MINI-EXPLOR

A FORTRAN-Coded Version of the EXPLOR

Language for Mini (and larger) Computers

by

Ken Knowlton
Bell Laboratories
Murray Hill, New Jersey 07974

ABSTRACT

The EXPLOR language, which generates designs from Explicit 2-D Patterns, Local Operations, and Randomness, has been useful not only in providing the computer novice with graphic output; it has also been a vehicle for introducing many other basic computational notions: algorithms, nested loops, sorting, heuristics and search, cellular automata, Monte Carlo calculations, finite state machines.

This paper described a simplified version of the system, coded in only 430 lines of FORTRAN, which can run on most 16-bit word (or larger) machines having 8K to 16K of core storage. Such an implementation, by means of line printer or teletype output, can now produce almost as rich a variety of results as previously-reported much larger versions.

****

Information on how to get MINI-EXPLOR may be obtained from the Computing Information Service Group, Rm.2C-548, Bell Laboratories, Murray Hill, N.J. 07974.

****

I. Introduction

The previously reported EXPLOR language[1,2] generates two-dimensional patterns, designs and pictures from Explicitly provided 2-D Patterns, Local Operations and Randomness. 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,[2] East Michigan University and Syracuse University; the "Santa Cruz" version has been exported to 30 other colleges and universities.

