SGI: An Interpreter for Shape Grammars

by

Ramesh Krishnamurti

Centre for Configurational Studies
Design Discipline
The Open University
Milton Keynes MK7 6AA
ENGLAND

Abstract

The design of the computer program SGI (pronounced sigi) for the generative specification of shapes using the shape grammar formalism (Stiny, 198Ø) is outlined. The program interprets commands from the SGI language. The commands which are described in detail allow for shape grammars to be defined interactively, and for shapes defined by these shape grammars to be generated via user-specified sequences of shape rule application. The program has graphics capability which permits line drawings of shapes to displayed on the screen and drawn on the graphics plotter. Examples of shape grammars defined using this program, and plotter line drawings of shapes generated by the grammars are illustrated.

Introduction

This paper outlines the design of an interpreter for shape grammars (Stiny,198Ø). The interpreter, SGI (pronounced sigi), provides a computational framework for languages of shapes to be defined using the shape grammar formalism, whereby shapes can be constructed by applying rules rather than be described point by point or line by line as is the case with current computer programs. SGI is primarily intended as a research and/or teaching tool.

SGI takes the form of an interactive computer package with graphics facilities. The package:

(a) accepts any shape grammar definition and any subsequent on-line modifications;

(b) applies any user specified sequence of rule applications and has backtrack capability;

(c) provides graphics (visual displays), hard copy (plotter drawings) and line printer descriptions of any partial or complete shapes generated at any stage of rule application;

(d) stores any shape or shape grammar defined at an interactive session and retrieves any shape or shape grammar so stored later in the session or at any later sessions.

Instructions to the computer program are specified through commands from the SGI interpreter language.

The present version of SGI is restricted to 2-dimensional shapes and shape grammars. This does limit its use in practical computer aided design. It is hoped that SGI will be extended, and if necessary upgraded, to enable 3-dimensional shapes and beyond the scope of this paper. It is sufficient to state that solutions to these problems are numerical rather than spatial.

SGI is implemented in Fortran and runs on the VAX 11/780. The shapes are displayed on a Tektronix 4027 colour graphics terminal and hard copy of the line drawings are obtained using the Tektronix 4663 graphics plotter. Colour photographs of the displayed shapes are obtained using the Calcomp31 (Dunn) colour graphics system.

It should be remarked that the graphics facilty provided for the program has been kept simple and that it satisfies the basic requirements for any shape grammar interpreter. However, enhancements and/or modifications to the graphics routines, to suit the particular needs of potential users, can be easily implemented.

The rest of this paper is organized as follows. First, a brief review of shapes and shape grammars is presented, and the relevant terms are stated in a form suitable for implementation on a computer. Second, the data structures used by the program to house the shapes are described. Third, the SGI language (or command sets) and the structure of the program are described. Last, examples of shape grammars run using SGI are illustrated.

Shapes and shape grammars

A good concise introduction to shape grammars is given in (Stiny, 1980). Still, it is worthwhile to restate some of the concepts in a form suitable for computer implementation.

A labeled shape consists of two parts: a shape made up of lines and a set of labeled points made up of symbols.

A shape is given by its set of maximal lines. A line is is described by its end points each of which is associated with a pair of coordinates in a two dimensional cartesian coordinate system. A line l contains another line m if the end points of m coincide with points on l. A line in a shape is maximal if no other line in the shape contains it.

The program stores and permits references via line numbers to only the maximal lines in a shape, though any line can be described to it. The program checks whether the specified line is maximal or not. In the event that the line is not maximal, it will be combined with one or more maximal lines in the shape to form a longer maximal line. This new maximal line replaces all the lines that have been combined. Thus, specifying a line to the program may decrease the number of maximal lines that are required to represent the shape.

A labeled point is described by a pair of coordinates and its associated set of labels. In general, a label may be any symbol from a specified alphabet. The program only accepts labels chosen from the letters in the set {A,...,Z} with the following restriction: each label associated with a labeled point must be distinct. Thus, a labeled point cannot have, for example, the label AA. In this case, the program treats it as the labeled point with the label A. The labeled points are referenced by their point numbers.

The label alphabet is the set made up of just those labels in the shapes in a shape grammar.

Labeled points are shown on the display screen with their labels arranged in lexicographical order. For example, if a labeled point has a label SYMBOL, it is displayed as a labeled point with label BLMOSY. Moreover, if two labeled points with distinct labels are specified to the program, and the labeled points share the same coordinates, the program regards them as a single labeled point whose label set is given by the union of the two label sets. For example, if a labeled point has the label A and another has the label B, and if they share the same coordinates, the program regards them as a single labeled point with the label AB.

A point p is <u>rational</u> iff each of its coordinates can be expressed as the ratio of two integers. A labeled point p:L, where L denotes it label set, is <u>rational</u> iff p is rational. A line l=<t,h>, where t and h are the end points of l is <u>rational</u> iff both t and h are rational. A labeled shape is <u>rational</u> iff each of its maximal lines is rational and each of its labeled points is rational. The program accepts only definitions for rational labeled shapes.

A labeled shape may have no lines or no labeled points. A <u>null</u> (or <u>empty</u>) labeled shape contains no lines and no labeled points. Where convenient, a labeled shape will be referred to, simply, as a shape.

A shape is a <u>subshape</u> of a shape if each maximal line in the first shape is contained in a maximal line of the second shape, and if for each labeled point in the first shape there is a labeled point in the second shape which shares the same coordinates and whose label set contains the label set of the first labeled point.

The transformations of translation, rotation, reflection, scale and finite compositions of these are referred to as the euclidean transformations, and denoted by T. A transformation T can be described as the pair of transformations:

$T = \langle T_x, T_y \rangle$

and represents the coordinate-coordinate mapping:

$T : \langle x,y \rangle \longrightarrow \langle T_x(x,y), T_y(x,y) \rangle$.

Both T_x and T_y can be expressed as linear polynomials over the rationals (see Krishnamurti, 1980), having form:

$a\,x + b\,y + c$.

The coefficients a,b and c are dependent on T.

The transformation T of a labeled shape σ, $\sigma = \langle s, P \rangle$, where s is the shape and P is its set of labeled points, is the labeled shape $T(\sigma) = \langle T(s), T(P) \rangle$ which is obtained by changing the spatial disposition and/or size of σ. More precisely, $T(\sigma)$ is defined as follows. Let $p = \langle x,y \rangle$ denote a point. Then,

$T(p) = \langle T_x(x,y), T_y(x,y) \rangle$

$T(s) = \{ \langle T(p_1), T(p_2) \rangle$ is a maximal line in $T(s) \mid \langle p_1, p_2 \rangle$ is a maximal line in s $\}$

$T(P) = \{ T(p){:}A$ is a labeled point in $T(P) \mid p{:}A$ is a labeled point in P $\}$

In other words, T takes each point, maximal line, and labeled point in the labeled shape σ to a corresponding point, maximal line, and labeled point in $T(\sigma)$.

A shape grammar is an algorithm described in terms of labeled shapes. In its standard form it consists of some shape rules and an initial shape. A shape rule: $\alpha \longrightarrow \beta$ consists of two shapes: α, the left side shape and β, the right side shape. The shape rule can be used to change a given shape, the <u>current</u> <u>shape</u> γ , into a new shape γ^* , whenever there is a similarity transformation T, that makes α a subshape of the current shape. In this case, the subshape is replaced by the same transformation of β. That is, γ^* is given by the shape expression:

$$\gamma^* \longleftarrow \gamma - T(\alpha) + T(\beta)$$

The shape rules are applied to the initial shape and to shapes produced from it. In this way, the <u>language</u> of shapes defined by the shape grammar, is obtained by generating its individual members.

Each shape rule is type classified according to the total number of distinct labeled points and points of intersection of the maximal lines in the left side shape, α. Let this number be denoted by N. A shape rule is of <u>type</u>:

1: if $N > 2$ and the points form a triangle;
2: if $N > 1$ and the points are all colinear;
3: if $N = 1$ and the point is not coincident with any maximal line in α or its extension;
4: if $N = 1$ and the point is coincident with all the maximal lines in α or their extensions;
5: if $N = \emptyset$.

The shape rule type plays an important role in rule application (see Krishnamurti, 1981).

In general, shapes can be constructed on any 2-dimensional rational cartesian plane. However, for computational reasons it is necessary to restrict the region in which the shapes are defined. The finite region on the rational cartesian plane is referred to as the <u>user</u> <u>window</u>. The program expects all shapes to be defined within this region.

The user window is mapped directly onto the display screen and each shape, defined with respect to this window, is mapped onto its corresponding line drawing on the display screen by the same transformation.

The user window can be redefined at any stage in the program. There is no restriction on the size of the window except that it be finite. At the outset, the user window has the same coordinates as the graphics display screen, which in the case of the Tektronix 4Ø27 is (Ø,Ø) to (619,419). That is, the origin is situated at the left hand bottom corner of the screen.

Figures 1 through 3 provide examples of shape grammars that can be defined using the program. Notice that the grammar in Figure 1 employs the symbol . as a label instead of a letter and the grammar in Figure 2 employs the shape as a "marker" which serves the same function as a labeled point, namely, to guide the generation process. In latter case, the euclidean transformations also apply to these marker shapes. These shape grammars will be considered later in the paper as examples of shape grammars that can be defined using the program.

