

EXPLOR - A Generator of Images from Explicit Patterns, Local Operations and Randomness

K C Knowlton,

Bell Telephone Laboratories

1970

UAIDE

Abstract

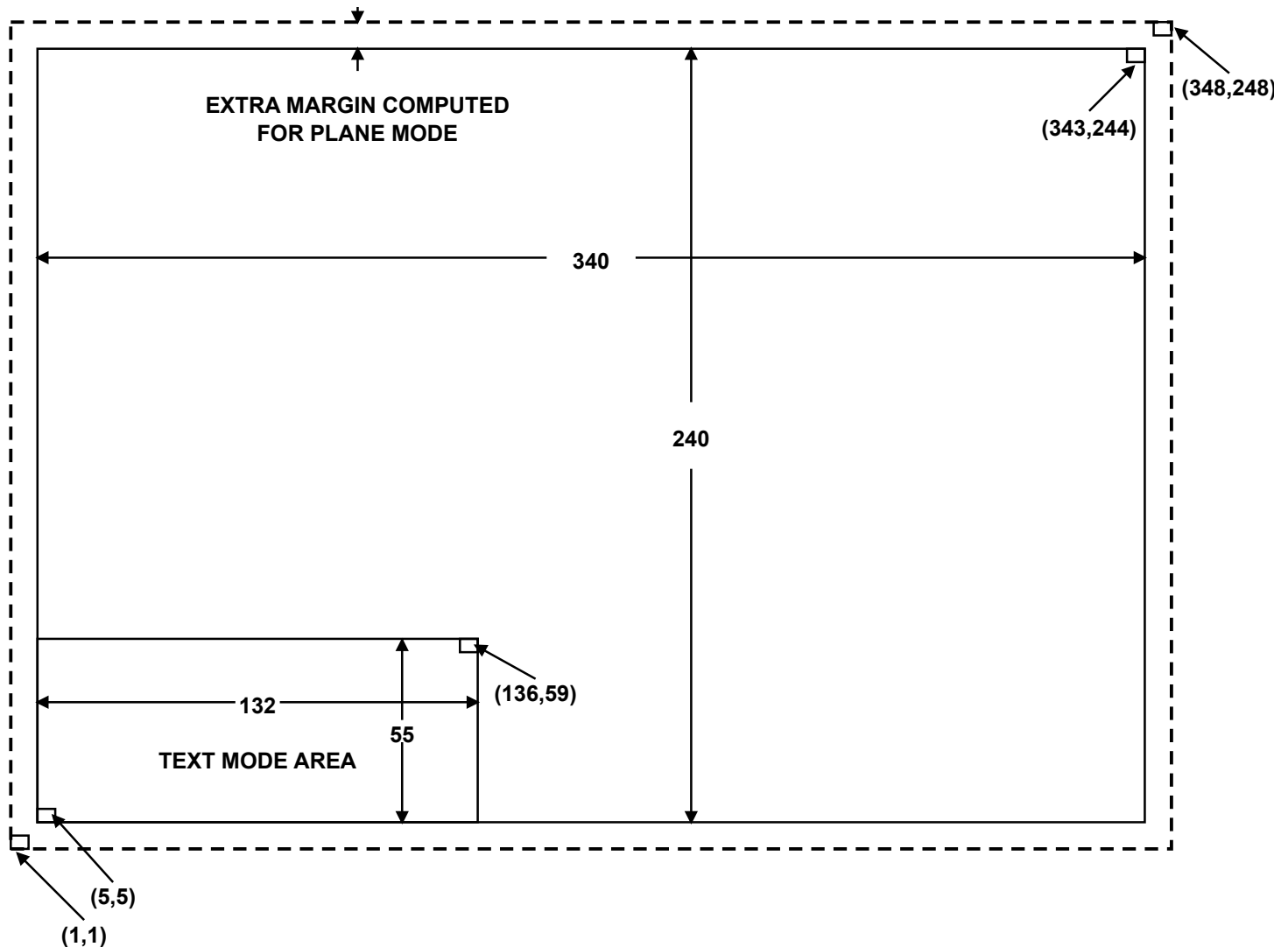
EXPLOR is a system for computer-generation of still or moving images from **EX**PLICITLY defined **P**atterns, **L**ocal **O**perations, and **R**andomness. Output images are rectangular arrays (240×340) of black, white, and **twinkling** dots; internally, information for each position is encoded as an alphanumeric character.

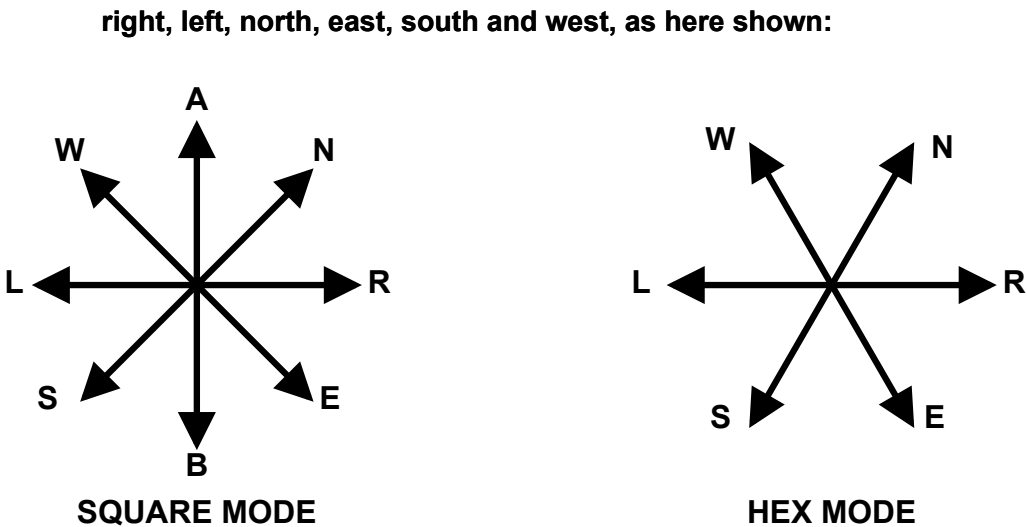
Scientific and artistic applications include the production of stimuli for visual experiments, the depiction of visual **phosphenes** such as moving checkerboards and stripes, and picture processing. The system may also be used to simulate a variety of two-dimensional processes and mechanisms, such as crystal growth and etching, neural (e.g. retinal) nets, random walk, diffusion, and iterative arrays of logic modules.

The EXPLOR system is useful for the production of still and moving pictures for a variety of research, educational and artistic purposes, such as the simulation of two-dimensional processes and the generation of displays for psychophysical experiments on human vision.

Each of the images generated is a two-dimensional array of white and black dots like those in Figs. 1 through 8, to be described later. Within the computer information for each position is stored as a digit 0,...,9, or a letter A,B,...,Z; the programmer specifies which of these characters are to be output as black and which as white dots, and which are to **twinkle**, i.e. be chosen at random (probability = $1/2$), frame by frame, to be black or white.