This paper presents an abbreviated but powerful version of the system, coded in only 430 lines of that portable subset of American Standard FORTRAN called PFORT.[3,4,5] 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.[6] 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, r0,r1,r2,r3)
```

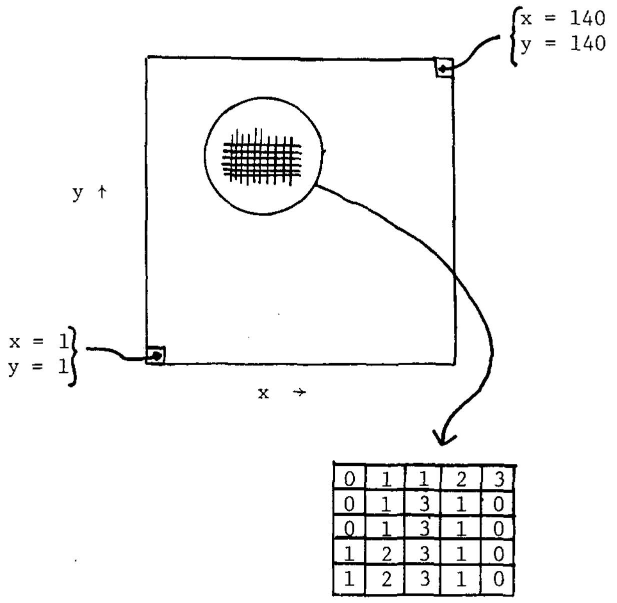
To use these routines effectively, the user needs to learn that part of FORTRAN dealing with the following:

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, MAXØ, MØD, LABS, ISIGN
arrays: 1, 2, and 3-dimensional

All parameters used in subroutine calls are <u>integers</u> (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-stored 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 1/2 cell left of center if width is an even number)

y is the y-coordinate of the center of the rectangle (or 1/2 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 pseudorandum basis. (100 means process all of the cells for certain).

FUNCTIONS

NUM (x,y) has a value (0-3) of the number 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 then 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: , ', ×, and ▩, 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 leftmost 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, from 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) contain 1's, 2's, or 0's."

The routine works in such a way that effects do not propagate during a single cell. 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 areas, and is imagined to come about as follows: a copy of the neighborhood of the "form" area centered at (xf,yf) is picked up, (re) oriented according to the value 1-8 of 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 four translation rules, four examples are here given:

(a) ..., $\overbrace{0000}^{0},\overbrace{1111}^{1},\overbrace{2222}^{2},\overbrace{3333}^{3}$) "from" area cell contents
(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 0'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 cells 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 user should note that in the foregoing description, there are many instances where individual digits of 4-digit numbers have independent significance; they have been grouped as a single integer only for convenience of the user in communicating through the structure and syntax of FORTRAN. In all instances, dummy variables in descriptions stand either for explicit 4-digit integers (with leading zeros optionally omitted) or for variables whose values during execution will be meaningful integers in the sense discussed.

III. Programming Examples (see pages 37-42).

By means of 6 sample programs and their results, I shall try to demonstrate the usefulness of MINI-EXPLOR by exhibiting the conciseness of programs written in it and a variety of graphic results. Since the main attempt here is to suggest possibilities of range and sophistication of programs, these examples are not particularly for the beginning programmer; they certainly should be understood rather quickly by someone who tends to use this language for teaching computer programming and/or computer graphics. It will become obvious that developing a facility with arithmetic expressions is an important part of making the pursuit interesting. In the sample programs, all FORTRAN statements are numbered to facilitate talking about them; only the indented statement numbers are referenced by the program. As visual aids to understanding, explicit patterns defined by calls to PUT4 and PUT16 are, in addition, outlined in the listing, and parameters are grouped as in the subroutine descriptions. Each result shown is a photograph of a single page of printer output: 80 lines of 120 (overprinted) character positions, printed 8 lines per inch.

Example 1 makes use of a module (lines 2-8) centered at (137,137) and so designed that lines of 3's lead to the center of each edge of the 7x7 area; these lines will connect when variously oriented copies of the module are placed side-by-side to fill 2-D space. The overall pattern is thus generated by the double DO loop (lines 9-11) using the "copy" form of CØMBN, with the random orientations achieved by expressing the orientation as NE(1,8). The program begins (line 1) by filling a large rectangle with 3's; at the end only the outer-most layer remains as the frame.

Example 2 is simply a contour plot of an arbitrarily chosen function of x and y:

$$\left|\frac{(x-y)\ (x+2y)}{70}\right|$$

The expression is evaluated at each cell (x,y), and values are turned into ramps of grey scale by reducing them modulo 4.

Example 3 is a "contour map" of 18 overlapping octagonal pyramids, where each "pyramid" (lines 7-8) is made by applying the translation rule 3012 to nine rectangles related thus:

diagram on page 36

A total of 18 such pyramids are constructed progressively narrower, taller, and higher in the picture. (Note in lines 2-6 how IX,IY,IW, and IH relate to K, the number of the pyramid). They are placed at random positions laterally except that extensive overlapping tends to be avoided as follows: if the tentatively chosen center of a pyramid falls on a cell that does not contain a 0, another choice is made for x (lines 3-4).

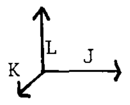
Example 4 is from the much celebrated "game of life" invented by John Conway[7,8] in this instance starting with the "π" configuration in the lower right and following it through 24 iterations, making a copy elsewhere between successive steps. The "game" is an iteration over a square raster where cells have two states which we shall here call spot (3) and vacant (0); spots appear and disappear in the next iteration according to the number of spots among the eight neighboring cells: in the next stage, spots will have appeared at previously vacant cells having had exactly three spots as neighbors, and (simultaneously) all previously existing spots will have disappeared except those that had exactly 2 or 3 neighboring spots. Our implementation of these rules is achieved in three steps as follows: first (line 8) the vacant cells (0's) that are going to become spots are changed to ;'s - i.e., if 3 of the entire neighborhood (757) are 3's, turn 0's into 1's (1123). Second, (line 9) those spots (3's) which have 2 or 3 3's as neighbors become temporarily 2's. Finally, (line 10) the spots to appear (1's) and to be preserved (2's) turn into 3's as all other spots become vacancies (0's) by the rule 0330.

Example 5 gives 24 instances of an 8-times iterated local operation which achieves some sort of growth from a single "occupied" cell (i.e., a 3 on a background of 0's). For each of these 24 trials the neighborhood and the acceptable-numbers-of-neighboring-3's are chosen at random but in accordance with the following considerations: (1) A number designating a neighborhood is randomly composed lines (4-8) by effectively adding or not adding the numbers 400, 200, 100, 40, 10, 4, 2 and 1 but, if the neighborhood thus chosen is null (0), the process is repeated until a finite neighborhood results. (2) The number-of-neighboring-3's must include 1 (i.e., designated as 1000 in line 9) so that the growth can start from a single isolated 3; three other randomly chosen numbers-of-3's (lines 9-11) are chosen, which may result in 2 kinds of slightly nonsensical but harmless specifications: (a) a digit may be repeated, in which case the second appearance is ignored (e.g., MANY = 1353 is effectively the same as MANY = 135 and (b) the acceptable number of neighbors may be larger then the specified neighborhood, in which case, of course, that number of 3's can never be counted - e.g., ...1738, 700, 3, 3333) is effectively the same as ...13, 700, 3, 3333). Appendix A specifies in detail, for those who really want to know, what happens in cases of nonvalid or questionable values of parameters.