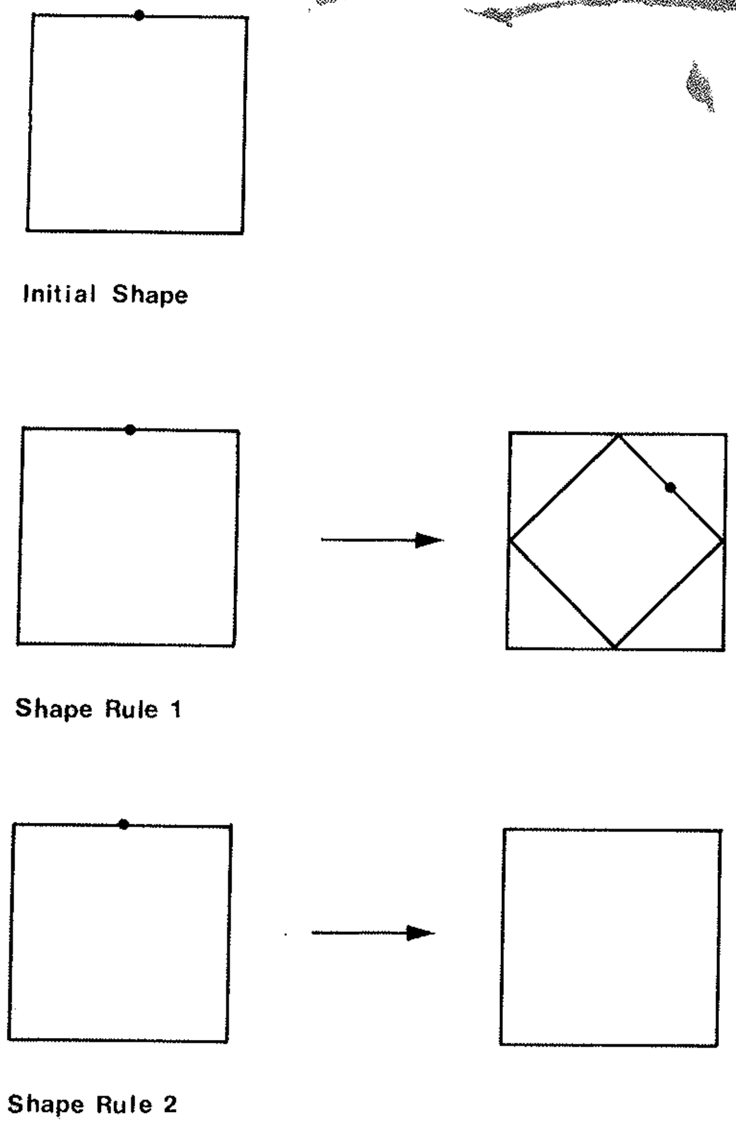
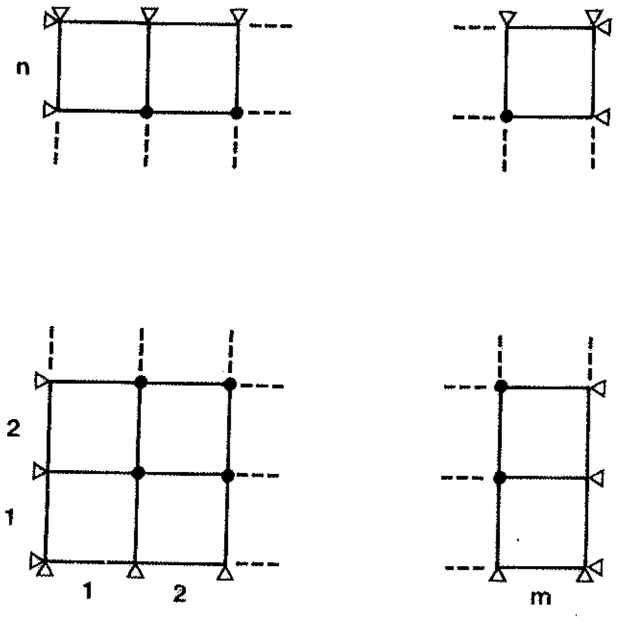
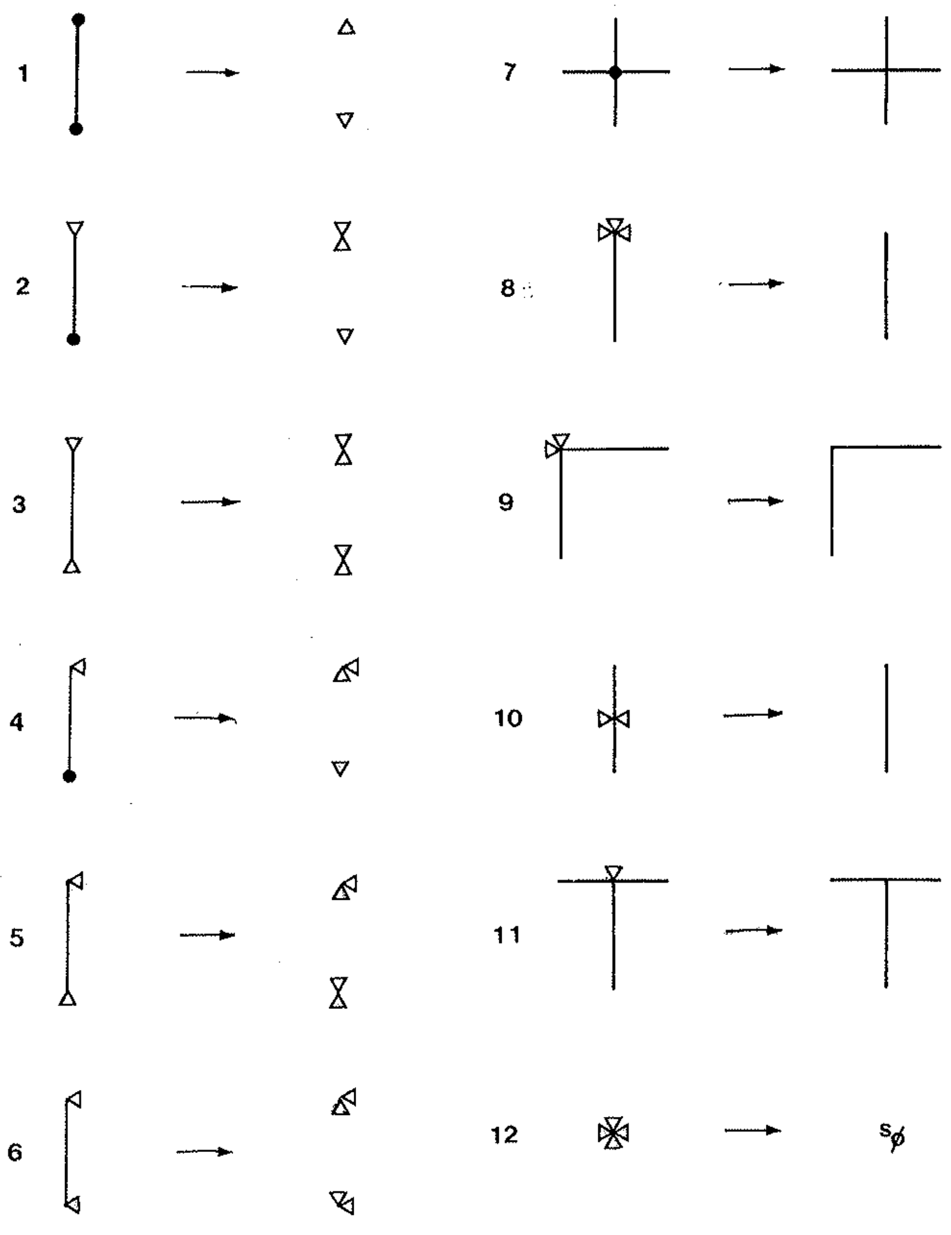


Figure 1: A simple shape grammar that inscribes squares within squares (from Stiny, 1980)



Initial Shape : An (m,n) rectangular grid

Figure 2: A shape grammar to generate rectangular shapes (from Earl, 198Ø)



(a) Line erasing rules

(b) Marker erasing rules
s_ϕ denotes the empty shape

Figure 2 (continued)

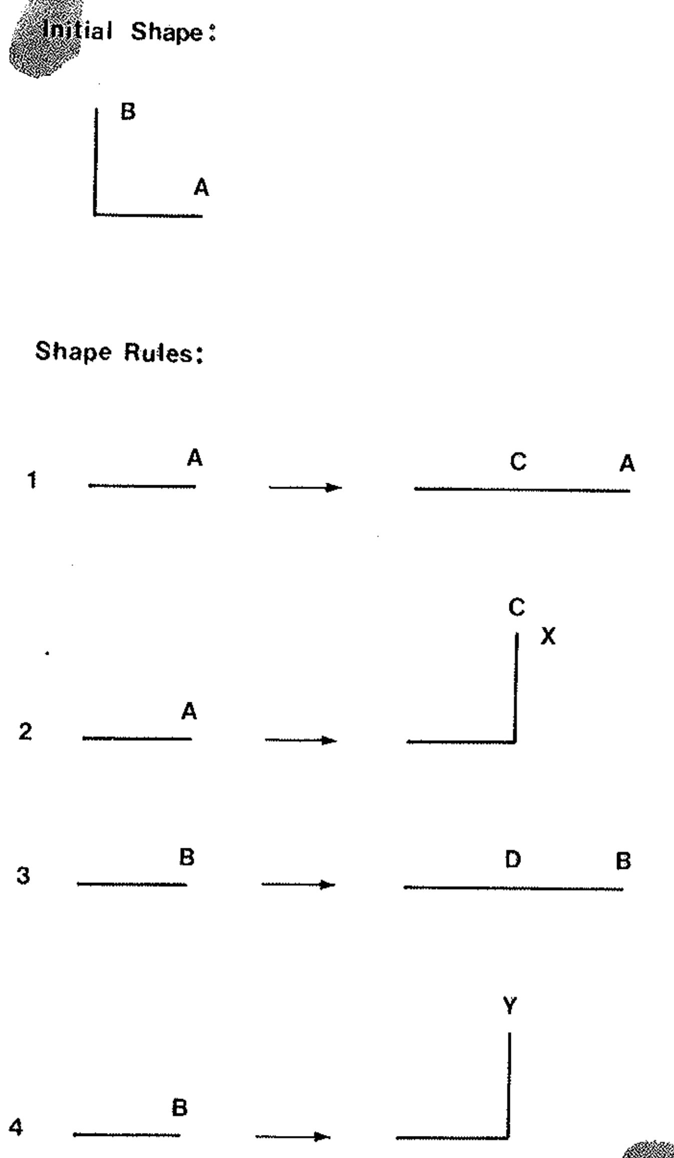


Figure 3: A shape grammar to generate grids

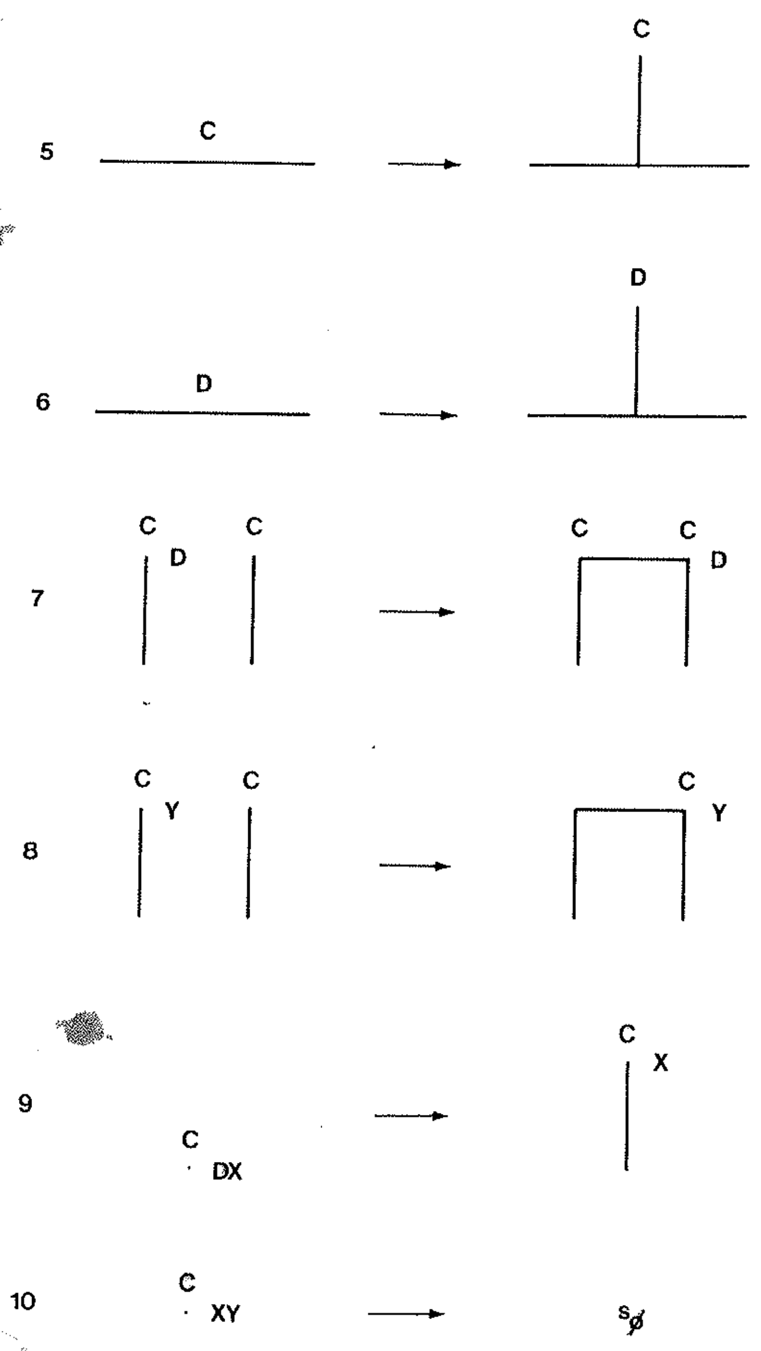
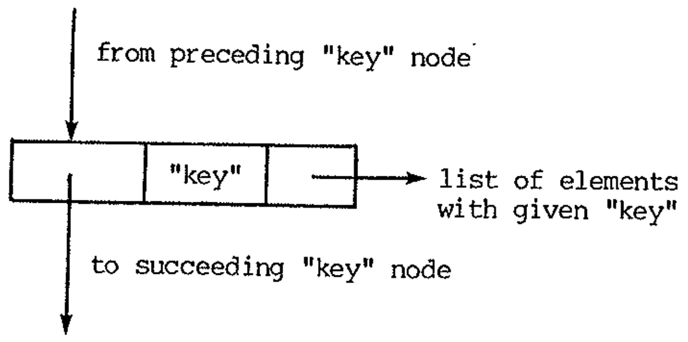