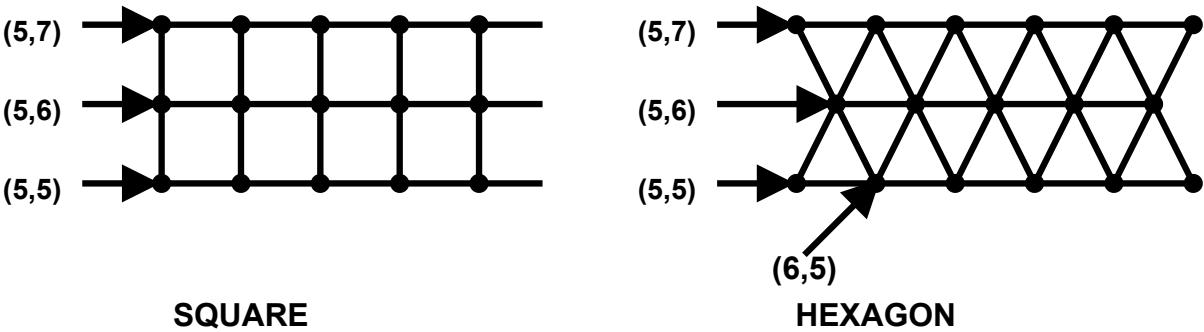
The programmer imagines significant areas for different modes of operation as shown below. The normal run mode area is 340 units wide by 240 high; in the test mode, only the indicated 132×55 area is computed internally, and it is output by printer, not via microfilm:





INSTRUCTIONS OF THE LANGUAGE

Computation and output are similar for square and hexagonal modes except that in film output for hexagonal mode, even-numbered lines are shifted right by half the dot spacing, and a different interpretation is given to the directions of nearest neighbors. Directions are specified by the letters A, B, R, L, N, E, S, W which may be thought of as meaning above, below, right, left, north, east, south, and west, as here shown:



I. INSTRUCTIONS OF THE LANGUAGE

EXPLOR is a macro language. A summary of instruction names and their purposes are given in Table I; each of these and its parameters will be described in detail after a few general considerations that apply to most instructions.

TABLE I - Preview of EXPLOR instructions

for picture output

- MODE**
establish mode as hex or square, run or test, wrap or plane
- WET**
specify character subsets as white, black and twinkling
- CAMERA**
output the array

for changing the internal array

- XL**
transliteration (table look-up and replacement)
- AXL**
adjacency-conditioned transliteration
- PXL**
specific-pair transliteration

BXL

boxed-array transliteration

BAXL

boxed-array adjacency-conditioned transliteration

BPXL

boxed-array specific pair transliteration

for defining patterns**SVP**

save a pattern from the internal array

PAT

explicitly define a pattern by octal numbers (at compile time)

for flow_of control**DO**

do a subroutine and return

GOTO

pass control to named location

IF

goto conditional on values of specified parameters

for instruction modification**CHV**

change the value of named parameter

CHP

change pattern named in an instruction

XLI

transliterate the specified part of an instruction.

Instructions have the form name

name opcode (n,p)list-of-arguments, goto

where **name** and **goto** are optional: a name is required only if control passes to this instruction from other than the one above it, or if this instruction is to be modified by another instruction; if and when the operation is performed, a **goto** if present causes control to pass to the named instruction, otherwise it goes to the line below.

The **periodic** and **probabilistic indicator** (n,p) determines whether the operation and goto will be effective when control reaches this point: every **n**th time through this point, the system **tries** to perform the operation, succeeding with probability $1/p$ (**p** is an integer); otherwise the operation is not performed and control goes to the next line, whether or not a goto is present. Alternatively, if an **X** precedes the **n** and/or the **p**, then the system tries all times except every **n**th time, and/or succeeds on all but $1/p$ of the trials. Examples and their meanings are here given:

```
(1,1)      always do this operation when control gets here
(1,16)     do this one with probability 1/16
(4,1)      do this one every fourth time
(8,2)      every eighth time through, flip a coin to decide
(X,50,1)   except for the 50th, 100th, etc. times, do it
(1,X,9)    do it with probability (1-1/9)
(X,2,X,50) almost always (p = 1-1/50), do it on the beat
```

A **transliteration**, noted as (xlit) in many descriptions of instructions below, is a scheme for replacing some or all of the 36 characters by other ones. It may be specified in one of four ways: First, a complete sequence of 36 characters specifies, in order, the characters into which 0,1,...,9,A,B,...,Z are to be translated. Thus (1234567890BCDEFGHIJKIM N0PQRSTU VWXYZA) says that each digit and letter go into the next higher one, with 9 changing to 0 and Z to A. Secondly: if the sequence is truncated, it is assumed that characters whose positions do not appear remain unchanged. Thus (ABCD) says 0 goes to A, 1 to B, 2 to C and 3 to D, everything else remaining as it was. Third, if the sequence ends in ..., it is assumed that the last character mentioned fills the remaining positions. Thus (012ABC...) means the same as (012ABCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC) and the designation (0...) means that everything becomes zeros. Finally, only specific transitions (at least two) may be specified, separated by commas. Thus (AB,CX,DE) says that A's become B's, C's become X's and D's become E's. If only one such transition is wanted, then a dummy, e.g. change C's to C's, must be added to distinguish this format from the second mentioned: (AB,CC).

1. Picture-output Instructions

MODE (n,p)(list)goto

e.g.

MODE (1,1)(WRP,RUN)

establishes the modes of operation by a list of up to three indicators from the following set:

WRP or PLN

to specify wrap-around (toroidal) continuity or plane-mode

TST or RUNto specify test mode (132×55 , printed output) or run mode (340×240 , film output)**SQR or HEX**

square or hexagonal arrangement of neighbors.

The specified mode remains in effect until countermanded by a subsequent MODE instruction. Default mode is (TST,WRP,SQR).

```
WBT (n,p)(whites,blacks,twinkles)goto
e.g.
WBT (1,1)(01234,56789,ABCXYZ)
```

specifies which characters are output as white, which as black and which ones twinkle (i.e. are independently chosen spot-by-spot, frame-by-frame, to be black or white with 50/50 probability.) In the example, 0 to 4 are white, 5 to 9 black and letters A, B, C, X, Y, and Z twinkle; other letters retain their previous significance. Printed output - i.e. from TST mode - is unaffected by WBT: here zeros come out as blanks and all other characters appear as themselves.

```
CAMERA (n, p)frames,goto
e.g.
CAMERA (4,1)2
```

causes the indicated number of output frames of film to be produced if the mode is RUN; if TST, a single page of printout occurs for every seventh frame of specified output.

2. Instructions for changing the array

```
#
XL (n,p)q(xlit)goto
e.g.
XL (2,1)3(56,78)
]
```

probabilistically subjects all characters of the array to the specified transliteration. In the example, every other time control reaches this point, the instruction is effective - at which time about one-third of the 5's are changed to 6's and about one-third of the 7's are changed to 8's, other characters remaining as they were.

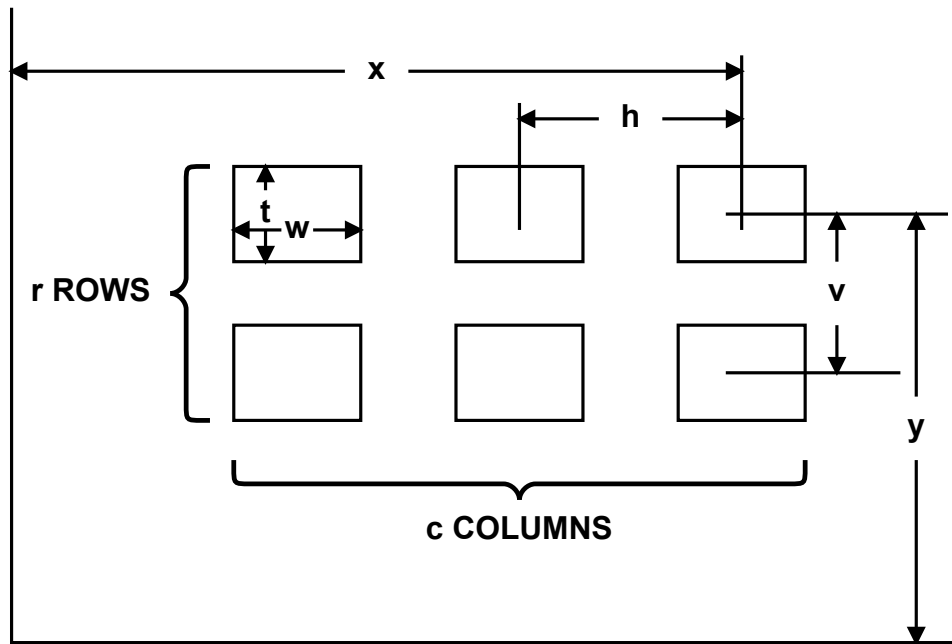
```
AXL (n,p)nums,dirs,chst,q(xlit)goto
e.g.
AXL (1,1)234,RNESW,ABCD,5(XXXXXXXXXX)
```