Example 6 demonstrates a generalizable method of making quasi-perspective renderings of architectural or other structures composed of cubes. First the picture of a single cube is formed (lines 2-8) which, in this case, appears in the lower right corner of the final output. Then the program iterates through 3-D space, in effect asking whether each call thus visited is a part of the structure, i.e., is "occupied" by a cube. If so, the basic cube is copied into the picture-plane position (IX,IY) of this cell (note that 0's, 2's and 3's representing visible surfaces of the cube are copied, but 1's are not copied by the set of rules

... 0000, 0123, 2222, 3333)

Since the order of copying cubes is back-to-front in an axis system that points toward the observer



the nearer cubes automatically obscure appropriate parts of the more distant ones. In this particular example, the test for occupancy by a cube (in line 12) is essentially the formula for a sphere, i.e., if

$$(J-9)^2 + (K-9)^2 + (L-1)^2$$

is less than 70, then the center of the cube at (J,K,L) is less than $\sqrt{70}$ from the point (9,9,1). For more complicated structures, the programmer may first want to set up a 3-D array to be filled with numbers indicating occupancy by cubes (or by one of a variety of basic forms): the test would then involve interrogating the array.

IV. Acknowledgments

For help in formulating the original EXPLOR language, I thank Lillian Schwartz, with whom I have made several computer art films, largely with EXPLOR.[9] For later variations of the language, I am indebted largely to students - at the University of California at Santa Cruz, East Michigan University and Syracuse University - and to Dan Ross who did a superb job of coding the elaborate, well documented and potentially interactive version of (FORTRAN-coded) EXPLOR which may be obtained from the Information and Computer Science Department at UCSC.

For help in exportability consideration of the present version, I thank Andrew Hall, Barbara Ryder and Brian Kernighan. For verifying its usefulness, and supplying actual printout for the figures of this paper, thanks to Prof. Joe Scala, who is presently using MINI-EXPLOR to teach beginning students in computer graphics/ programming at Syracuse University, and thanks to his assistnat, David Carr, for scrutinizing the code and making several helpful suggestions.

REFERENCES

1. Knowlton, Ken, "EXPLOR - A Generator of Images from Explicit Patterns, Local Operations, and Randomness," Proc. of the 9th Annual UAIDE Meeting, Miami Beach, Florida, (1970)pp.543-583.

2. Knowlton, Ken, "A Report on the Use of FORTRAN-Coded EXPLOR for the Teaching of Computer Graphics and Computer Art," Proc. of ACM SIGPLAN Symposium on 2-D Man-Machine Communication, Los Alamos, N.M., Oct. 1972.

3. USA Standard FORTRAN, USA Standards Institute, New York, New York, 1966.

4. Clarification of FORTRAN Standards - Initial Progress, Communications of the ACM, Vol. 12, May 1969, pp. 289-294.

5. Clarification of FORTRAN Standards - Second Report, Communications of the ACM, Vol. 14, October 1971, pp. 628-642.

6. B. G. Ryder, "The PFORT Verifier," Software Practice and Experience, Vol. 4, No. 4, October-December 1974, pp. 359-378.

7. Gardner, Martin, "Mathematical Games," Scientific American, October 1970, pp. 120-123

8. Gardner, Martin, "Mathematical Games," Scientific American, February 1971, pp. 112-117.

9. Knowlton, Ken, "Collaborations with Artists - A Programmer's Reflections," in Graphic Languages, F. Nake and A. Rosenfeld, Eds., North Holland Publishing Co., (1972), pp. 399-418.

APPENDIX A
ILLEGAL OR QUESTIONABLE VALUES

MINI-EXPLOR does not issue warning comments other than diagnostics which emanate from the local FORTRAN compiler. My attitude, stemming largely from experience with this and other graphics, is that both logical and esthetic mistakes are most often and most easily diagnosed by looking at the pictorial result. A complete description of the language should, nevertheless, say what happens or fails to happen when wrong or questionable values or combinations of parameters are used. (This information may be particularly useful to the more sophisticated programmer who may, for example, be generating these values randomly and/or indirectly, as the neighborhood operation of Example 5.) This appendix attempts to supply the gritty details.

These comments do not include the effects of overflow, truncation, etc. which happen differently on different machines.

NUM (x,y)
Is defined even before any picture-changing operations have been performed (the very first function or subroutine of a program, except NE, first causes the picture to be set to 0's).

NE (min, max)
If the range |max-min| + 1 is greater than 199 it is taken to be 199 and the resulting value is a choice from the smallest supplied parameter to that number plus 198.

Note the tendency to write lines 9-11 of example 5 as MANY=1000+100*NE(2,8)+10*NE(2,8)+NE(2,8). Some FORTRAN compilers will "optimize" this effectively to MANY=1000+111*NE(2,8).

CALL SHØW (x, y, w, h)
If rectified area is null (no overlap between area specified and internally represented picture) no action is taken (a blank page does not result).

CALL PUT (x,y,n)
If n < 0 or n > 3, no action results.

CALL PUT4 (x,y,n)
If n < 0, no action results.
If n > 9999, additional leading digits are ignored.
If any digit >3, corresponding picture cell is unchanged.

CALL PUT16 (x,y, n1,n2,n3,n4)
Each n corresponds to 4 successive picture cells and is treated individually as in PUT4.

CALL CHANJ (x,y,w,h,%, r)
CALL LØCØP (x,y,w,h,%, many, nabrs,these, r)
CALL CØMBN (x,y,w,h,%, xf,yf,orientation, r0,r1, r2,r3)
If w or h or % ≤ 0 no action results.
If % >100 it is taken to be 100.

Translation rules
If any rule is <0 or >3333, no action is taken
If any digit >3 is taken modulo 4.

many (numbers of neighbors)
If many >8888 ormany< 0, no action results.
Leading 0's are ignored.
0, to be an acceptable count, must appear in units position.
If a digit repeats, the effect is the same as if it appeared just once.
There is no automatic check that the combination of many and nabors makes sense. It can, therefore, be time-consuming to get a null result - as a local operation on cells having exactly 6 neighboring cells containing 2's, but where nabrs specifies 5 or fewer neighboring cells.

nabrs (specification of neighborhood)
If nabrs < 1 or nabrs > 757, no action.
Designated neighbors are determined by successive attempts to subtract 400, 200, 100, 40, 10, 4, 2 and 1, without yielding a negative result, each time subtracting from the previous result.

these (ok - numbers)
If these < 0, no action.
If these > 8888, no action
If a digit appears more than once, the effect is the same as if it appeared once.
0, to be a designated number, must appear in units position.
If any digit is >3, the number of neighbors having this value will always be zero.
If orientation is any number other than 1 through 8, no action is taken.

APPENDIX B
IMPLEMENTATION CONSIDERATIONS

The following is a list of suggestions and/or actions required on the part of the implementer:
1. The system as exported consists only of source code for functions and subroutines. The implementer is required to provide users with means for writing main programs and other subroutines, for compiling, loading and linking with object forms of the supplied routines; for means of initiating and terminating runs (time out, STØP, exit to the system, ...).
2. The internal picture is stored in packed format, seven picture cells to the word; a great deal of packing and unpacking is done by the function IPAK and the subroutine SPLIT, respectively. The object form of these should be examined to determine whether a significant increase in speed would result from recoding them directly in assembler language.
3. The FORTRAN function MØD is often a closed subroutine rather than compiled as in-line code. Consequently, where speed is important, MØD(J,K) as been expressed in many places in the supplied code as J-(J/K)*K. The implementer may wish universally to use or to avoid the use of MØD, according to how it is performed by his FORTRAN.
4. For those FORTRAN systems which dynamically allocate labelled COMMON, it will be necessary for the main program to contain all of the following, and it is suggested that all functions and subroutines be supplemented to include these specifications:

```
      CØMMØN/MAP  /LINE(2800)
      CØMMØN/RAND /IRAN(199)
      CØMMØN/SNGLS/NM(560)
      CØMMØN/XLTBL/INTØ(4)
```

5. Some teletype machines and other output devices do not have, in effect, a carriage return independent of a line feed; overprinting can usually be achieved on such devices by backspacing. The routine SHØW must be appropriately revised to do overprinting in this way.
6. The implementer may have at hand more impressive graphic output equipment than printers and teletypes; he is responsible for revising, augmenting, or replacing the SHØW subroutine accordingly.
7. For reasons of speed or convenience, the user may want to use the random number generator supplied with the local FORTRAN. The function NE and the labelled COMMON region/RAND/ must remain, however, because of their use by CHANJ, LØCØP, and CØMBN.
8. The WRITE statements in SHØW usually cause the loading of an unnecessarily large conversion/output routine. It may be replaced by a routine which outputs only alphanumeric strings.

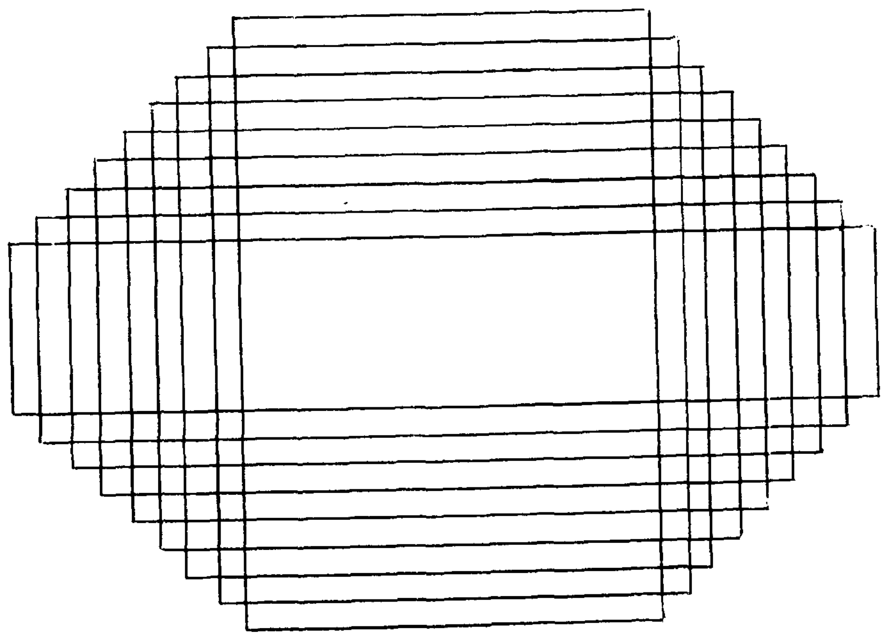
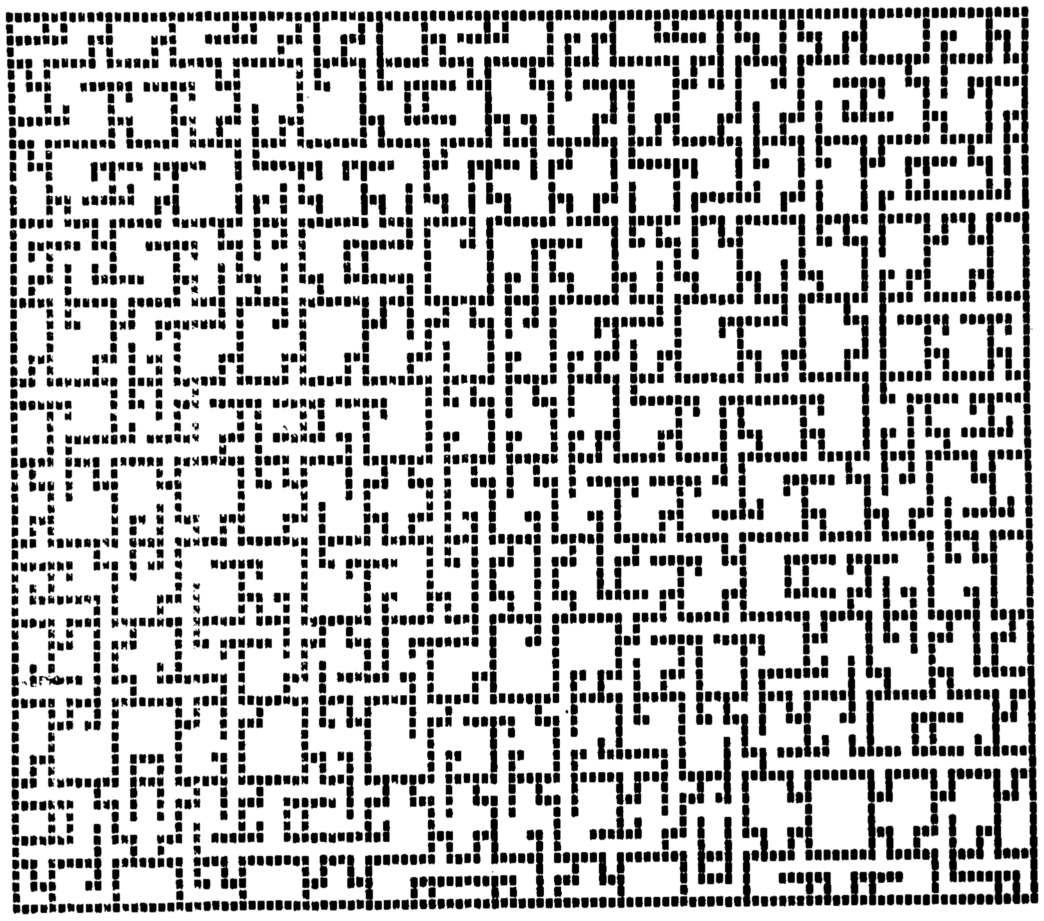


Diagram for example two

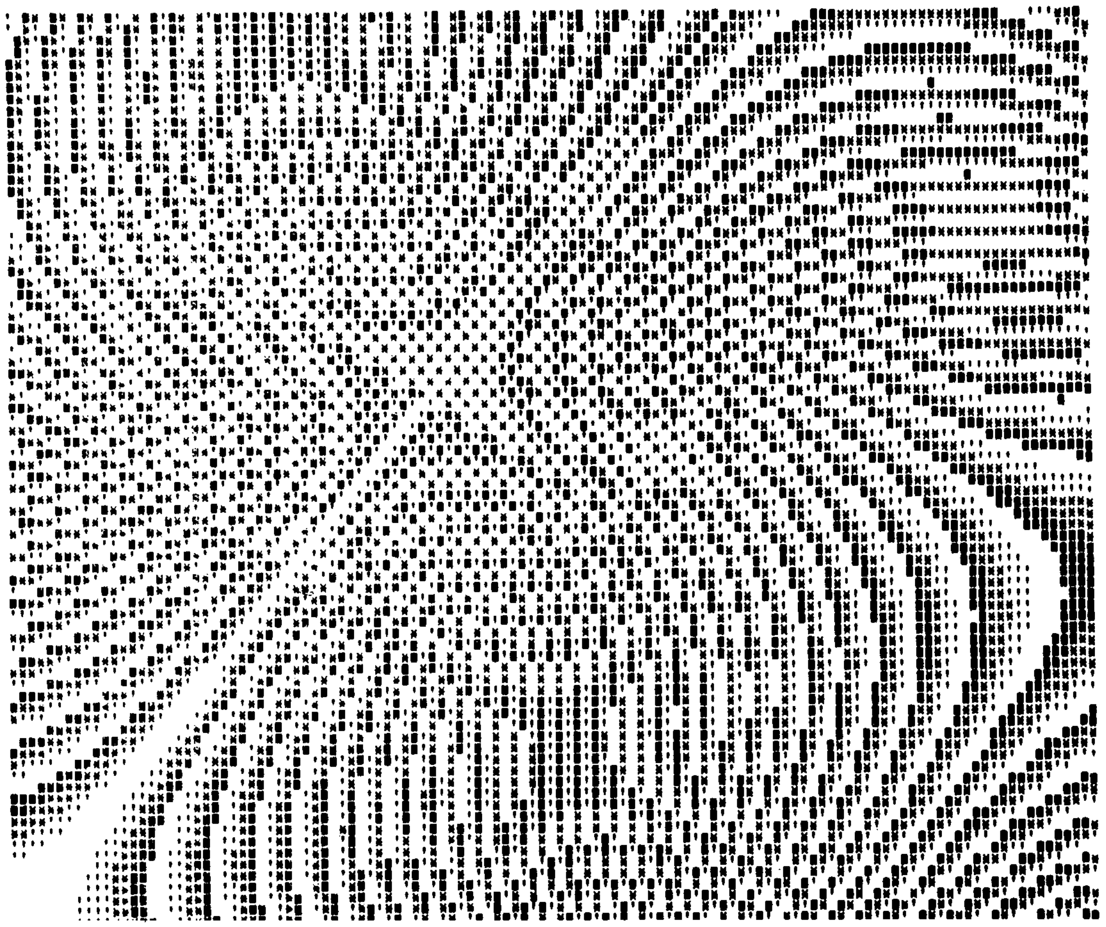
EXAMPLE 1.   RANDOMLY ORIENTED JUXTAPOSED MODULES.