Figure 3 (continued)

Shape representation

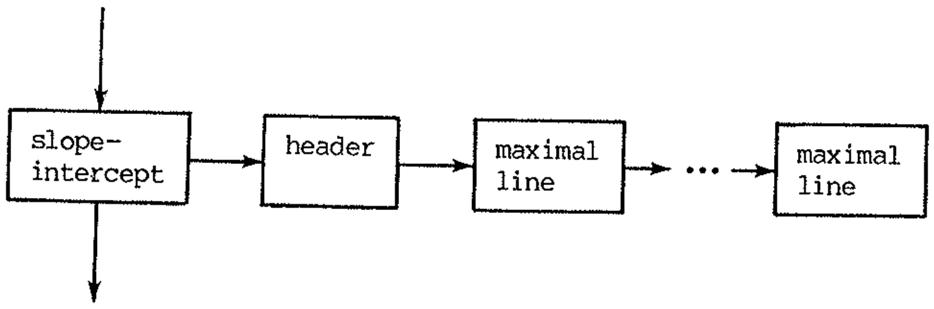
The data structures employed by the program, to store the shapes and shape grammars and to drive the shape algorithms for the boolean operations and relations on shapes are detailed in (Krishnamurti, 1980, 1981).

A shape is represented by a pair of list structures, one to represent the maximal lines and the other to represent the labeled points. Each list structure is arranged according to some prescribed "key" values. The general arrangement of the lists take the form shown below:

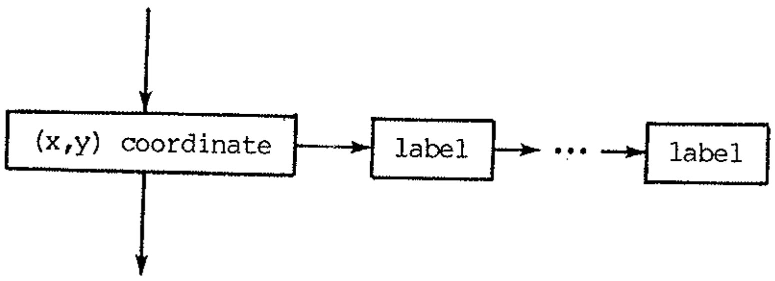


Consider first the maximal lines. The lines are arranged in increasing order of their slope-intercept values. If two lines are <u>colinear</u> - that is, they share the same slope and intercept values - they are arranged according to the (x,y)-coordinate values of their <u>tails</u> (Krishnamurti, 1980). For two colinear lines, $l_1 = <t_1,h_1>$ and $l_2 = <t_2,h_2>$, l_1 <u>precedes</u> l_2 iff $(x,y)[t_1] < (x,y)[t_2]$. For any maximal line $l = <t,h>$, where h is the <u>head</u> of the line, $(x,y)[t] < (x,y)[h]$. Since colinear maximal lines are disjoint, $(x,y)[h_1] < (x,y)[t_2]$.

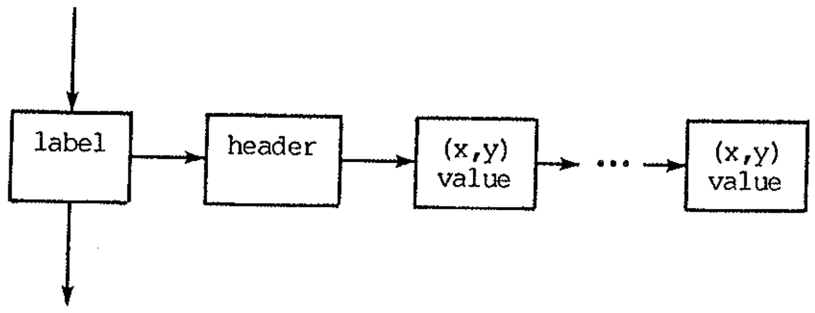
The slope-intercept values are arranged in a linked list, each of whose nodes represents a distinct slope-intercept. The nodes contain two link fields, one which points to the "next" slope-intercept node in the list and the other which points to a linked list that represents the colinear maximal lines with the given slope and intercept. The list of maximal lines may have a "header" node which contains additional information. The node representing a line l=<t,h> contains one link field which points to the "next" line in the list and two fields that point to t and h respectively. The lines are numbered according to their position in the list structure. The data structure for the lines is shown below:



Now consider the labeled points. These are represented by two complementary list structures, one to display the labeled points and the other is used by the shape algorithms. The list structure for display purpose consists of a linked list each of whose nodes represent the coordinates of a point and contain a a pointer to the list of labels associated with the point. The labeled points are arranged in increasing value of their (x,y)-coordinates. The labeled points are numbered according to their position in the list. The data structure is shown below:



For the list structure used by the shape algorithms, the label alphabet is represented by a linked list, each of whose nodes represents a label. Each node contains a pointer to the list of coordinates of labeled points in which the given label occurs. The labels are arranged in increasing lexicographical order. The list of coordinates may have a "header" node. The data structure is illustrated below:



The data structures for the labeled points cross-refer to each other. The display list structure for the labeled points is created only when required except for certain shapes which are displayed frequently.

Rational numbers are stored as pairs of integers reduced to their primitive form (Krishnamurti,198Ø). Since most of the rational values used by the program occur in pairs such as the (x,y) coordinates of a point or the slope-intercept value for a line, an array with four integeral fields is employed by the program. Thus, for example, if a list node represents a maximal line, its two information fields, one for the tail of the line and the other for the head of the line, contain references to entries in this array.

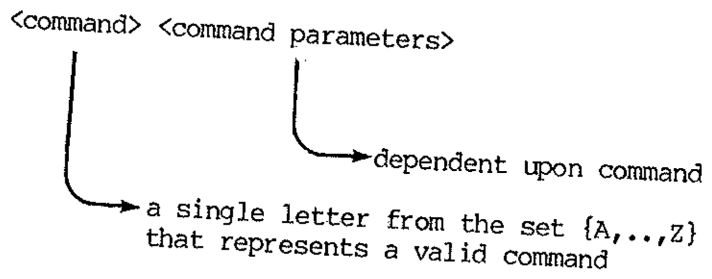
Transformations are represented by nodes with three fields each of which points to a pair of rational numbers. Each field represents two coefficients, one for the x-coordinate transform

T_x and the other for the y-coordinate transform T_y. [Recall that the linear polynomial form for both transformations each contain three coefficients.] That is, the transformation,
$T = <T_x = <a_x, b_x, c_x>, T_y = <a_y, b_y, c_y>>$
is stored as the triple of pairs: $<(a_x, a_y), (b_x, b_y), (c_x, c_y)>$.

The program has predefined limits on the amount of memory made available to store the shapes, pairs of rational numbers, labeled points, shape rules, and transformations. These limits are given by to the maximum numbers: of list nodes available to to store the list structures and transformations; of data nodes available to store the pairs of rational numbers; and of rule nodes available to store the shape rules. These limits can be easily altered (though this requires recompiling the program) to suit the requirements of users who wish to work with large shape grammars. If these limits are exceeded, the program will initiate an error recovery procedure which enables the user to save on disk, any shape grammar that [s]he has constructed thus far.

The Structure of SGI

An overview of the program is diagrammed in Figure 4. The program has two phases (monitor states) which are indicated in the diagram by rectangles. The shape grammar is constructed in the first phase, which is the start-up monitor state. When the program is in this phase, the user is prompted by the string: "SGI>". The shapes are generated in the second phase. In this phase, the program prompts the user using the string: "GEN>". These two phases are referred to as the SGI phase and the GEN phase respectively. Each phase has its own set of commands, its own set of commands some of which are shared by both sets. The commands have a simple structure as shown below:


<command> <command parameters>
dependent upon command
a single letter from the set {A,..,Z} that represents a valid command

Commands to be performed by the program are indicated by typing in a single character together with any additional parameters that may be required. Once a shape grammar has been defined, typing a G command will place the program in the GEN phase. In this phase, the user can apply any sequence of shape rules.

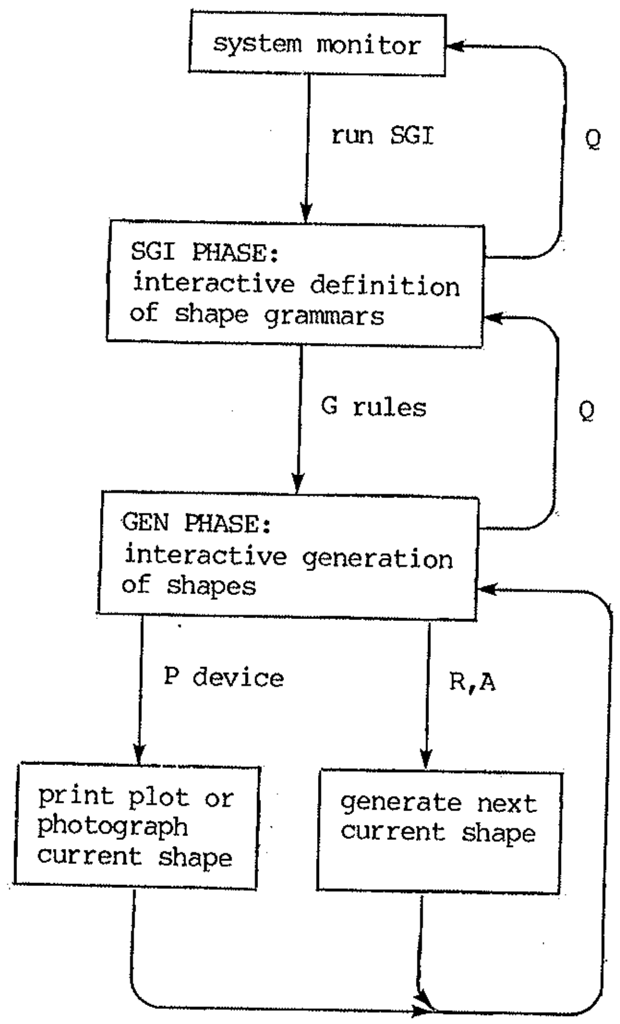

system monitor
run SGI
Q
SGI PHASE: interactive definition of shape grammars
G rules
Q
GEN PHASE: interactive generation of shapes
P device
R,A
print plot or photograph current shape
generate next current shape

Figure 4: An overview of the program structure

Interactive definition of shape grammars

In the grammar construction phase (referred to in Figure 4 as the SGI phase) there are three display shapes which the user has at his disposal to work with. These are:

1. The initial (or current) shape;
2. The left side shape;
3. The right side shape.

The initial (or current) display shape is used both as the starting shape and the current shape for the shape generation process. The left and right display shapes are employed in the construction of the shape rules and are referred to as the rule shapes. Any one of these display shapes can be activated by typing an A command (see the section on the command sets). The activated shape is the shape that is currently displayed on the screen and that the subsequent commands affect. The activated display shape is referred to as the active shape. Initially, each of the display shapes is empty.