is a similar probabilistic transliteration but it applies only to those characters made eligible by adjacent characters: if certain numbers of neighbors in specified directions are among the given character set, then, with probability $1/q$ the transliteration is applied. In the example, if exactly 2 or 3 or 4 of the neighbors to the right, north, east, south or west are A's or B's or C's or D's, then about one-fifth of the digits 0-9 thus situated are changed to X's.

```
PXL (n,p)dir,q(tpls)goto
e.g.
PXL (1,1)R,1(12A,13B,24C,23X,14Y,348)
```

transliterates one member of specific pairs. For this operation, the array must contain only digits 0 to 7. In the example, the direction relation is **right of** and it says that 1's with 2's right of them become A's, 1's with 3's right of them become B's, 2's with 4's right of them become C's, etc. All three of the foregoing transliterations may be applied inside of regularly arrayed boxes specified by:

```
x,y  coordinates of center of top right box
w,t  how wide and tall each box is
h,v  center-to-center horizontal and vertical spacing
c,r  number of columns and rows of boxes.
```



An additional parameter, **pat** is either the name of an explicit pattern prescribing exactly which boxes of such an array should in fact be treated, or if an integer, it is the reciprocal of the box-by-box probability of treating the box. If $w > h$ and/or if $t > v$, boxes overlap; overlapped areas are multiply-transliterated. Parts of boxes or entire boxes **wrap around** if the mode is WRP, otherwise those parts or entire boxes falling outside the frame are ignored. The three instructions have B prefixed to the op code, thus:

```
BXL (n,p)pat(x,y,w,t,h,v,c,r)q(xllt)goto
BAXL (n,p)pat(x,y,w,t,h,v,c,r)nums,dirs,chst,q(xlit)goto
BPXL (n,p)pat(x,y,w,t,h,v,c,r)dir,q(tpis)goto
```

3. Instructions for defining patterns

Explicit patterns, specifying a subset of boxes of a rectangular array, are defined by 12-digit octal numbers, (one digit represents three boxes, a one-bit meaning treat the box). All patterns thus defined are multiples of 36 in width, leading zeros being optional. If the number of columns is more than 36, then two or more of the 12-digit numbers are taken to specify the top row of the pattern, the next group of numbers defines the next row, etc. Thus the pattern named TABLE, which may be taken as a side view of a long 85×4 table, is coded remotely (not in line) as follows:

```
TABLE PAT 37777,777777777777,777777777776
PAT 00000,2000000000000,000000200000
PAT 00000,2000000000000,000000200000
PAT 00000,2000000000000,000000200000
```

A pattern may be picked up from the array and saved by the **same pattern** operation which gives the pattern a name and specifies x and y location of the square at the top right corner of the area:

```
SVP (n,p)name,x,y,width,height, goto
```

A pattern thus defined from the internal picture is the indicated height and a multiple of 36 wide (wide enough to accommodate the specified width): The pickup routine does not **wrap around** in picking up characters from the array. Characters picked up are translated according to the current WBT setting, **white** ones turning into one bits, **black** ones into zeros and **twinkling** ones are randomized and preserved as unchanging one or zero bits.

4. Instructions for flow of control

```
GOTO (n,p)goto
```

when operative, according to (n,p) simply causes control to pass to the indicated goto. Another transfer of control is conditional upon whether the first parameter is greater than, equal to, or less than the second:

```
IF (n,p)(parainm1,pred,param2)goto pred = GT,EQ,LT
e.g.
IP (3,2)(SIZE,GT,16)LOC7
```

]

In the example, if according to $(3,2)$ the test is operative and if the current value of the variable SIZE is greater than 16, then control goes to the instruction named LOC7. Gotos, clearly, are mandatory for GOTO and IP statements.

```
DO (n,p)subname, goto
```

causes the current instruction location to be stacked on a pushdown list and control to go to the subroutine whose first instruction has the indicated name. A subroutine normally terminates logically with the special goto DONE, which causes a pop from the pushdown to direct control so as to continue from beyond the call (to the goto of this instruction if there is one, otherwise to the next line).

5. Instructions for instruction modification

Many instructions may be operated upon directly (the system is largely interpretive) or indirectly by changing values of parameters referenced. Table II contains a resume of all instructions; there doubly underlined parameters are character strings that may be changed directly by XLI instructions below, or pattern names changeable by CHP; and singly underlined parameters may be named variables which may be changed in value by CHV:

```
CHV  (n,p)param, change, val1,val2,goto [change = SET, ADD, SUB,MPY, DIV]
e.g.
CHV  (1,1)LENGTH, ADD, 5,15
```

sets the parameter to, or adds to it, or subtracts from it, or multiplies it by, or divides it by a value randomly chosen between val1 and val2, inclusive. In the example, the parameter LENGTH has added to it a number from 5 to 15. If no random selection is wanted, then only the desired number or parameter name need be given as val1, except that if there is a goto, then val2 must also appear (in this case redundantly) in order to preserve the appropriate parameter position for the goto. All parameters have value = 1 until changed by CHV. If a zero value occupies a probability position, p or q, it is taken to be 1.

```
CHP  (n,p)inst,newpat,goto
e.g.
CHP  (1,1)LINE9,PAT7
```

]

causes a change in the named pattern of a BXL, BAXL or BPXL instruction.

```
XLI  (n,p)name,part,q(xlit)goto
e.g.
XLI  (1,1)LINE5,DIRS,1(NR,RE,EB,BS,SL,LW,WA,AN)
```

literally transliterates characters of a part of many kinds of instructions, according to the part specified:

```
NUMS      the numbers of AXL or BAXL
DIRS,DIR  the direction(s) of AXL, BAXL, PXL, or BPXL
CHST      the character set of AXL or BAXL
XLIT      the transliteration of XL, AXL, BXL, or BAXL
WBTS      the character strings of a WBT instruction
TPLS      the triples of PXL or BPXL
```

In the example, the instruction named LINE5 has its named directions changed, each to the next most clockwise direction.

In considering transliteration of instructions by XLI, all character strings may be considered to be stored literally as originally written until later changed; the only characters accessible to change are the digits and letters originally or subsequently appearing. Therefore, some foresight is often necessary if XLI is to be used, for example in providing dummy directions such as X, Y and Z which are ineffective until these characters are transliterated into meaningful directions.

TABLE II - Summary of EXPLOR instructions.

Highlighted parameters may be given explicitly as integers, or as named variables. Gotos are optional except for TEST and GOTO. Doubly underlined parameters may be changed by instruction-modification instructions CHP or XLI.

```

for picture output
MODE  (n,p)(list)goto  [WRP vs PLN, RUN vs TST, SQR vs HEX]
WBT   (n,p)(whites,blacks,twinkles)goto
CAMERA (n,p)frames,goto
for changing the array
XL    (n,p)q(xlit)goto
AXL   (n,p)nums,dirs,chst,q(xlit)goto
PXL   (n,p)dir,q(tpls)goto
BXL   (n,p)pat(x,y,w,t,h,v,c,r)q(xlit)goto
BAXL  (n,p)pat(x,y,w,t,h,v,c,r)nums,dirs,chst,q(xlit)goto
BPXL  (n,p)pat(x,y,w,t,h,v,c,r)dir,q(tpls)goto
for defining patterns
SVP   (n,p)name,x,y,width,height,goto
PAT   list-of-octal-numbers
for flow of control
DO    (n,p)sub,goto
GOTO  (n,p)goto
TEST  (n,p)(param1,pred,param2)goto  [GT,EQ,LT]
for instruction modification
CHV   (n,p)param,change,val1,val2,goto  [change= SET,ADD,SUB,MPY,DIV]
CHP   (n,p)inst,newpat,goto  [for BXL,BAXL or PXL]
XLI   (n,p)name,part,q(xlit)goto  [part= NUMS,DIR(S),CHST,XLIT,WBTS,TPLS]
```

II. EXAMPLES

The flexibility of the language will be demonstrated, and some of its uses illustrated, by eight examples of programs and their pictorial results. These appear together in sets in Figures 1 through 8. The program presented in each case is assumed to be preceded by the initialization

```
MODE (1,1)(RUN, WRP, SQR)
WBT (1,1)(ABCD,0123,WXYZ)
XL (1,1)1(0...)
```

which, unless explicitly countermanded by other instructions listed, establishes the ground rules for mode and output, and clears the entire surface to black. Each listed program is assumed to be followed simply by a command to output one frame:

```
CAMERA (1,1)1
```

In each case, there is one variable with two alternative values: the upper one causes the left-hand set of outputs, the lower the right-hand set. Another variable shows three alternatives which yield the top, middle and bottom pictures, respectively. In all but the last example, the latter set indicate the interruption of an iterative process after different numbers of iterations.