```
C                                     PICTURE FRAME
  1    CALL CHANJ(57,40,114,79,100, 3333)
C                                     THE BASIC MODULE
  2        CALL PUT16(134,140, 0003,0300,0,0,0)
  3        CALL PUT16(134,139, 3333,3300,0,0,0)
  4        CALL PUT16(134,138, 0000,0000,0,0,0)
  5        CALL PUT16(134,137, 3303,3330,0,0,0)
  6        CALL PUT16(134,136, 0303,0000,0,0,0)
  7        CALL PUT16(134,135, 0333,0000,0,0,0)
  8        CALL PUT16(134,134, 0003,0000,0,0,0)
C                                     COPY 11 X 16 TIMES
  9    DO 11 IX=5,110,7
 10    DO 11 IY=5,75,7
   11 CALL COMBN(IX,IY,7,7,100, 137,137,NE(1,8), 0000,1111,2222,3333)
 12    CALL SHOW(60,40,120,80)
```



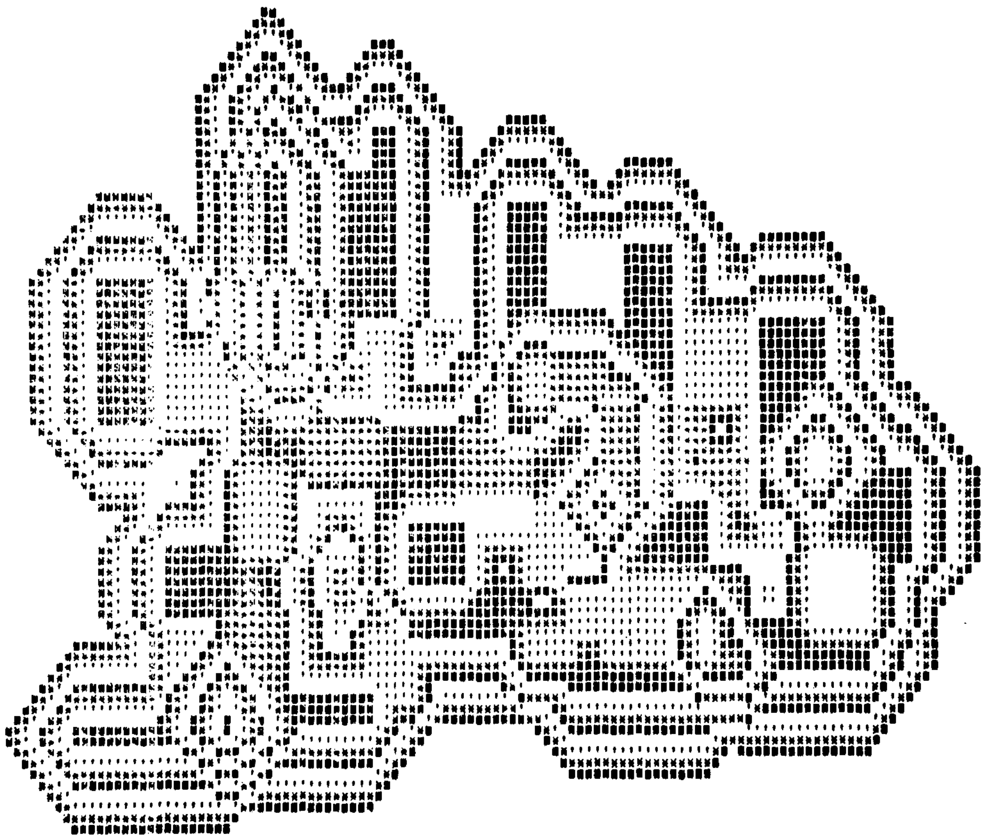
EXAMPLE 2.   CONTOUR PLOT OF A MATHEMATICAL FUNCTION.