Usually, the first step in defining a shape grammar is to set up the user window. This is done by typing in the X and Y commands.

The commands affect either the lines or the labeled points depending upon the mode the program is currently viewing. The mode can be set by typing in a V command.

Commands are available to add or delete lines or labeled points. The lines and labeled points are numbered, any one of which may be chosen as the current line or point, by typing a C command. It is this line or labeled point that is altered by the commands to move, rotate, enlarge, shrink, fasten, label and unlabel (see the section on the command sets).

It should be noted that any command that involves a line or a point operation that results in a line or a point falling outside the user window, will not be carried out. If such a situation should occur, the user window has to be redefined by typing the X and/or Y commands.

In order to define a shape rule, the left and right side display shapes have to be constructed with the aid of the above mentioned line or point commands. Once the sides of a shape rule have been constructed, a copy of (the active shape or) the left and right side rule shapes can be entered as a (side of a) shape rule by typing an E command. Each shape is associated with a <u>rule number</u>. The shape rules may be defined in any arbitrary order.

The shape rules are arranged in the <u>standard</u> form, namely, as the pair: $< \alpha , \beta >$.

Existing shape rules can be replaced or erased. Often, shape rules may share the same left or right side shape. Also, the left side shape of some shape rule may be identical to the right side shape of another shape rule. The F command fetches a copy of (a side of) a shape rule onto (the active shape or) the left and right side rule shapes.

A shape or shape grammar can be saved on disk using the W command and subsequently reinstated using the R command.

When all the shape rules and the initial shape have been constructed, the user can enter the generation phase (described in the next section) by typing a G command.

When the user wishes to terminate an interactive session, typing a Q command, returns the program to the system monitor.

Interactive generation of shapes

In the shape generation phase (referred to in Figure 1 as the GEN phase) the user can apply any of the shape rules in the shape grammar to construct a new shape from the current shape. To enter the GEN phase from the SGI phase, the user must type in a G command. The program first examines if:
(1) the initial shape is non-empty;
(2) the shape rules have non-empty left sides.
If neither condition is satisfied, the user is not allowed to enter the GEN phase. In the event that some shape rules have empty right sides, the program will enquire if the user wishes to proceed with the generation phase.

Shape rules can be applied in two ways, by typing an A or an R command:
(1) Rules can be applied to the first subshape instance found;
(2) Each subshape instance is examined and when the program has found one, enquires from the user if [s]he wishes:
   (a) to apply the rule to the given subshape instance;
   (b) to reconsider the subshape instance after all possible instances have been examined - that is, when all the subshape intances have been considered, the program will "roll" through the list of subshape instances that

have been marked for reconsideration until the user has exercised one of other three options;

(c) to discard the subshape instance;

(d) to quit - that is, the shape rule is not applied to any of the subshape instances.

The algorithm for shape rule application is described in (Krishnamurti, 1981).

When a shape rule has been applied, a record of the rule application is noted in a rule stack, which the user can access at any subsequent stage in the shape generation process. This provides the user with the means to backtrack to an earlier stage in the generation process, which he can do by typing a B command.

In the event that a shape rule application results in the replacement shape extending beyond the boundary of the user window, the program will automatically scale the window by an appropriate quantity so that the generated shape fits into the new window, the coordinates of which are reported to the user.

In order to determine a subshape instance, the program attempts to compute an euclidean transformation such that the transformation of the left side of the shape rule is a subshape of the current shape. If, in the event that there are two such transformations with different coefficients which give rise to identical subshape instances with identical shape replacements, the program will regard the two transformations as equivalent. That is, the program determines the equivalence classes of the transformations under the operations of rotation, reflection or compositions of these, where each equivalence class determines a distinct subshape instance.

When a shape has been generated, the user can, by typing a P command, obtain a line printer description or a hard copy line drawing of the shape.

In the GEN phase, the shape rules are rearranged in their reduced form (Krishnamurti, 1981). That is, each shape rule is given by the 4-tuple: $< \alpha , \alpha - \beta , \beta - \alpha , \alpha . \beta >$. The components of the 4-tuple serve the following purpose. The 1-st component is used in the determination of the subshape instances. The 2-nd and 3-rd components are employed in shape rule application algorithm. The 4-th component is required to recover the shape rule to the standard form.

A display of a shape rule in both its standard and reduced forms can be obtained by typing an S command.

When the user has generated the shapes he requires, he can return to the SGI phase by typing a Q command.

It should be noted that in the GEN phase, the active shape is always the current shape.

The command sets

Each valid command type is represented by a single letter from the set {A,..,Z}. Some command types consists of several commands, each of which is specified by an additional command parameter. For example, the T command represents an euclidean transformation, and the command TS indicates that the transform is a scaling. Each phase has its own set of commands, and some commands are common to both phases. These shared commands have identical formats. The command sets for the SGI and GEN phases are summarized in Tables 1 and 2 respectively. The parameters described within square brackets ([ ]) are optional. Note that the commands listed in Table 2 are just those commands which are either exclusive to the GEN phase, or result in a different action to that in the SGI phase. For a complete description of the SGI and GEN phase commands, the reader is referrred to the SGI user manual (Krishnamurti, 1982).

The parameters, other than those that specify a particular command, can be supplied either with the command string or when the program prompts the user to supply them. Moreover, where possible, the program is forgiving in that incomplete command strings or parameter information can be specified. In such situations, the program will, by prompting the user, attempt to complete the command. For instance, if the program expects the coordinates of a point, and the user has supplied only the x-coordinate, the program will prompt for the y-coordinate. In order to ensure that unexpected things do not occur, the user has the facility to terminate any command prematurely, simply by typing a blank input to a program prompt. The program will not carry out a command until it has received all the relevant information.

Some of the commands are repetitive. That is, the program will, after executing each instance of the command, prompt the user for the parameters for a fresh instance of the command. Such commands are terminated by typing in a blank input to a program prompt.

Command	Parameters	Description
A	display shape	Activate and display the indicated shape. The subsequent commands refer to this active shape
B		reset the deBugging switch
C	line/point no.	select the indicated line (or point) as the Current line (or point)
D	[line/point no.]	Delete the current [or indicated] line/point and renumber if necessary
E	[rule no.]	Enter a copy of the active shape as a side of the indicated shape rule. If no rule number is supplied, increment the <u>highest rule entered</u> by 1 to give the new rule number
ER	[rule no.]	Enter a copy of the left and right rule shapes as the indicated shape rule
F	[side] [rule no.]	Fetch a copy of the indicated side of the indicated shape rule onto the active shape. If no side is supplied, the entire shape rule is fetched onto the rule shapes. If no rule number is supplied, the <u>last rule entered</u> is fetched
G	production rules	enter the Generation phase. The indicated production rules are the only shape rules that can be referenced in the generation phase

Table 1: Summary of the commands available for constructing shape grammars

Command	Parameters	Description
H or H*	command(s)	Help - provide a brief (and if * is supplied, a complete) description of the indicated SGI phase command(s)
H.	status key	Help - provide the requested SGI phase status information
I		re-Initialize the program to the original state
J (line mode)	position to line no. position	Join - move the current line along the X- and/or Y-directions so that the indicated position on the current line coincides with with the indicated position on the indicated line, and renumber if necessary. The positions are specified as shown in Figure 5
J (point mode)	point no.	Join - delete the current point and union its labels with the labels of the indicated point and renumber if necessary
K or KD		draw a clean version of the active shape. If the D parameter is supplied, reset user-accessible variables to their default values; that is, the program is placed in line mode and no current line is defined, and the display of the line/point numbers switched off
KC	graphics element and colour map	change the colour associated with the indicated graphics element to the indicated colour

Table 1 (continued)

Command	Parameters	Description
L (line mode)	position add/remove symbol label(s)	Label - add to or remove from the indicated position on the current line, the indicated label(s)
L (point mode)	add/remove symbol label(s)	Label - add to or remove from the current point, the indicated label(s)
M	on/off symbol	Mark (display) or unmark the numbers assigned to the lines/points
MX or MY	on/off symbol	Mark or unmark the coordinate axes
N (line mode)	from coordinates to coordinates	create a New line from the indicated point to the indicated point (This command is repetitive)
N (point mode)	coordinates label(s)	creates a New labeled point at the indicated coordinates with the indicated label set (This command is repetitive)
O	[pair of rule nos.]	Order rules - if no parameter is supplied, the nonempty shape rules are renumbered from 1 consecutively. This command may alter the number of the highest rule and the last rule entered. Otherwise, interchange the numbers associated with the indicated shape rules
P	device shape(s)	Print, Plot or Photograph the indicated shape(s) The shape(s) can be either i)the active shape, or ii) (a side of) a shape rule

Table 1 (continued)

Command	Parameters	Description
Q		Quit - terminate program and return to system monitor
R	file name	Read the indicated shape from disk and add to active shape
R.	file name	Read in a new shape grammar from disk
S	[side] rule no.	Show (display) the [indicated side of the] indicated shape rule
T	x-coordinate and y-coordinate transformations	Transform - apply the user supplied transformation to the current line/point
TR	angle [M] (line mode)	Rotate the current line/point through the indicated degrees counterclockwise. If the parameter M is supplied, the point of rotation is the mid-point of the current line
TS	scale factor	Scale the current line/point by the indicated scale factor
TX	distance	move the current line/point along the X-direction by the indicated distance
TY	distance	move the current line/point along the Y-direction by the indicated distance
TI	axis [M] (line mode)	Invert (reflect) the current line/point about the indicated axis, which if the parameter M is supplied, passes through the mid-point of the current line

Table 1 (continued)

Command	Parameters	Description
PH		Print help - provides a listing of the user manual
U	comment(s)	writes User's comment(s) onto comments file (This command is repetitive)
V	mode	View - switches the program into the indicated mode, namely, lines or points. The subsequent commands affect the elements in the current mode
W	file name	Write active shape onto disk
W.	file name	Write current shape grammar onto disk
X	minimum and maximum coordinates	reset the X-coordinates of the user window to the indicated values
Y	minimum and maximum coordinates	reset the Y-coordinates of the user window to the indicated values
Z	set	Zap the indicated set, namely, lines or labeled points from active shape

Table 1 (continued)

Command	Parameters	Description
A	selection rule	Apply the indicated shape rule to the first subshape instance found, if any, to yield a new current shape
B		Backtrack to the previous current shape in the generation process
D		reset the Debugging switch
H or H*	command(s)	as in Table 1 but refers to the GEN phase commands
H.	status key	Help - provide the requested GEN phase status information
P	device shape(s)	as in Table 1 except that shape rules can plotted or photographed in both their standard and reduced forms
Q		Quit - terminate and return to the grammar definition phase
R	selection rule	apply the indicated shape Rule to a specified subshape instance (if any). When a subshape instance has been determined, the program will enquire if the user wishes to apply the rule. If not, the program continues to search for other subshape instances
S	[side] rule no.	Show (display) rule - as in Table 1 except that the user has the choice of seeing the indicated shape rule in both the standard and reduced forms

Table 2 : Summary of some commands available for generating shapes

In the event that a command or parameter string cannot be properly decoded as result of some error, the program reports an error message. All program messages are designed to be as instructive and helpful as possible. Of course, the ultimate judge of whether the program is successful in this objective is the user himself.

