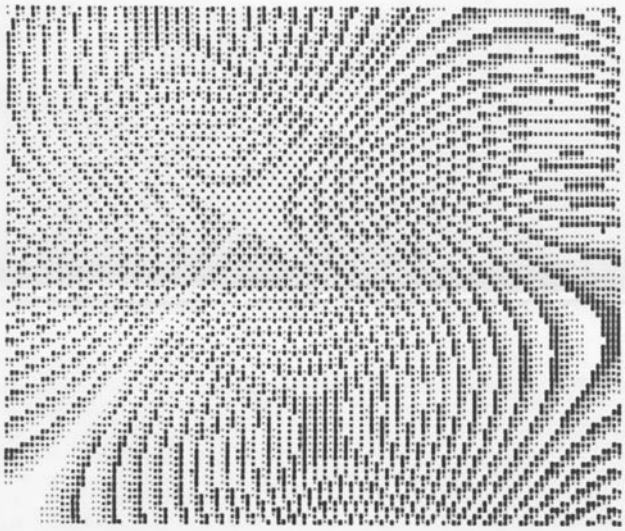
The ISMUS system now includes a PDP11-20 with 8K of core memory, and attached teletype with paper-tape reader and punch. Memory capacity will have to be expanded if ISMUS is to attain the



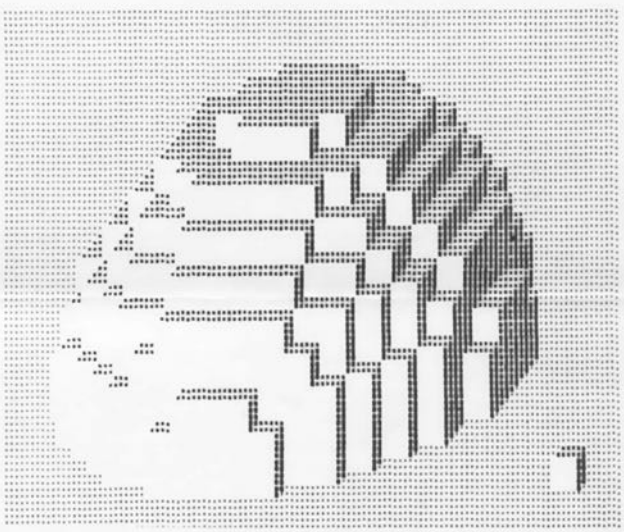
1



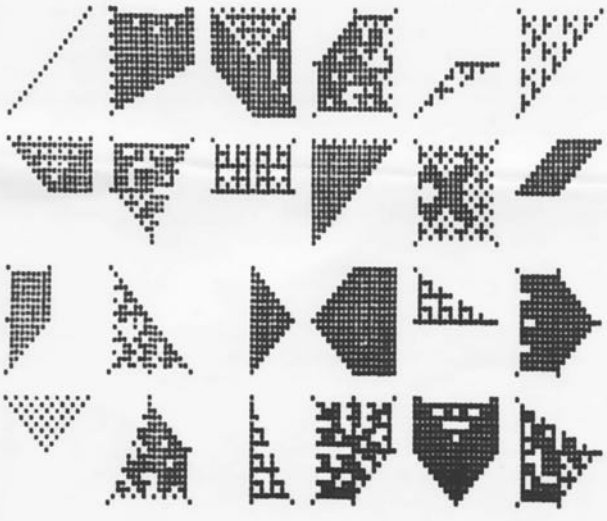
2



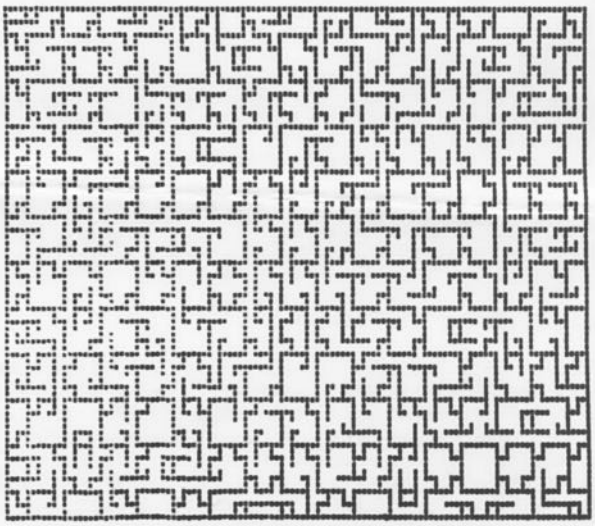
3



4



5



6

full power envisioned for it. Either fixed-head disk or additional core, or, most probably, both, will be needed soon.