```
C                                   2-D ITERATION PIC. CELL BY CELL
  1    DO 3 IX=1,120
  2    DO 3 IY=1,80
     3    CALL PUT(IX,IY ,MOD(IABS((IX-IY)*(IX+2*IY))/70,4))
  4    CALL SHOW(60,40,120,80)
```



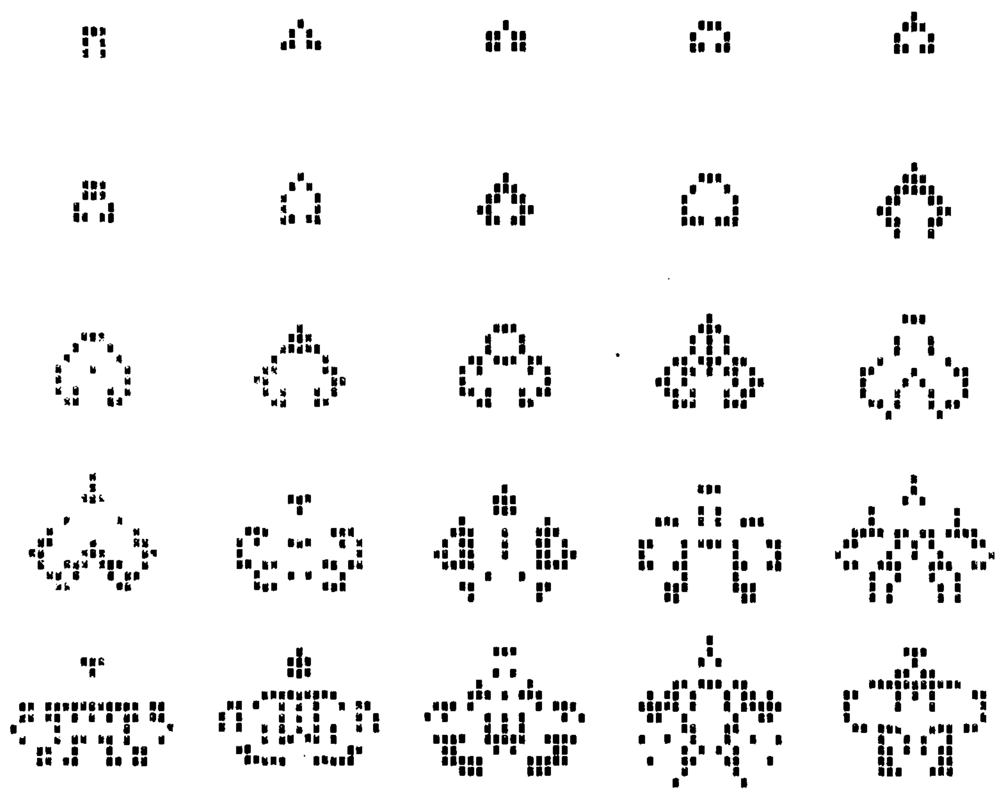
EXAMPLE 3. CONTOUR PLOT OF SUPERIMPOSED OCTAGONAL PYRAMIDS.