The commands can be entered free formatted. No separators are required between pararameters except between numeric tokens in which case the program expects at least one blank space.

There is no case distinction for the character set. That is, the program does not distinguish between lower and upper case characters.

Any input string to the program can be terminated by a semicolon (;). Any character string that follows a semicolon is treated as a comment.

Each command can be classed as belonging to one of the following command categories:

1. Switch
2. Graphics
3. Rule
4. Input/Output
5. Termination
6. Aid

1. Switch commands

A switch command enables the user to set or reset certain program variables. Consider first the SGI phase commands.

The A (activate) command sets the specified display shape to be the active shape which the subsequent graphics commands affect. The active shape is displayed on the screen.

The G (generate) command switches the program to the shape generation phase. This command has an integral parameter, say N, which indicates that shape rules 1 through N are to be used as the production rules in generating the shapes. This number N can be less than the number of shape rules defined in the SGI phase.

The M (mark) commands set or reset the display of either the coordinate axes, or the line or labeled point numbers.

The V (view) command sets the program mode which defines whether the graphics commands affect the lines or the labeled points in the active shape.

The X (X-axis) and Y (Y-axis) commands respectively define the new x- and y-coordinate axes for the user window.

The I (initialize) command sets the program to the default initial state. All information resident in the program is lost except the current coordinates for the user window and colour maps for the graphics elements.

In the GEN phase the commands available are M, V, X and Y.

2. Graphics commands

These commands perform graphics type actions on the lines and points in the active shape. The commands affect either the lines or the labeled points depending upon the mode the program is in.

The C (choose current) command selects the indicated line or labeled point as the current line or point on which graphics type operations can be performed.

The D (delete) command removes from the active shape, the indicated line or labeled point. This line or labeled point can be the current line or point.

In line mode the J (join) command fastens the current line to another line in such a manner that two specified positions, one on the current line and another on the second line, share the same coordinates. That is, the current line is moved along the X- and Y-directions until the specified positions coincide. Any position on (or outside) a line is specified, by typing an end point of the line and a fraction (which can be negative). The fraction is the ratio of the distance of the position from the indicated endpoint and the length of the line. For example the position specifier 1/5 L denotes a position 1/5-th of the line length away from the left hand end point line. Figure 5 gives examples of positions on a line and their specifiers. In point mode, the J commmand unions the current labeled point with the specified labeled point. In other words, the current point is deleted and its labels are unioned with those of the specified labeled point.

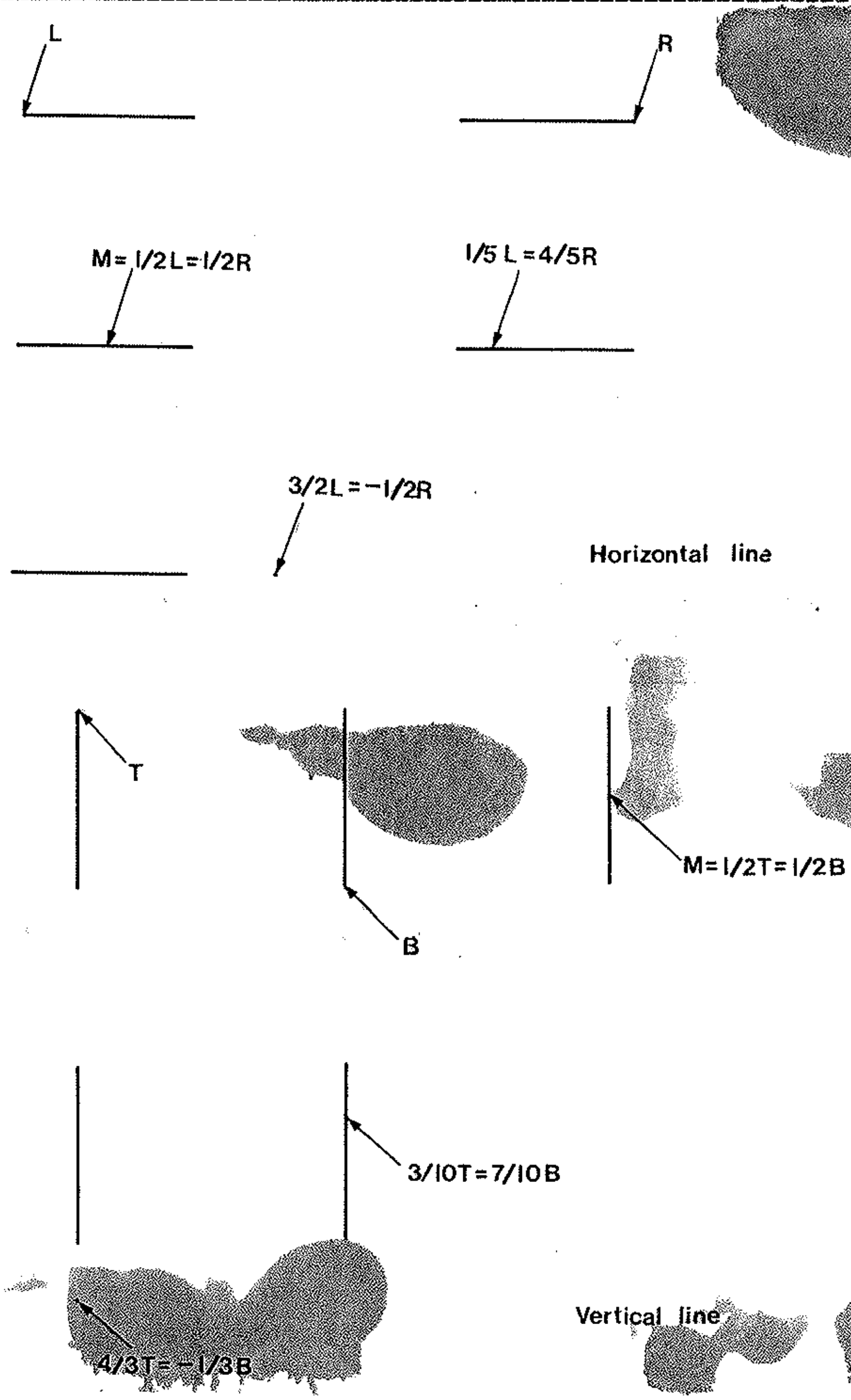


Figure 5: Specifying positions on a line

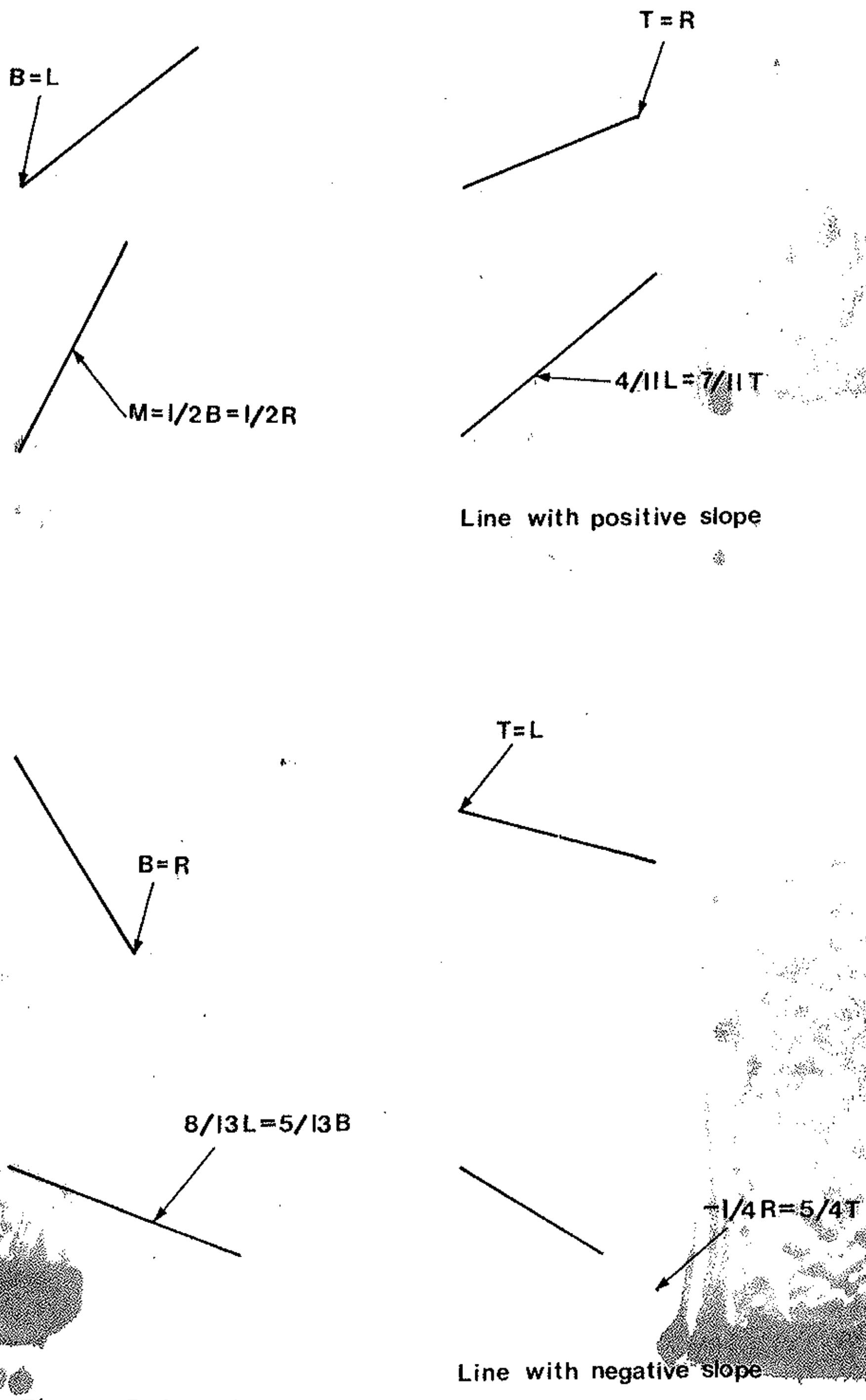


Figure 5 (continued)

The L ([un]label) command adds labels to or removes labels from, the current point if the program is in point mode, and the indicated position on the current line if the program is in line mode.

The N (new) command creates in the active shape, new lines or new labeled points. This command is repetitive in that the the program prompts the user for more lines or labeled points. A blank input to one such prompt terminates the command. When a line is entered, the program first determines whether it is a maximal line. If it is not, it is combined with maximal lines in the active shape to form a new maximal line. When a labeled point is entered, the program first determines if there is a labeled point already present at the specified coordinates. If so, the command is equivalent to a label (L) command.