SYSTEM INTERFACES

The block diagram shown in Figure 1 reflects present thinking as to system configuration. Obviously this configuration will change as system design progresses and constraints and limitations are more clearly identified.

ISMUS HARDWARE SYSTEM

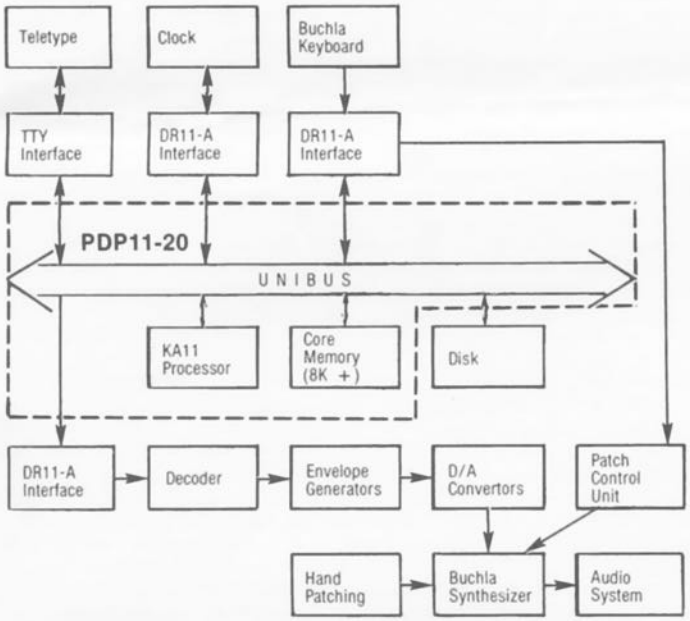


Figure I

User input to the system is via the Buchla keyboard and a teletype unit. The keyboard input at this time is considered to be very general and is not simply limited to the pitch-timing input normally associated with a keyboard. The keyboard provides digital signals to the system as shown.

The PDP-11 computer is the principal unit in the proposed system. Communication with this system is via its "UNIBUS", as shown. A real time clock is provided to serve as a system reference. The clock runs separately from the rest of the system, and provides a tempo reference.

The PDP-11 delivers to the interface system two basic types of information units: PATCH UPDATE commands and CONTROL commands. Patch update comands dictate changes in system interconnection, permitting dynamic program-controlled reconnection of the sound-producing modules in the Buchla synthesizer. Control commands provide numerical values and destinations of control voltages which drive these modules. As shown in the figure, control voltage values must be converted to analog form before being routed to the synthesizer module inputs. Envelopes and other time varying control voltages will be produced by envelope-processing units which will receive voltage values and achievement times from the PDP-11.

The remainder of the system consists of the Buchla synthesizer and switching units to control interconnection of the modules which comprise the synthesizer. Audio output from the synthesizer units drives a speaker system and an audio tape recorder.

An important feature of the proposed system is the ability to dynamically modify system interconnection under program control. It must be admitted that this feature is most demanding of design ingenuity.

SOFTWARE

Figure II shows current software design.

The command PLAY triggers the Link Processor, which prepares the layers indicated by the user for performance. The Real-Time Processor concurrently handles playback of stored layers and acceptance and recording of keyboard performance.

Eventually, as the ISMUS system is built up and becomes more complex, additional sophistication will have to be built into the Link Processor and, especially, the Real-Time Processor. For example, simultaneous, and perhaps conflicting, demands for the same resources will have to be fielded smoothly, with disrupting the musical flow.

ISMUS SOFTWARE SYSTEM

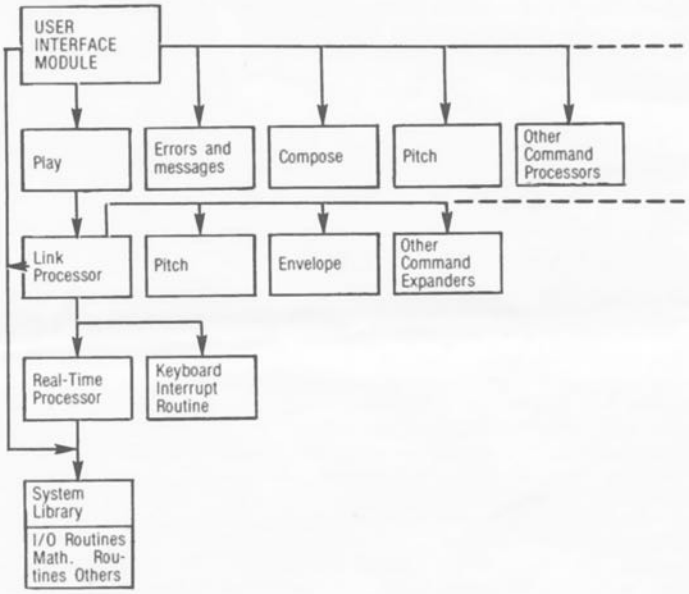


Figure II

MUSYS—software for an electronic music studio

Software — Practice & Experience, 3, 4 (Oct.-Dec. 1973), 369-383.