The first example (Fig. 1) is related to crystallization, etching, annealing, and nucleation on a substrate (i.e. simultaneous environment-dependent sublimation and crystallization) [1]. The computation starts with either 1/2 or 1/3 white spots (A's) on a black background (0's). The program then *agitates* by turning black (A's to 2's) one-sixth of the white spots above, below, right of or left of black ones, and then turns white (C's) one-sixth of the black spots next to white ones. (Next for computational purposes, all blacks are recoded as 0's and whites as A's). Then the program *coalesces* by turning white the black spots with predominantly white neighborhoods (i.e., where 3 or 4 of the 4 orthogonally adjacent spots are white), and by likewise turning black the white spots in predominantly black neighborhoods. The process iterates, performing the coalescing operation twice after each agitation.

Other potential uses of EXPLOR are the generation of highly detailed patterns for experiments on vision [2], and simulation of some of the *visual phosphenes*, such as squirming checkerboards or fringe patterns, which a person sees when he closes his eyes and presses upon them.

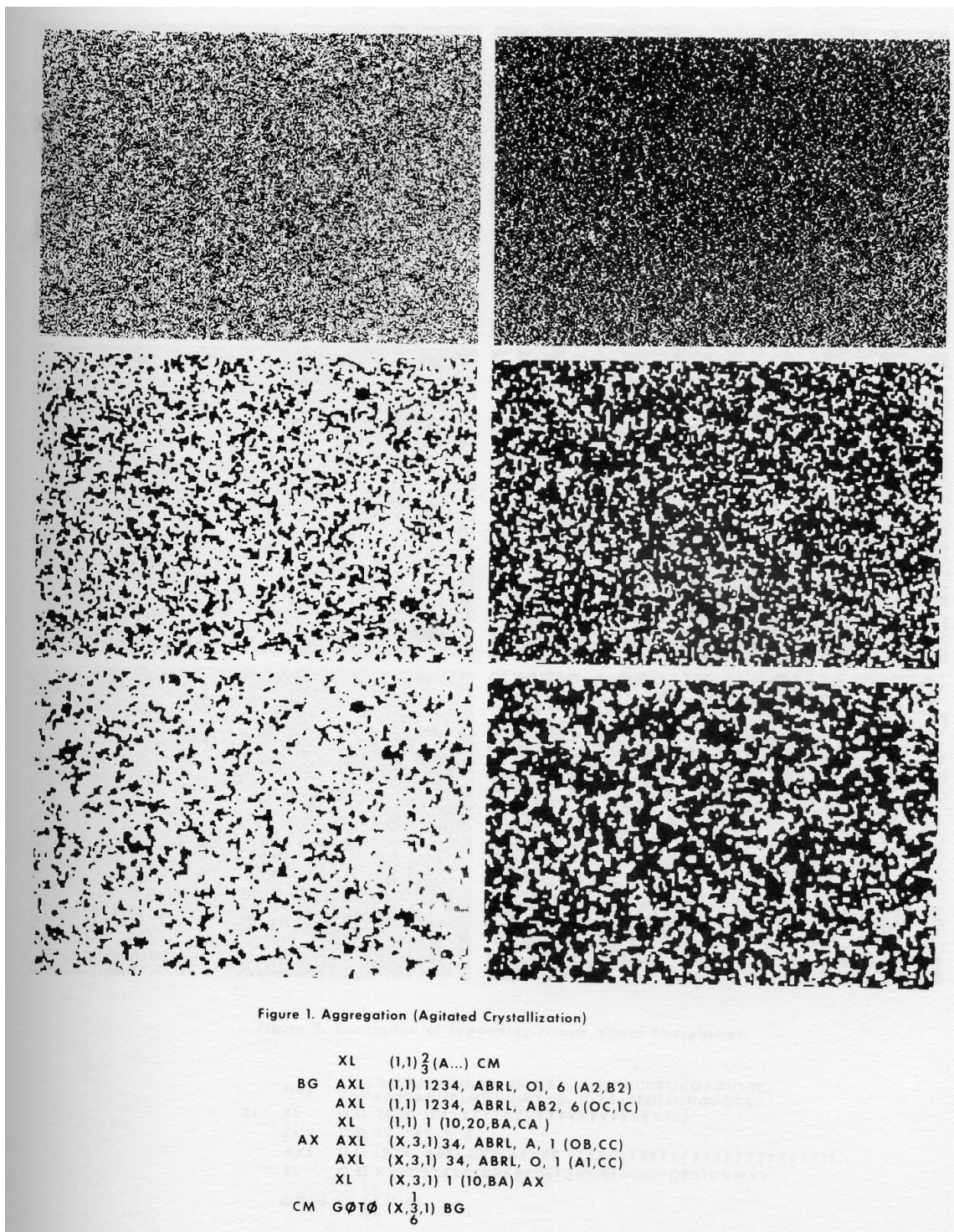


Figure 1

Figure 2 shows a program for generating expanding fringes from randomly placed nuclei (Y's). The area around a nucleus expands at each step to include all of, or half of, or one-third of the adjacent spots, this probability being chosen at random for each iteration. Each spot progresses backward through the alphabet as it ages, according to the transliteration (00123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ). Spots represented by N, and older ones, are eligible, with 1/3000 probability, to become new nuclei for expanding fringes. The left-hand and right-hand outputs show fringes which are three and six iterations thick, respectively.