The Z (zap) command removes either the lines or the labeled points from the active shape.

The T (transform) command does an euclidean transformation on the current line or labeled point. The specific transformation is described by the single character that follows the "T". For example, TR is a rotation; TS performs a scaling; TX and TY are translations along the X- and Y-axis respectively; and TI represents axial inversion (or reflection) - TIX performs the inversion about an X-axis, and TIY performs the inversion about a Y-axis. The current line can be inverted about an axis which passes through, and can be rotated about, either its mid-point or the origin of the coordinate system. The TS command can be used to either shrink or enlarge the current line. The user can also specify his own transformation, by not supplying the the specific transformation type parameter. In this case, the program prompts the user to provide the coefficients for both the x- and y-coordinate transformations.

The K (teKtronix) command displays a clean version of the active shape on the screen. The KD command also provides a clean version of the active shape and in addition, resets some user accessible variables to their default values. For example the program is placed in line mode with no current line defined and the line/point numbering is switched off. The KC command changes the colour maps and hence, the colour for the various graphics elements such as line, text and background colours.

In the GEN phase the only command available is K.

3. Rule commands

Rule commands invoke or manipulate the shape rules. There are four rule commands in both phases, one of which is shared, namely, the S (show rule) command which displays a (side of a) shape rule on the screen. In the GEN phase, both the standard and reduced forms of the shape rule can be seen.

In the SGI phase, the rule commands are used to define and organize the shape rules. The E (enter rule) stores a copy of either the active shape as a side of a shape rule or the left and right side display shapes as a shape rule. The F (fetch rule) places a copy of either an indicated side of a shape rule onto the active shape or a shape rule onto the left and right side display shapes. Only shape rules that have been defined earlier can be fetched. The F command is useful when the left side of a shape rule is the right side of another shape rule. Such shape rules often occur in practice. The E command can be used to erase or replace existing shape rules. The shape rules can be defined in any arbitrary order. For both the E and F commands, the active shape must be one of the two rule shapes.

The O (order shape rules) command organizes and renumbers the shape rules consecutively from 1 onwards, in the process removing all shape rules that have been erased. Any shape rule that has been erased by an E command is still associated with a rule number. The O command can also be used to interchange two shape rules. This facility is useful when several versions of a shape grammar, each differing in a few rules, are required to be tested.

In the GEN phase, the rule commands correspond to shape rule application commands. The R and A (apply rule) commands apply a shape rule to generate a new shape from the current shape. The A command applies the shape rule to the first subshape instance found. The R command gives the user the choice of which subshape instance is to be replaced.

The B (backtrack) command applies, in reverse, the most recently applied shape rule to the current shape, to obtain the immediately previous shape in the generation process.

4. Input/output commands

There are three input/output commands: the R (read file), W (write file), and P (print, plot or photograph) commands.

The R command reads a shape or a shape grammar from a file on disk. Prior to reading in a shape grammar, the program reinitializes itself. Consequently, any information resident in the program is lost. When reading in a shape, the user window is readjusted to accommodate both the shape and active shape to which the shape is unioned. The read command is not available in the GEN phase.

The W command writes the shape grammar or the active shape onto a file on disk. In the GEN phase, only the current shape can be written to disk.

It should be remarked that if there is more than one file with same name resident on disk, the program will read/write the highest numbered version of the file. There is, as yet, no provision for specifying the version number for a file.

The P command provides a hard copy output of the active shape or a shape rule. In the GEN phase, the hard copy output for both the standard and reduced forms of the shape rule can be obtained. The PL command gives a line printer description of the specified shape(s). THe PG command provides a graphics plotter line drawing of the specified shape(s). The PC command takes a photograph of the specified shape(s) with the camera.

5. Termination command

There is only one termination command in each phase, the Q (quit) command. If this command is typed in the GEN phase, the program returns to the SGI phase. If the command is entered in the SGI phase, the program returns to the system monitor.

6. Aid commands

These are general purpose commands to aid the user during the course of an interactive session and to provide debugging facility for program development and improvement. Some of the commands are not available to the general SGI user.

The H (help) command provides 1) status information on the program variables; and 2) information on the various commands. The status information provided are:

1) In the SGI phase:
   a) the coordinates of the user window;
   b) the active shape;
   c) the current line or labeled point and its coordinates;
   d) the current program mode;
   e) the shape rules entered;

2) In the GEN phase:
   a) the coordinates of the user window;
   b) the current sequence of shape rule application;
   c) the number of production rules in the grammar;
   d) the rule type for each shape rule.

The PH (print help) command generates a printer listing of the SGI user manual (Krishnamurti, 1982). The user is advised to read the manual prior to using the program.

The U (users comments) command allows users to enter any comments, suggestions for improvement, or requests for specific facililties. These comments are placed in a file which is in the directory of the programmer in charge of the program. This provides a machinery wherby users can communicate their needs.

The following commands are not available to general users except at the discretion of the programmer in charge of SGI.

The B (in the SGI phase) and D (in the GEN phase) commands generate debug trace information. The PD (print dump) command generates a dump of the list structures. These commands have been found to be extremely useful during program development. Since the program is still considered to be in its infancy, these commands have not been removed. It is recommended that they are used judiciously as they produce reams of printout.

Example 1: Inscribing squares within squares

Consider the simple shape grammar shown in Figure 1. The grammar consists of two shape rules. The language defined by this grammar contains shapes consisting of n(>Ø) squares, one inscribed in another. The SGI program to construct the grammar is developed below. Each command has a comment which describes the action of the command.

First, set up a user window, say, between (-1Ø,-1Ø) and (1Ø,1Ø). The commands to do this are:

```
X -1Ø 1Ø ; sets the X-axis for user window
Y -1Ø 1Ø ; sets the Y-axis for user window
```

Second, define the initial shape. The following sequence of commands constructs a square with corners at (-5,5), (5,5), (5,-5) and (-5,-5):

```
AI ; activate the initial shape
VL ; set program to line mode - that is, view lines
N ; inform program that you are ready to input lines
```

```
line? -5 5 5 5
line? 5 5 5 -5
line? 5 -5 -5 -5
line? -5 -5 -5 5
line?<CR> ; <CR> denotes a carriage return
M+ ; display the line numbers
C2 ; choose the line numbered 2 as the current line
LMA ; label the mid-point of the current line with label A
PG ; plot the initial shape
... <--- program -user interaction with the program requesting
... <--- information about plotter parameters
do you wish to title plot (y or n)?Y ; user replies with a yes
ok, enter title (max 60 characters)?(a) Initial shape
... ;  further program - user interaction
... plotter disconnected  <--- program indicates that plotting
                               is completed
```

The plot of the initial shape is shown in Figure 6(a). Notice that the label symbol ● in Figure 1 has been replaced by the letter A.

Third, since this shape appears as the left side of shape rules 1 and 2, it is convenient to save it.

```
W SQUARE ; save the initial shape on a disk file named "SQUARE"
```

Fourth, construct the shape rules. Consider shape rule 2 first.

```
AL ; activate the left side display shape
R SQUARE ; read in the square constructed above
AR ; activate the right side display shape
R SQUARE ; again, read in the square constructed above
ZP ; remove all  (in this case, the only) labeled points in the
     active shape
ER2 ; define shape rule 2
```

At this stage, it should be remarked that the left and right side display shapes remain unaltered and that the right side shape is currently active. If the user is in any doubt, the commands:

```
H.A ; request the currently active shape
H.V ; request the current mode
```

provide the necessary status information.

Consider shape rule 1. The command sequence:

```
N ; inform program of new lines
line?-5 Ø Ø 5
line? Ø 5 5 Ø
line? 5 Ø Ø -5
line? Ø -5 -5 Ø
line?<CR>
```

constructs a square inscribed inside the original square.

```
M+ ; switch on the line numbers
C2 ; choose line numbered 2 as the current line
LMA ; label the mid-point of the current line with label A
ER1 ; define shape rule 1
```

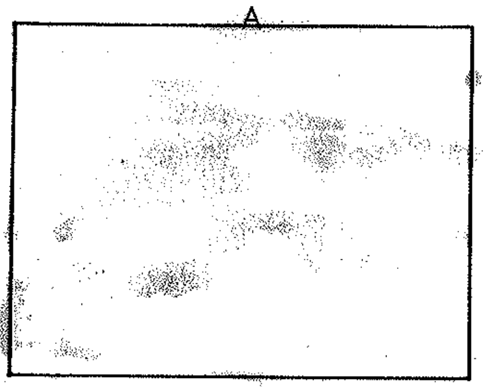
The commands:

```
PG1 ; plot shape rule 1
PG2 ; plot shape rule 2
```

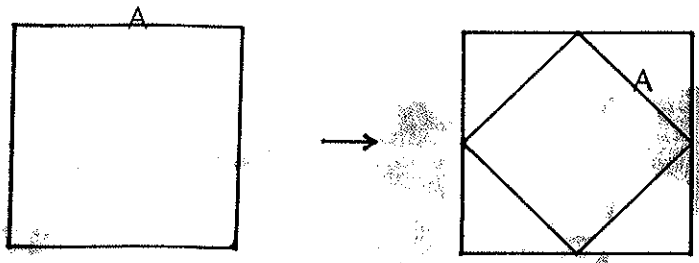
provide line drawings on the plotter for shape rules 1 and 2 (see Figures 6(b) and 6(c) respectively).

The shape grammar has been defined. The following command saves the shape grammar on disk:

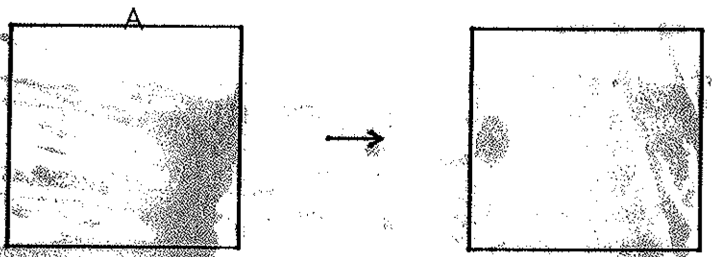
```
W. GRAMMAR ; save the grammar on a disk file named "GRAMMAR"
```



(a) Initial shape



(b) Shape rule 1



(c) Shape rule 2

Figure 6

Now, let us consider generating some of the shapes in the language defined by this grammar. For this, we need to enter the generation phase:

G2 ; inform the program that shape rules numbered 1 and 2 are the production rules for the grammar

At each stage in the generation, shape rule 1 can be applied to two distinct subshape instances which are axial reflections of one another (see Figure 7). A sequence of:

A1 ; apply rule 1 to first subshape instance found

will generate the intermediate shapes shown in Figure 8. And a sequence of commands consisting of some combination of:

A1 ; apply rule 1 to first subshape instance found

and

R1 ; apply rule 1 to user-specified subshape instance : in this case the program will display each subshape instance and will enquire from the user if [s]he wishes:

<u>apply</u> <u>to</u> <u>subshape</u> <u>instance</u> (<u>y</u>,<u>n</u>,<u>r</u> <u>or</u> <u>q</u>)?