```
C                                DO 18 PYRAMIDS
  1    DO 3 K=1,18
  2       IY=8+3*K
    3     IX=NE(15,105)
  4            IF(NUM(IX,IY).NE.0)GO TO 3
  5       IW=18-K
  6       IH=K-1
C                                EACH IS 9 OVERLAPPING RECTANGLES
  7    DO 8 J=1,9
    8     CALL CHANJ(IX,IY,IW+2*J,IH+2*(10-J),100, 3012)
  9    CALL SHOW(60,40,120,80)
```



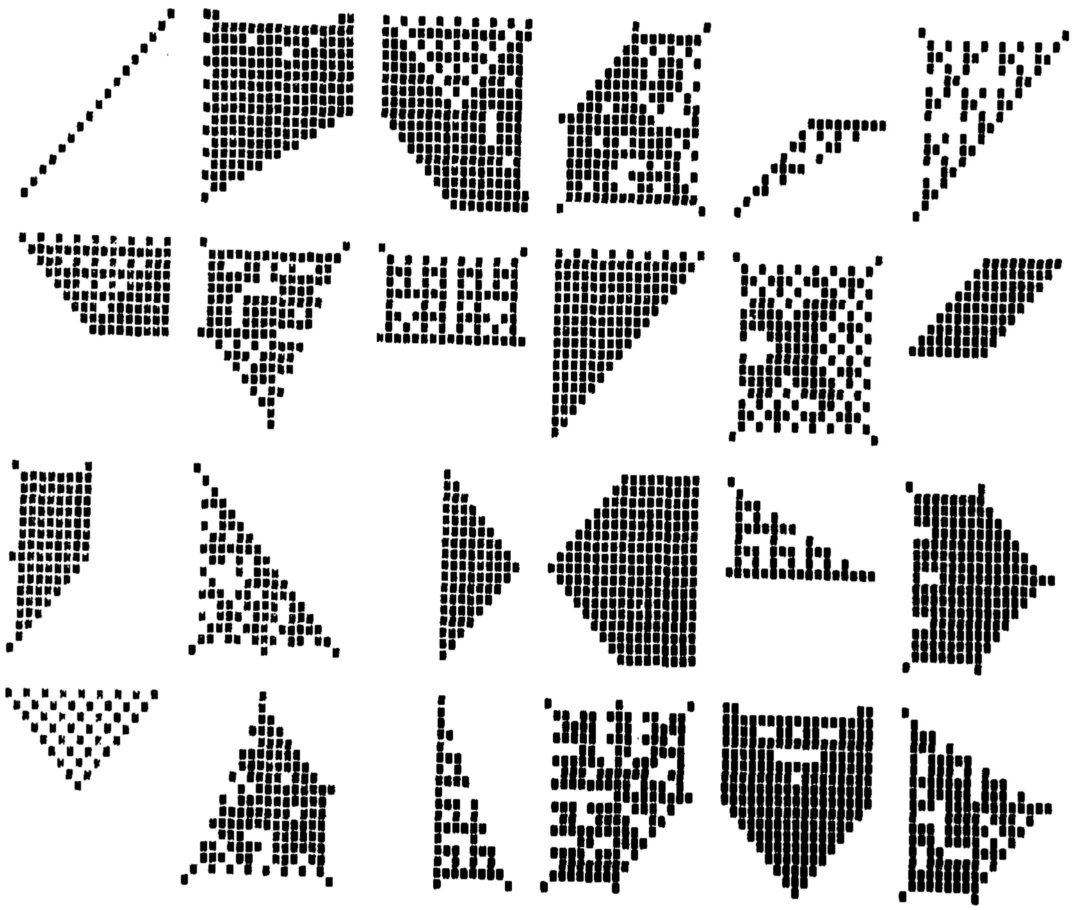
EXAMPLE 4.   GAME OF LIFE, STARTING WITH PI.