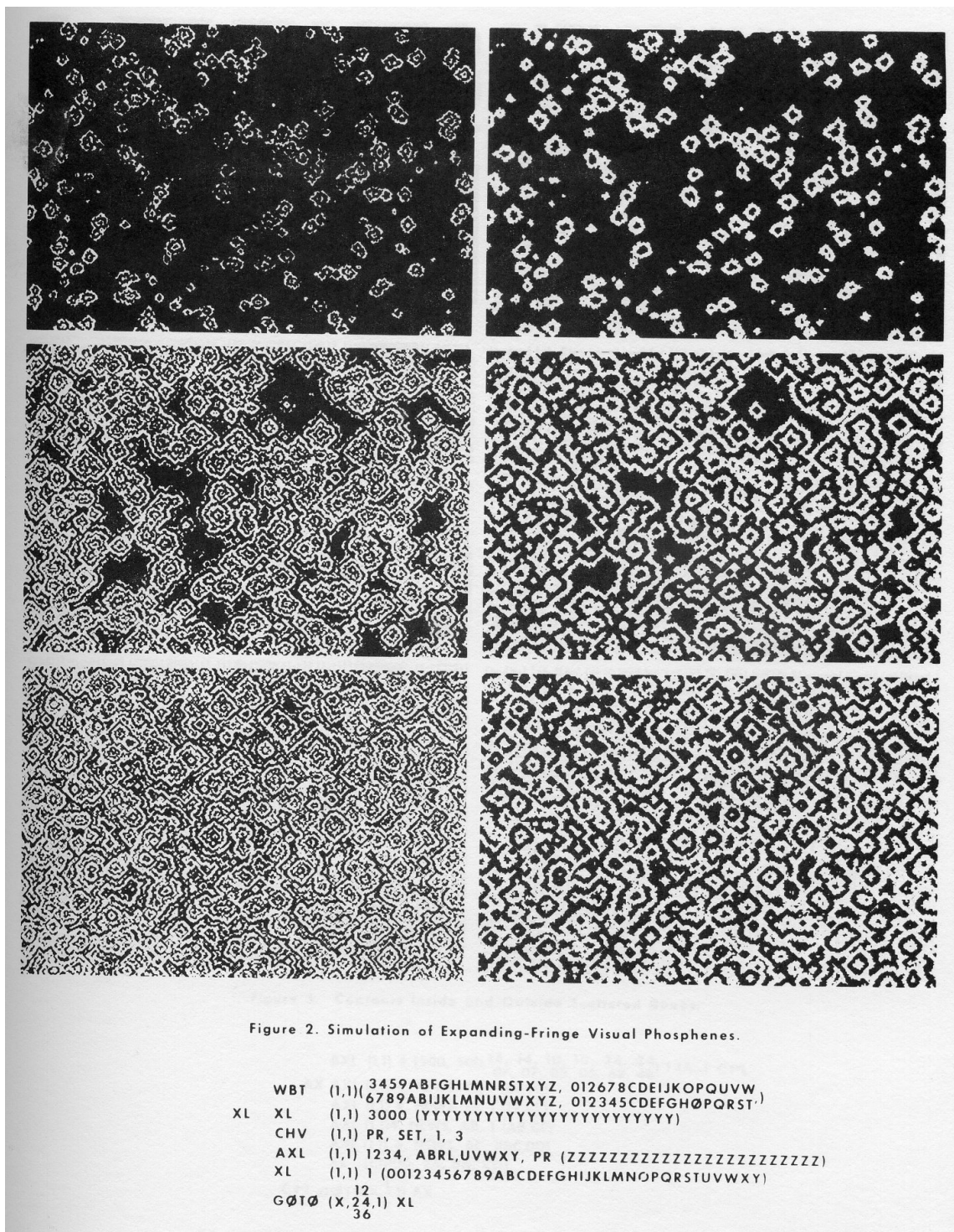


Figure 2

Figure 3 simply illustrates the production of interesting designs by starting with a probabilistic selection from an array of overlapping boxes. Into solid black areas, white and black layers are successively defined; into white areas the converse.

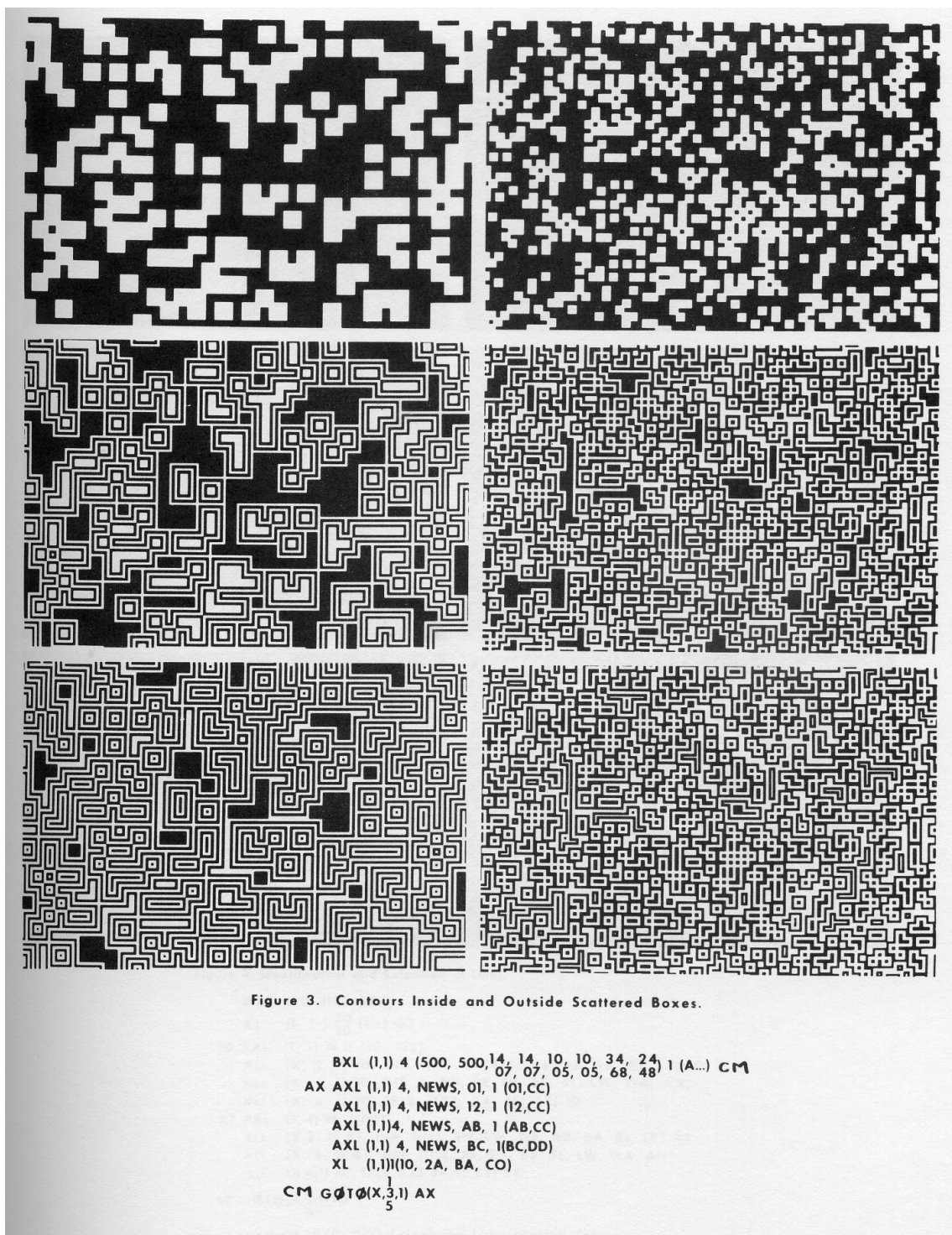


Figure 3

The example of Fig. 4 is related to picture processing; it demonstrates the use of the PXL instruction in identification of and extension of short line segments [4]. Starting with a scattering of 1/10 or 1/20 white (1's in this case) on black, the first PXL instruction changes 1's with 1's (or 2's) north of them to 2's. The result is that any north-south diagonal lines, except for the north end, are changed to 2's. The next instruction flips the direction in the PXL in order to catch the unchanged end. The following two instructions change the direction, by 4-5° clockwise, of the line treated and transliterate the triples of the PXL so that horizontal lines get changed to 3's, east-west lines to 4's and vertical lines to 5's. Whereas the first PXL and the three following XLI's serve to identify endpoints of the 4 orientations of lines, the second PXL and its following XLI's serve, in similar manner, to extend the line segments one unit.