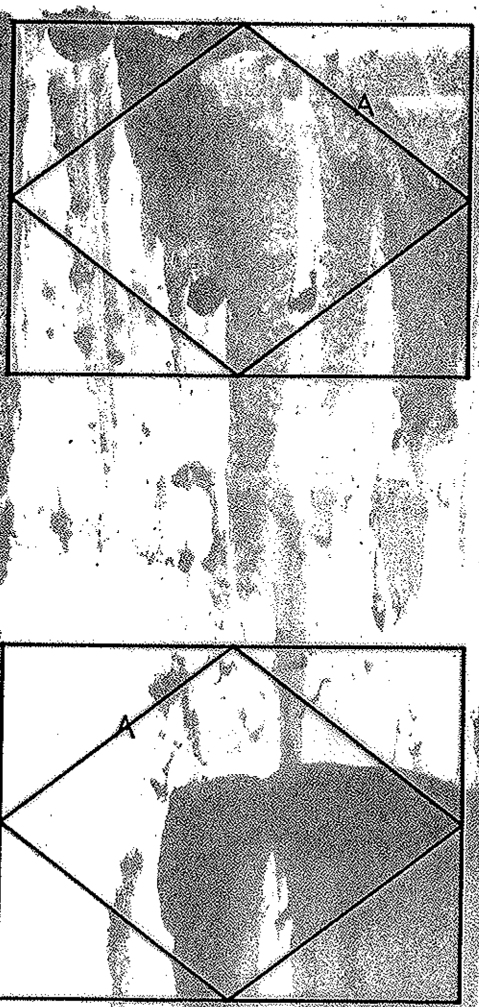
(see the section on interactive generation of shapes for a description of the various options)

will generate the intermediate shapes shown in Figure 9.

Applying the erasing rule, by typing the command:

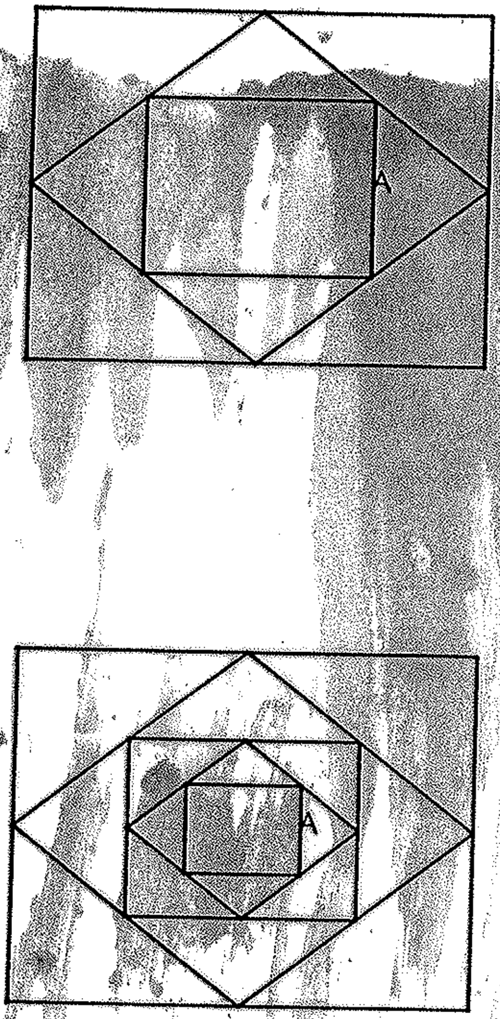
A2 ; apply shape rule 2

to each of the intermediate shapes in Figure 8 yield the shapes in Figure 1Ø which correspond to shapes in the langauge defined by the grammar. Notice that the bounding square for all such shapes is always the same.



The two distinct subshapes

Figure 7



Shapes generated using the A command

Figure 8

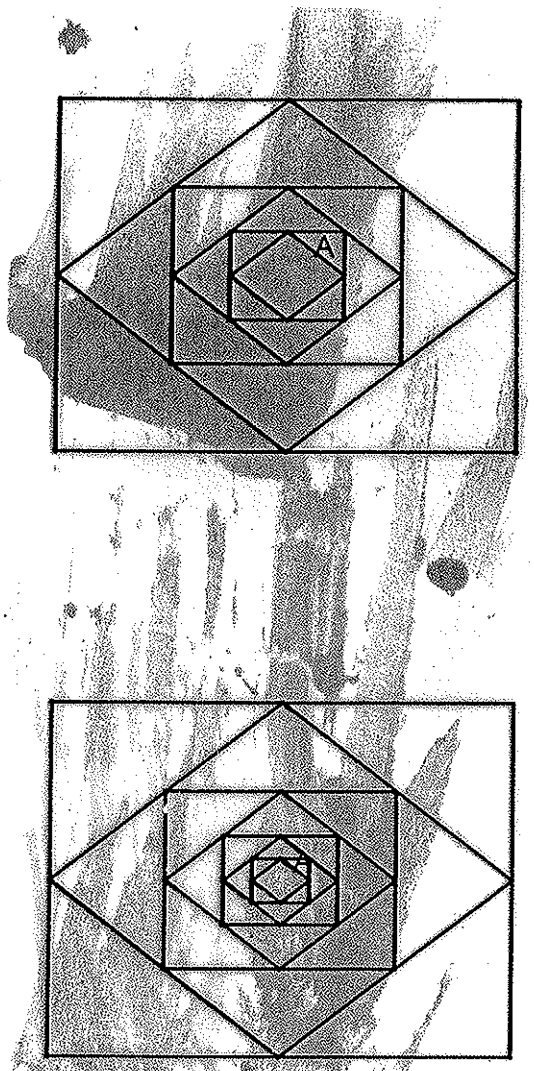
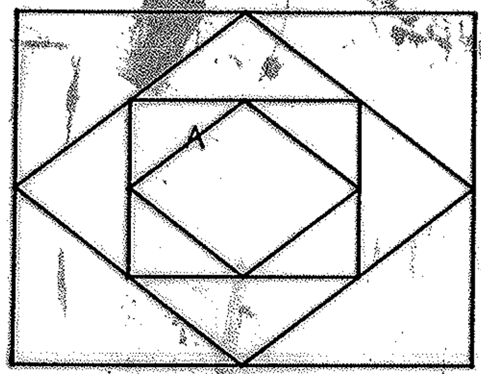
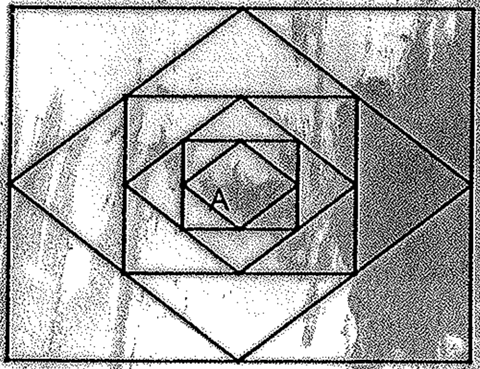


Figure 8 (continued)

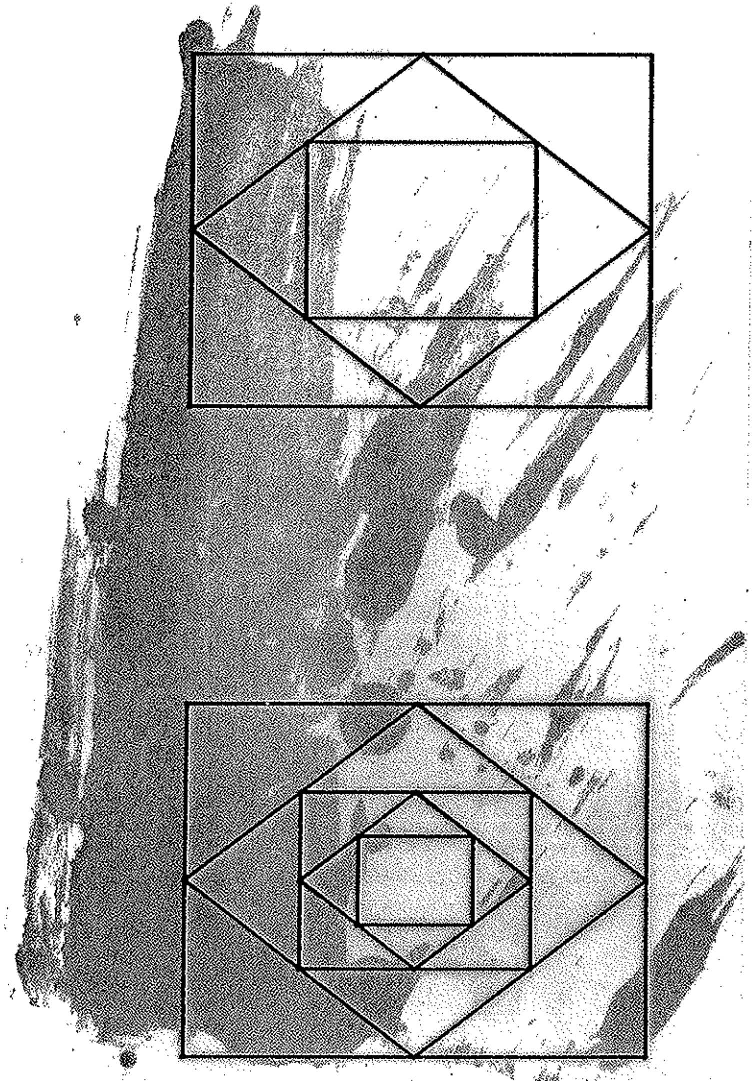
Figure 8 (continued)





Shapes generated using the R and A commands

Figure 9



Terminal shapes in the language

Figure 1Ø

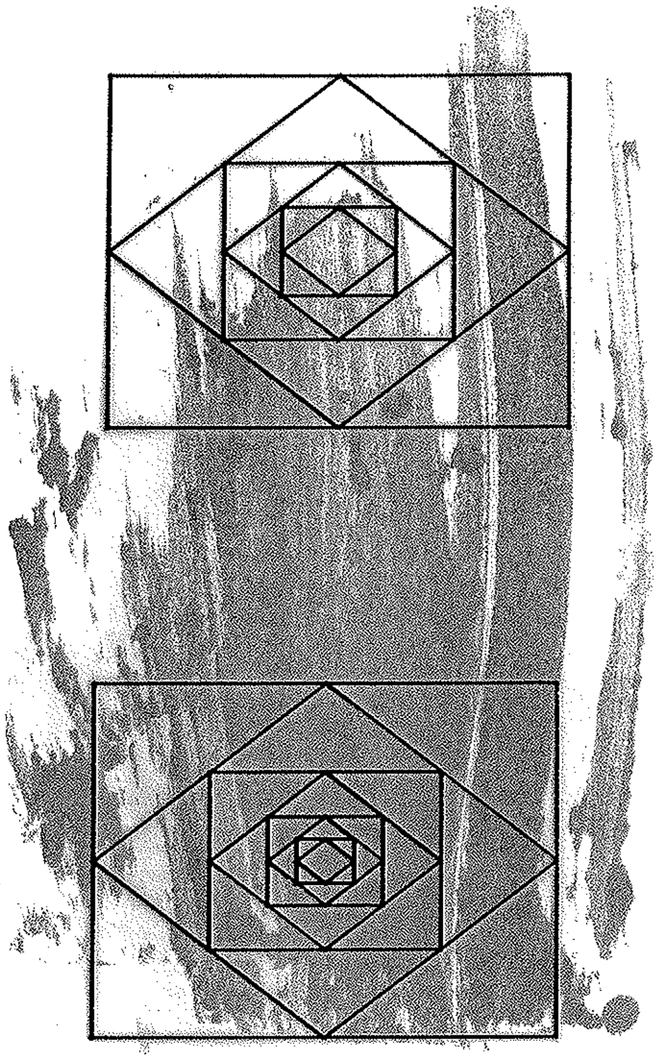


Figure 1Ø (continued)

Figure 1Ø (continued)

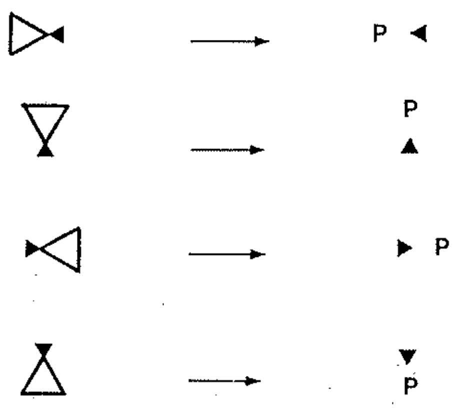
Example 2: Rectangular shapes

We now consider a class of shapes that has attracted the attention of researchers over the past decade (see Earl, 1980 for a reasonable bibliography). The shapes correspond to floor plans, and in particular, to the arrangement of rooms within a rectangular framework and are referred to as rectangular shapes (Earl, 1980). The grammar in Figure 2 generates all possible rectangular shapes within a fixed framework which is given by the intial shape. Strictly, the grammar in Figure 2 is not a shape grammar in the technical sense in that the initial shape cannot be specified with arbitrary dimensions but instead, must be generated by applying a set of rules to some fixed initial shape. This problem is considered in the next example.