In recent years the image of the electronic music composer has changed from that of the mad tinkerer, with discarded radio and TV test equipment, to that of a sophisticated user of fourth-generation mini- (and maxi-) digital computer systems. As the hardware that found its way into the studio became more complex, miniaturized, and digitalized, the next step to minicomputer-controlled devices came quite naturally. And with it cam programming languages specifically designed for music composition.

Since the late 1950s we have seen digital computers applied to music composition in a variety of ways. The first was the generation of conventional scores (in either lexigraphic code or plotter-drawn graphic renderings) for conventional instruments, based on compositional rules programmed in an appropriate non-numerical language. Following work done at Bell Labs, large computers were later connected to very fast digital-to-analog converters to produce analog sound in 25,000 12-bit samples per second using programs that simulated sound-generating devices.

In 1969, with the advent of modular electronic music synthesizers, the place of the minicomputer as an integral part of the electronic music studio became a reality. Now the actual sound generation could be left to analog signal-generating devices under digital control from a program running in real time. However, the problem of defining a programming language to describe musical events still remains. Interfacing the composer to this hybrid system is a serious research problem and this article describes one very simple (and admittedly limited) approach implemented recently at the London Electronic Music Studios by the author. A dual PDP8 system was involved with specially built digitally controlled analog signal generators.

The MUSYS program supplies the composer/programmer with a text editor to prepare or modify his program/score, a compiler to assemble the score into coded lists of command data, a performance program to read the assembled lists and prepare it for delivery to the sound generators, and the delivery program (the conductor?) to actually issue the program commands to the sound-generating devices in real time.

In order to describe a musical event, the composer/programmer is required to specify in the program all the analog components needed to generate the desired sonic effect, and to specify the temporal coordinates to locate the event in time. All this is essentially analagous to the informational content of a conventional graphic score, retranslated into a linear string of ASCII symbols as a command language entered through a teletype. As most musical events require complex descriptions, MUSYS is wisely provided with a macro-generator to relieve some of the programming detail.

The paper gives a very practical notion of the system, and lists a number of compositions by notable composers (Hans Werner

Henze and Harrison Birtwistle included) for which MUSYS was successful. There is no discussion of the aesthetics involved, which is admirable, but one is left to wonder about the problems of musical style inherent in the limitations of one programming language over another.

CREATIVE COMPUTING

Creative Computing is a new non-profit magazine of educational and recreational computing published by Ideametrics, P.O. Box 789-M, Morristown, New Jersey 07960. Subscriptions to this "refreshingly informative" magazine are $8 per year (foreign $9.50).

David H. Ahl, editor-in-chief and publisher has a special art issue planned for March-April 1976.

Why not contribute to this issue? Contributions should be 250 to 1500 words — or more if you have a lot to say! Typed, double-spaced. Please consider the questions below in preparing your article.

Get your material in EARLY. Absolute, final, last cut-off date is October 15, 1975 but don't wait 'till then. Also, early material has a much higher probability getting a good spot in the issue. DO IT TODAY!

How/why did you become involved with the computer (in producing art)?

What is your art background?

What role does the computer play for you . . . simulation, tool, etc.? What is your role?

Are your computer works related to non-computer art?

Do you have a final image in mind when work begins?

Could your work be done without the aid of a computer? If yes, why use the computer?

To what extent are you involved in the technical production of your work, for example, in programming?

Do you feel art work created with a computer has now or will have an impact on art as a whole in the future?

Do you intend to continue using the computer to create art pieces?

Do you recommend the use of the computer for others in creating works of are?

Along with your article, opinion, or other good words we would like illustrations, graphics, and photos of your work. Reproduction quality please (sharp B & W artwork, glossy B & W photos 5 x 7 or larger, preferably 8 x 10).

READERS AND WRITERS!! Please submit additional questions you'd like us to focus on.

Please send all material, artwork, responses questions, etc. direct to the *Creative Computing* art issue guest editor:

Ruth Leavitt
5315 Dupont Ave. South
Minneapolis, MN 55419
(612) 825-9005

AIMS AND MEMBERSHIP

The Society aims to encourage the creative use of computers in the arts and allow the exchange of information in this area. Membership is open to all at £2 or $6 per year, students half price. Members receive PAGE eight times a year, and reduced prices for the Society's public meetings and events. The Society has the status of a specialist group of the British Computer Society, but membership of the two societies is independent.