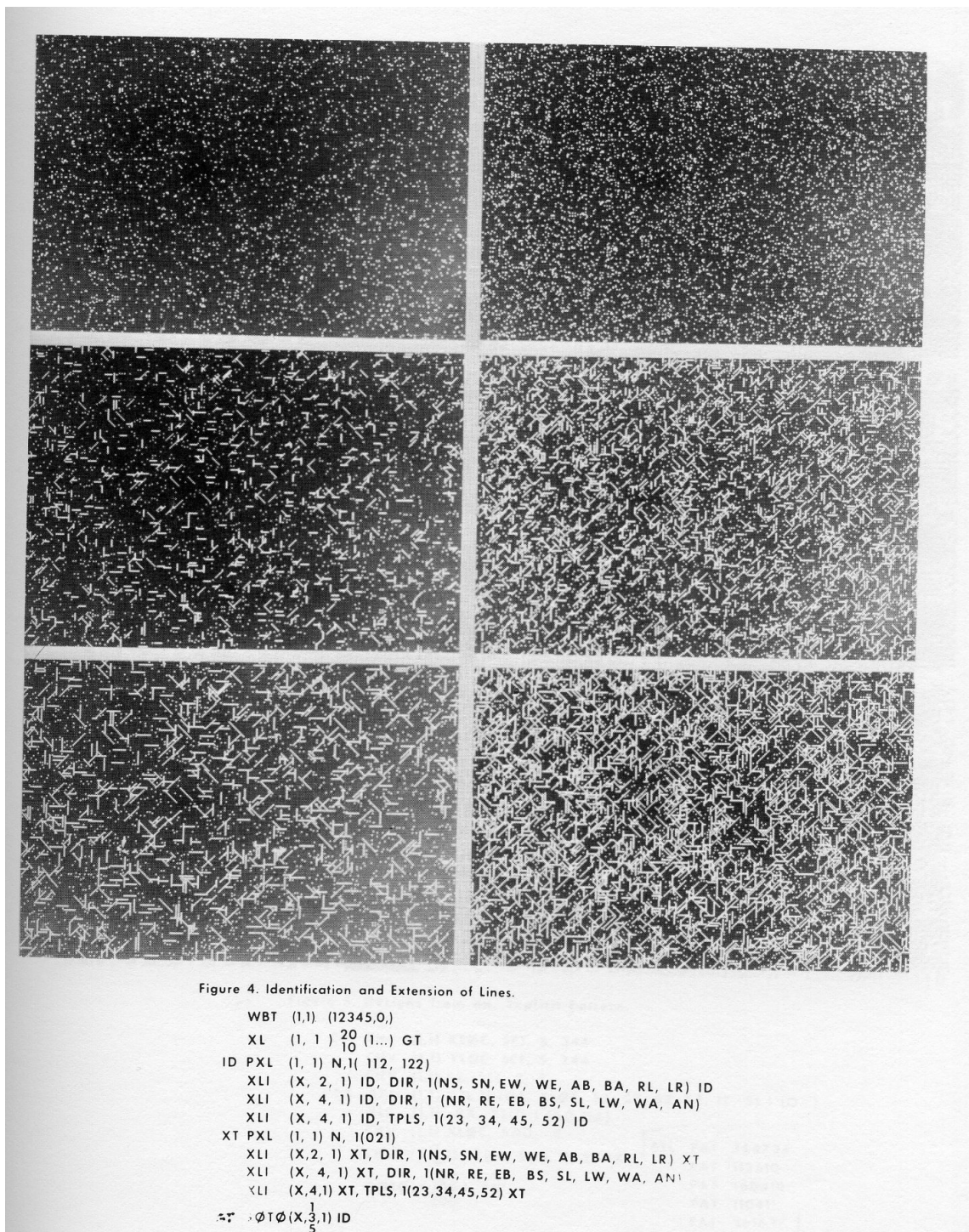


Figure 4

Figure 5 illustrates the production of designs based on an explicitly defined pattern, in this case the letters **BTL**. Each time the pattern is laid down, an x-y location is chosen at random, also a size from 2 to 5. The pattern is first written in 0's, then displaced downward and to the right and written in A's. In the left-hand outputs only one black-white pair is written each time; in the right hand set, four successively displaced black-white pairs are overlaid each time.

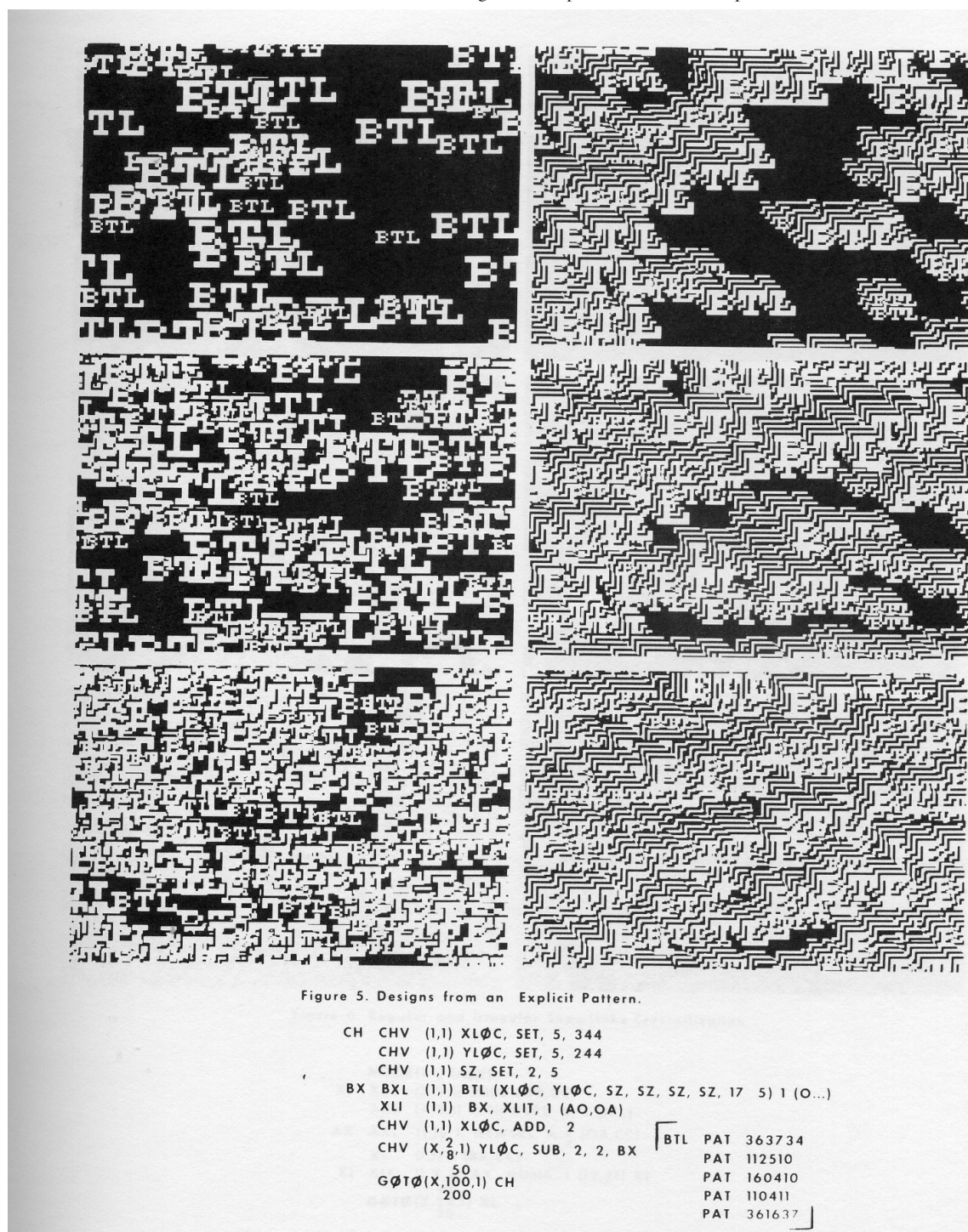


Figure 5

Figure 6 demonstrates computation and output for the hexagonal mode, used here for simulation of snowflake crystallization. Each snowflake starts from a nucleus of a B totally surrounded by A's; growth is either regular or probabilistic, and into spots adjacent to either one or two crystallized spots, the latter choice being determined at random each time through the loop.