```
C                             INITIAL CONFIGURATION
  1        CALL PUT4(107,7,3330)
  2        CALL PUT4(107,6,3030)
  3        CALL PUT4(107,5,3030)
  4     DO 10 J=1,24
C                             MAKE A COPY BEFORE NEXT STEP
  5        IX=MOD(J-1,5)*24+12
  6        IY=72-((J-1)/5)*16
  7        CALL COMBN(IX,IY,24,16,100, 108,8,1, 0000,1111,2222,3333)
C                             THE RULES OF LIFE
  8        CALL LOCOP(108,8,24,16,100, 3,757,3, 1123)
  9        CALL LOCOP(108,8,24,16,100, 23,757,3, 0122)
   10      CALL CHANJ(108,8,24,16,100, 0330)
 11     CALL SHOW(60,40,120,80)
```



EXAMPLE 5. GROWTH FROM NUCLEI BY RANDOMLY SELECTED RULES.

```
C                                 DO 6 X 4 TRIALS
  1   DO 13 J=1,6
  2   DO 13 K=1,4
C                                 PLACE NUCLEUS
  3      CALL PUT(20*J-9,20*K-9, 3)
C                                 SELECT NEIGHBORHOOD AND COUNTS
   4           NABRS=100*NE(0,7)
  5            NABRS=NABRS+NE(0,7)
  6            NABRS=NABRS+40*NE(0,1)
  7            NABRS=NABRS+10*NE(0,1)
  8                IF(NABRS.EQ.0)GO TO 4
  9            MANY=1000+100*NE(2,6)
 10            MANY=MANY+10*NE(2,8)
 11            MANY=MANY+NE(2,8)
C                                 ITERATE 8 TIMES
 12      DO 13 L=1,8
   13    CALL LOCOP(20*J-9,20*K-9,17,17,100, MANY,NABRS,3, 3333)
 14   CALL SHOW(60,40,120,80)
```



EXAMPLE 6.   HEMISPHERE OF SMALL CUBES.

```
C                             BACKGROUND
      CALL CHANJ(60,40,120,80,100,1111)
C                             BASIC CUBE
      CALL PUT16(107,13, 1122,2224,4444,4444)
      CALL PUT16(107,12, 1222,2234,4444,4444)
      CALL PUT16(107,11, 0000,0334,4444,4444)
      CALL PUT16(107,10, 0000,0334,4444,4444)
      CALL PUT16(107,09, 0000,0334,4444,4444)
      CALL PUT16(107,08, 0000,0334,4444,4444)
      CALL PUT16(107,07, 0000,0314,4444,4444)
C                             ITERATE THRU 3-D ARRAY OF CELLS
   DO 16 J=1,17
   DO 16 K=1,17
   DO 16 L=1,9
      IF(((J-9)**2+(K-9)**2+(L-1)**2).GE.70)GO TO 16
C                             THIS CELL OCCUPIED, COPY CUBE
      IX=30+5*J-2*K
      IY=30+5*L-2*K
      CALL COMBN(IX,IY,7,7,100, 110,10,1, 0000,0123,2222,3333)
16 CONTINUE
   CALL SHOW(60,40,120,80)
```