Libraries and institutions can subscribe to PAGE for £2 or $6 per year. No other membership rights are conferred and there is no form of membership for organizations or groups. Membership and subscriptions run from January to December. On these matters and for other infomration write to Alan Sutcliffe or Kurt Lauckner (U.S.A.)

COMPUTER ARTS SOCIETY ADDRESSES

Chairman: Alan Sutcliffe, 4 Binfield Road Workingham, Berkshire, Eng.

Secretary: John Lansdown 50/61 Russell Square, London WC1B4JX.

Dutch Branch (CASH): Leo Geurts and Lambert Meertens, Mathematisch Centrum, Tweede Boerhaavestraat 49, Amsterdam, Holland.

U.S. Branch (CASUS) Coordinator: Kurt Lauckner, Mathematics Dept., Eastern Michigan University, Ypsilanti, Michigan, 48197 U.S.A.

This issue of PAGE was edited by Kurt Lauckner.

O.C.L.E.S.

William P. Uphoff

Recently I had the pleasure to attend a conference on computer arts at Purdue University. It was a fitting time for my introduction to the computer and computer artists since my own work has led me to a point of economic and technical complexity that I feel unable or unwilling to push beyond its present form. I have little desire to become an electrical engineer, but desire to utilize control systems which have in the past required their assistance. My best effort to date, in collaboration with an engineer and without computer assistance, is the O.C.L.E.S. (Observer Controlled Light Emission System). The system utilizes high speed integrated circuit switching logic to control five optical systems housed in a projection apparatus.

OCLES was an interdisciplinary project between art and engineer-created by myself and Jack Thomas with assistance from Gene Wiskerson and Fred Price. The project consisted of procurement of money from the State of California, industry outside of the institution and personal monies. Mr. Thomas and I worked extensively together in all phases of the project and coordinated the various departments of California State University, Northridge, including Art, Engineering, Psychology, Music, Physical Science and Education. I designed all the optics, support systems and exterior display shells, while Mr. Thomas invented all the electronic circuitry.

OCLES deals with programmed and random directional sequences, environmental control, audience participation and dematerialization of sculpture by shifting the emphasis from the object to the experience. OCLES is a light projection system operated by five people simultaneously, each person having 19 controls which govern one optical system. Altogether the five participants have complete command over visual and audio output of OCLES with 3.5×10^8 possible combinations. The light display that OCLES emits enables a very precise pattern of color to be established and changed in very small increments. The pattern can be manipulated at fast or slow rates. It may be moved in the forward direction, stopped, reversed and returned to its exact original configuration. Recently OCLES was on display at the Cranbrook Science Institute Planetarium, Bloomfield Hills, Michigan.

I believe that Art can and must exist on many levels. A device such as OCLES, created to generate visually stimulating movements of color and abstract images, can excite participants to a purely ethereal experience. On a different, more intellectual level, however, OCLES can create a situation whereby individual participants can learn basic cues about their own creativity. It is a decision-making process which constitutes these cues; when different characteristics of the end-product are controlled by the participant to get visual feedback loops established between the total end-product and the other participants.

The computer appears to be a natural next step to facilitate control over interactive works of art such as OCLES. The advantage of such a device is that artists would not have to design and build an electrical system each time they make a piece of art. The artist would simply program the desired output via the computer. This would allow the artist more time for aesthetic concerns instead of technical ones.

My recent involvement in computer-assisted art demands the same critical and serious attitude any other work of art would demand. The same issue, that of making vital art, must be held in tact while developing a meaningful dialog between the issue, tool, and artist. To my disappointment, many computer artists appear to forget the issue in favor of a fascination with the tool or what the tool can do. Perhaps this is where the problem lies. The computer does not make art, artists do. The artists who choose the computer as their medium should learn more about their tool. An old Japanese woodcarver, named Munakata, in a lecture, devoted over an hour to his tools, denoting their importance as a means to the language he used in carving. He was quick to note that only by knowing one's tools could one be truly free to make art.

Many computer artists rely on programmers to delineate the limitations within which they must work. Would it not be more advantageous for artists to learn programming as an essential step in the formulation of their art? It is pure nonsense for an artist to say that he only wants to use the computer, but does not want to understand it. Yet how many artists working in the medium hold such an attitude? The new programs beginning at universities, where the computer is used by artists for their work, can and must address themselves to the importance of understanding the tool. These programs must equally address themselves to aesthetic issues so that the computer, in the future, will not be a fascination for artists. New programs like ARSTECNICA at University of Massachusetts, Amherst under Robert Mallory, have an excellent opportunity to educate a new generation of computer artists, if a playful attitude and a serious approach can be fostered by instructors.