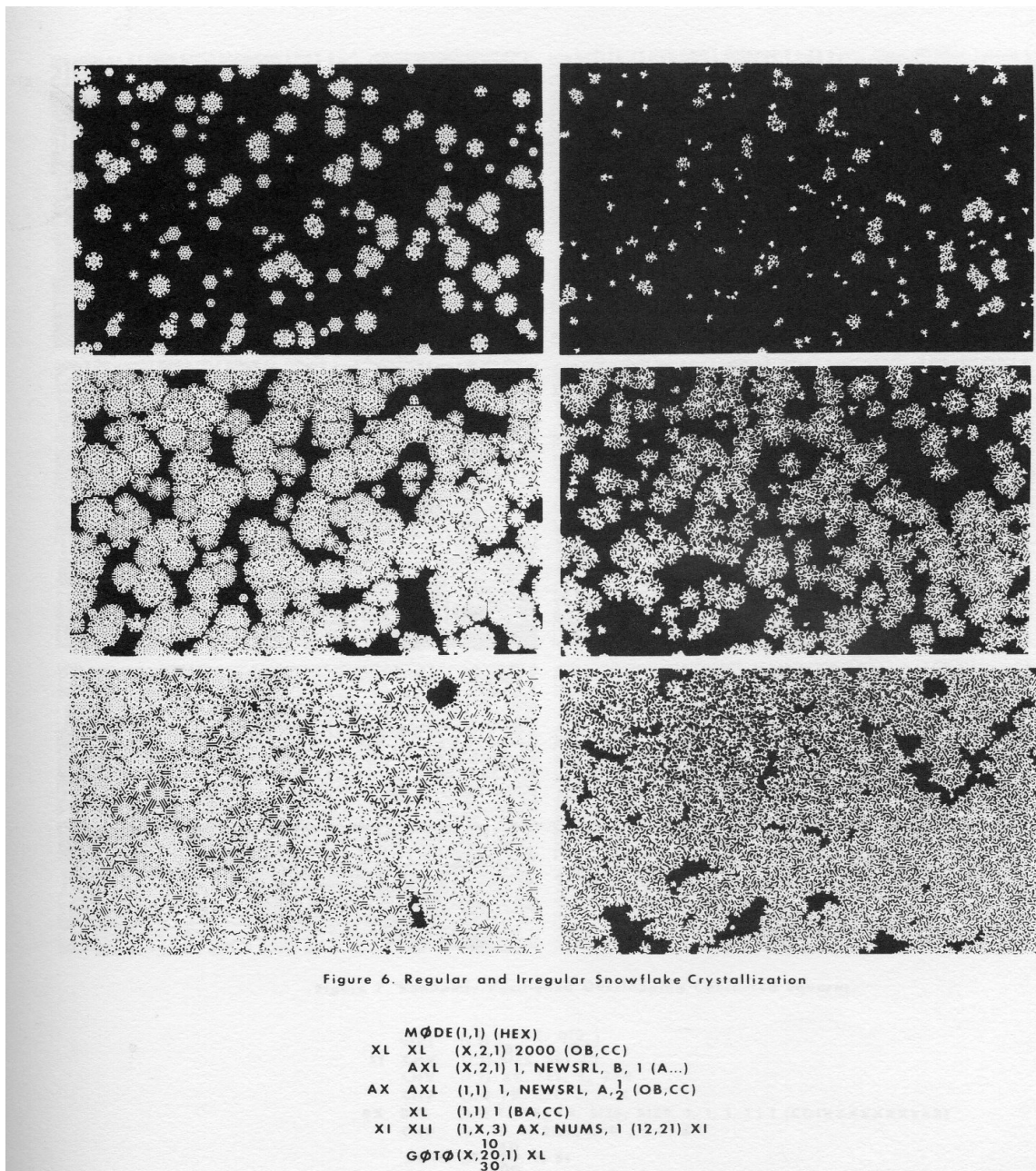


Figure 6

Figure 7 shows further artistic effects obtained by cyclic transliteration of randomly positioned sets of squares within squares. Where sets overlap, more complicated and subtle patterns emerge. The left-hand illustrations are similar to scenes from a film produced largely by the BXPLOR system [5]; the right-hand illustrations demonstrate the use of [twinkling](#) spots in outputs.

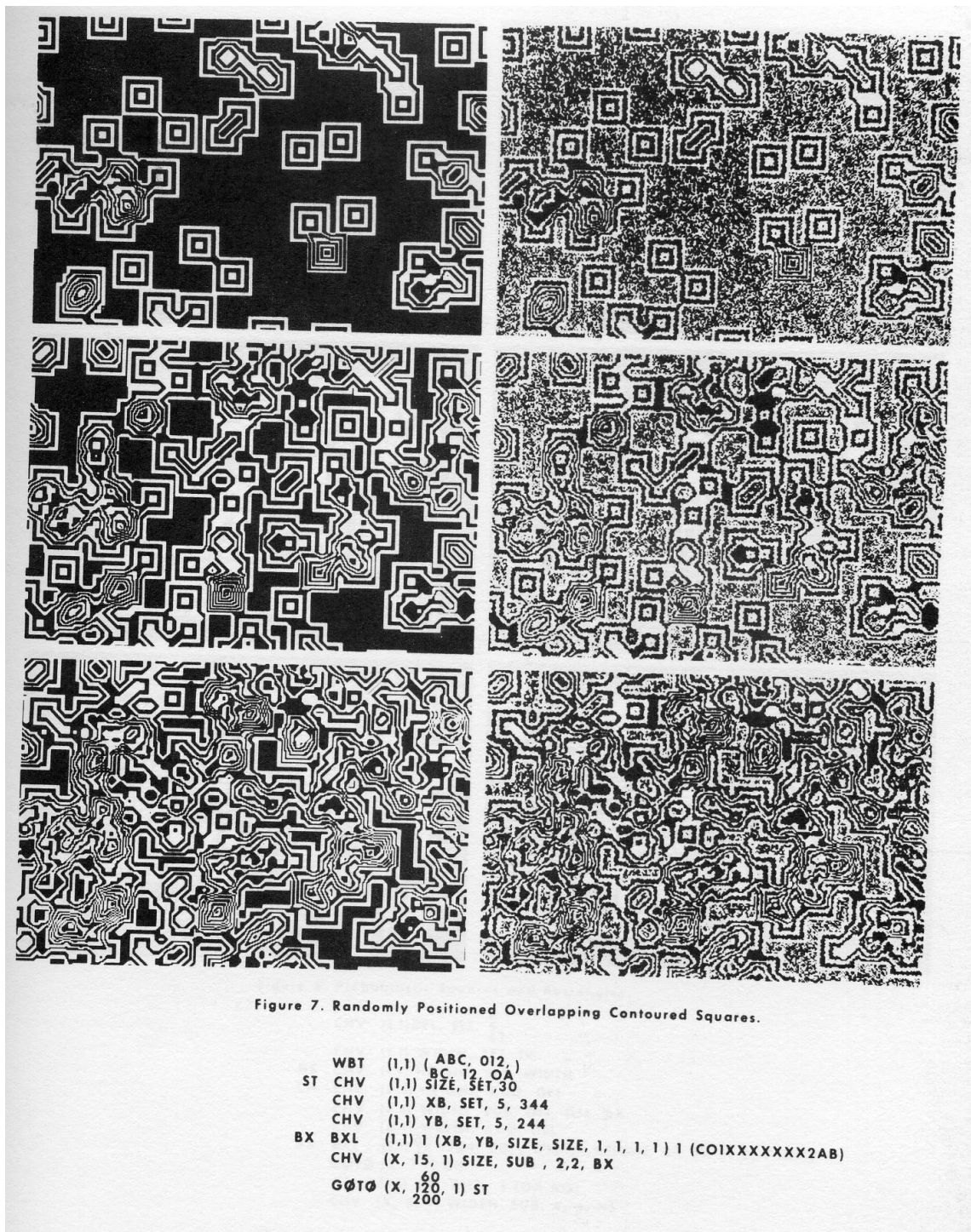


Figure 7

The last example, Fig. 8, is another demonstration of artistic effects, generated here simply as a probabilistic collection of squares and rectangles. The forms are drawn in five successively smaller sizes, alternating between black and white.

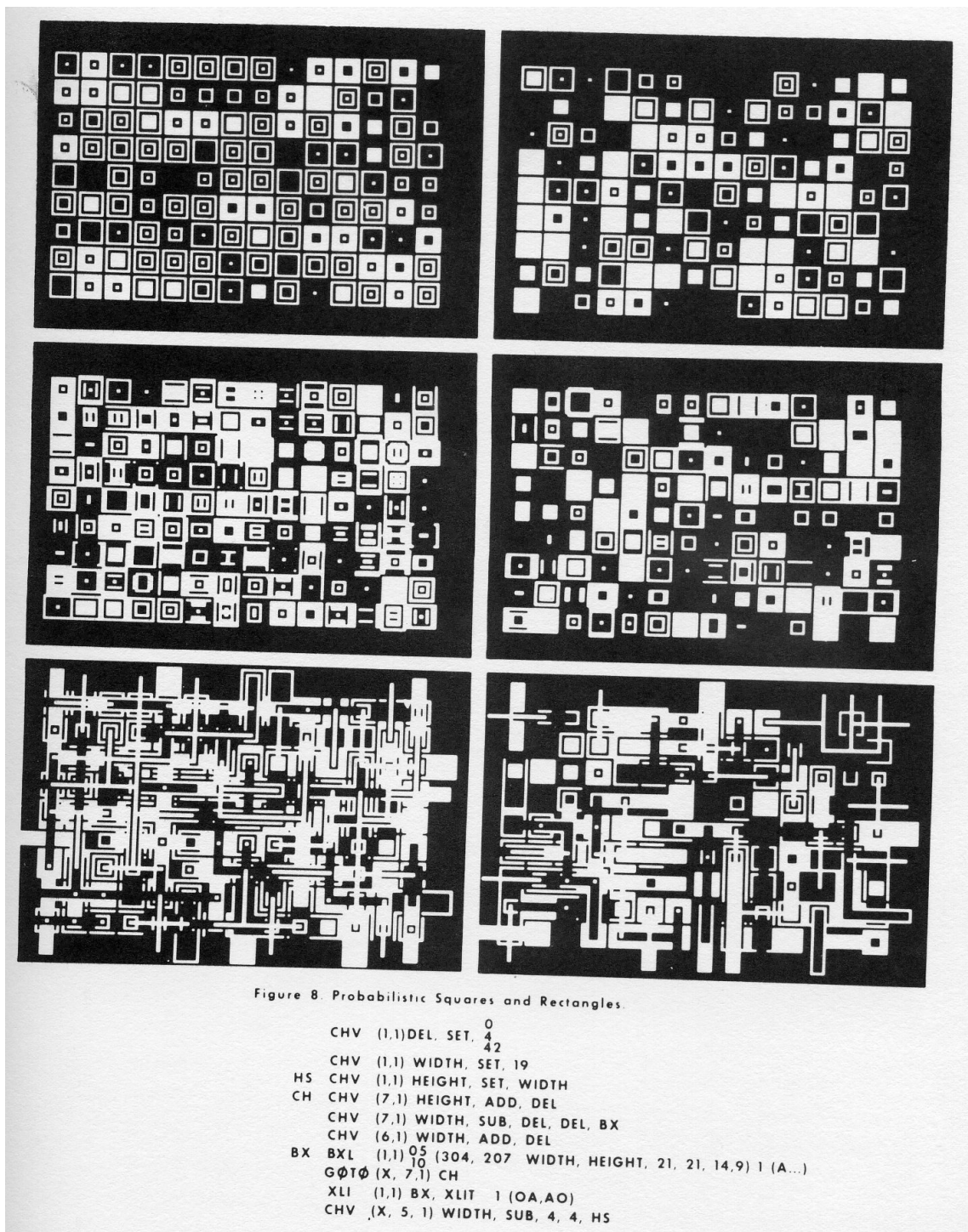


Figure 8

III. TECHNICAL DETAILS

The prototype version of EXPLOR was implemented, because of historical reasons, in BE-FAP [6] on an IBM 360/50 emulating an IBM 7094 (with hardware-implemented `convert` instructions); it occupies 420008 locations, which includes the internal picture storage. The output device is a Stromberg DatagraphiX 4060, using a 4020 simulator modified in the following ways: right and left margins were extended giving an effective raster of 1024×1366 ; the `period` is output as the largest plotting dot, and character spacing in typewriter mode is changed to four 4020 units.

Additional facilities, and also restrictions, implied by these circumstances are discussed in the following paragraphs

1. Software-imposed restrictions

All of the conventions of Bell Labs macro FAP apply as concerns instruction names and gotos, blanks, op code and argument positions, continuations on next card, comments, and macro-extension of the language. Variables are restricted to positive integers from 0 to 32767. Furthermore, there are limitations on character strings and numbers of arguments as follows:

- pairs defining a transliteration: at most 17
- triples in PXL or BPXL: at most 63

- numbers of neighbors in AXL and BAXL: at most 12 - if there are more than 8, clearly some are redundant (ineffectively) or not meaningful until changed by XLI
- directions: at most 12 - some likewise meaningless until changed by XLI.

The order of computation on the internally stored picture is as follows: for the sake of speed, an entire machine word (6 characters) is treated **simultaneously**; changes within a word do not alter the effective environment of other characters in that word. Effects can, however, propagate from word to word: order of computation within a box (or the whole frame) is bottom line first, left to right. The order of treating boxes of an array is top line first, right to left.

2. Computation time

Actually measured run times for various operations and circumstances are as follows:

- outputting a full frame of twinkles 2.9 sec, outputting a full frame, no twinkles 2.4 sec max (the latter scales downward linearly, by omitting full words of blanks from both ends of lines, to :) outputting a blank frame .5 sec
- XL, full frame $pr = 1$, .3 sec. $1/2 \geq pr \geq 1/74$, .6 sec. $pr < 1/74$.6 sec $\times$ % of words treated
- PXL, AXL with one direction (more, as in (b) if $pr \neq 1$), 1 sec. per additional direction in AXL, .4 sec. (e.g. AXL, all eight directions 3-8 sec).

For boxed-array operations (BXL, BAXL, and BPXL) figures given for (b) and (c) decrease in proportion to the area treated; for test mode operation, all times are cut by a factor of ten because only a small part of the surface is computed.

All other operations are essentially preparatory the time which they consume is insignificant compared with those given above.

IV. SUMMARY

The EXPLOR system is a convenient, versatile, and efficient system for generating scientific and educational as well as artistic displays. All pictures generated derive from explicitly defined patterns, local operations and randomness.

The system has unique facilities for specifying periodic and/or random application of its operations, and flexible means of specifying uniform or locality-dependent translation of internal symbols.

The current version has a particularly fast-running scheme for spot-by-spot randomization; in this and most other respects it should serve as a good model for other implementations.

REFERENCES

1. A. J. W. Moore, [Nucleation of Solids from the Vapor Phase](#), J. of Australian Inst. of Metals II, No. 4, pp. 220-226, (November 1966).
2. B. Julesz, [Computers, Patterns, and Depth Perception](#), Bell Laboratories Record 44, No. 8, pp. 261-267 (September 1966).
3. G. Oster, [Phosphenes](#), Sci. Am. 222, No. 2, pp. 83-87 (February 1970).
4. L. G. Roberts, [Machine Perception of Three-Dimensional Solids](#), Technical Report No. 315, M.I.T. Lincoln Laboratory, Lexington, Mass., (May 1963).
5. [Pixillation](#), a 5 min. color, 16 mm film by Lillian Schwartz and Ken Knowlton, sound by Gershon Kingsley, 1970. Produced for and distributed by AT&T, Attn: Martin Duffy, 195 Broadway, New York, New York.
6. 7094 Bell Telephone Laboratories Programmer's Manual, Bell Telephone Laboratories, Inc., Murray Hill, New Jersey (1963).

More Computer Animation Papers 1964-1976