It is worthwhile to reflect briefly on the rules. Shape rules 1 through 6 remove lines from and change marker positions in the original initial shape. Shape rules 7 through 12 erase the marker shapes. Earl (1980) has shown that by selecting a subset of the line erasing rules (1 through 6) and a subset of the marker erasing rules (7 through 12), different classes of rectangular shapes can be generated. For instance, dissections of rectangles into rectangles can be generated using rules 1,2, 7,9,10 and 11.

In order to define the shape grammar using the program, the first step is replace the labeled points and marker shapes by points associated with a letter label. Clearly, the label • can be replaced by the label, say Q. The marker appears in the grammar in four distinct orientations: ▷ , ▽ , ◁ and △ .

Each orientation must be uniquely represented. The obvious solution is to assign a distinct label to denote each distinct marker orientation. But, this would involve replacing each shape rule by a set of shape rules each of which corresponds to the different marker orientation. Alternatively, by using a single label, say P, to represent the shape ▷ , and by positioning the label with respect to the lines in the shapes, the original set of rules can be preserved. For convenience, let the tip of the marker by denoted by a ▶. Then, the four orientations of the marker can be represented by the four distinct positions of the label P with respect to the ▶ , as shown below:



It should be observed that the line erasing shape rules are applied to the line segments between two consecutive grid lines in the initial shape. Consequently, to ensure that the program does not expend a lot of fruitless computation searching for all possible subshape instances, it is advisable to label the mid points of the line segments with the label, say S. This would then add an extra erasing rule which removes all labeled points with the label S. The shape rules take the form shown in Figure 11.

Consider the initial shape shown in Figure 12. Suppose we wish to generate the divide the initial shape into rectangles. As stated earlier, we require just seven of the thirteen rules: 1,2,7,9,1Ø,11 and 13. In order that the shape rules appear to the program as rules 1 through 7, they have to be renumbered. The command sequence:

```
O 3 7  ; interchange rules 3 and 7
O 4 9  ; interchange rules 4 and 9
O 5 1Ø ; interchange rules 5 and 1Ø
O 6 11 ; interchange rule 6 and 11
O 7 13 ; interchange rule 7 and 13  -  notice  rule 7 currently
         refers to the original rule number 3
```

accomplishes the renumbering of the rules. We can now enter the generation phase by typing in:

```
G7 ;  enter generation phase with shapes rules numbered from  1
      to 7 inclusive as the production rules for this grammar
```

A labeled rectangular shape generated by applying the line erasing rules 1 and 2 is shown in Figure 13. The corresponding rectangular shape generated by applying the label erasing rules 7,9,1Ø,11 and 13 is shown in Figure 14.

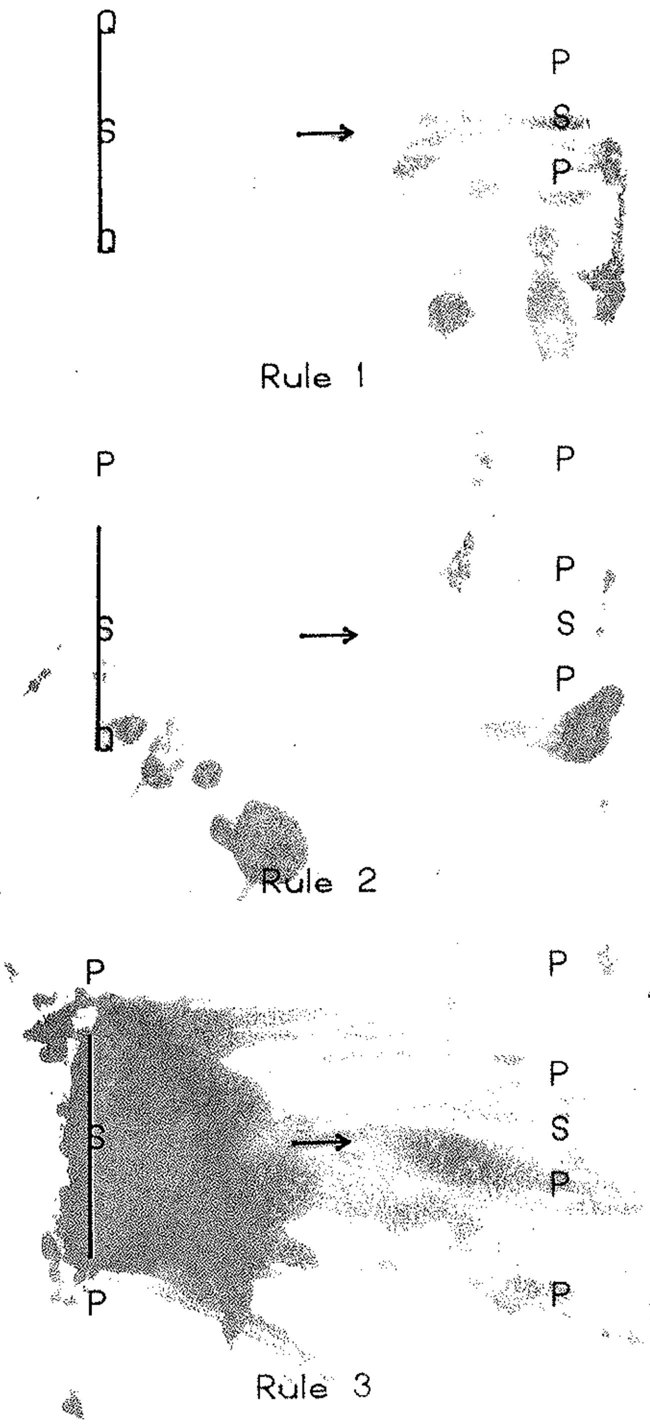
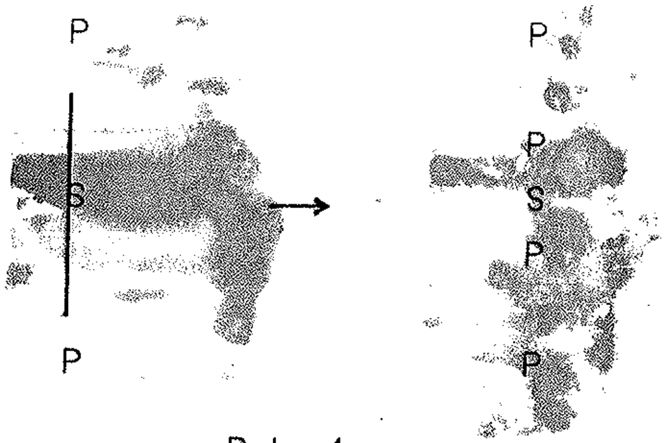
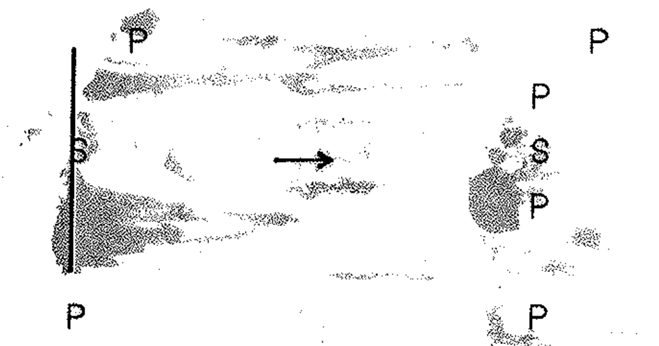


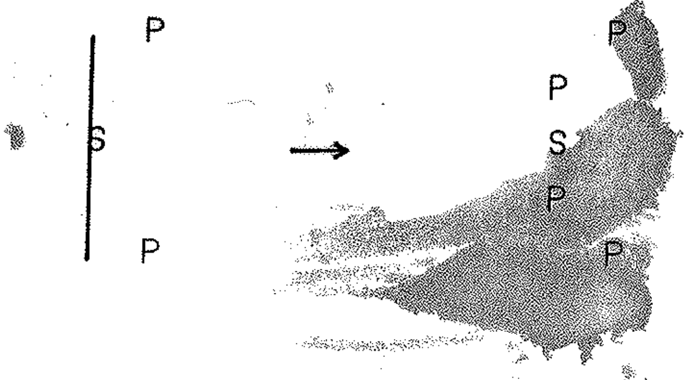
Figure 11



Rule 4

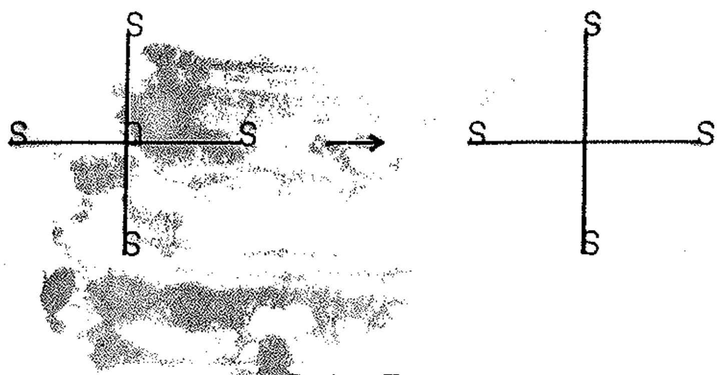


Rule 5

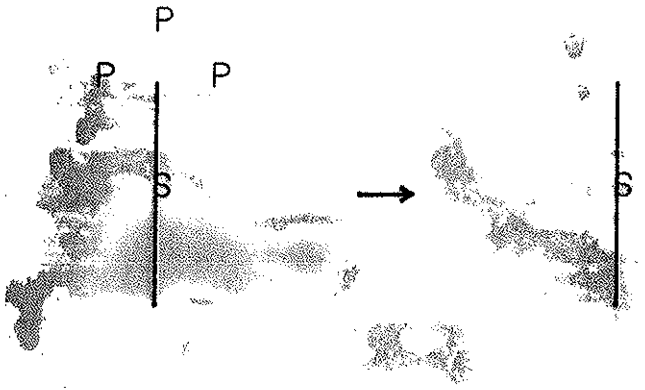


Rule 6

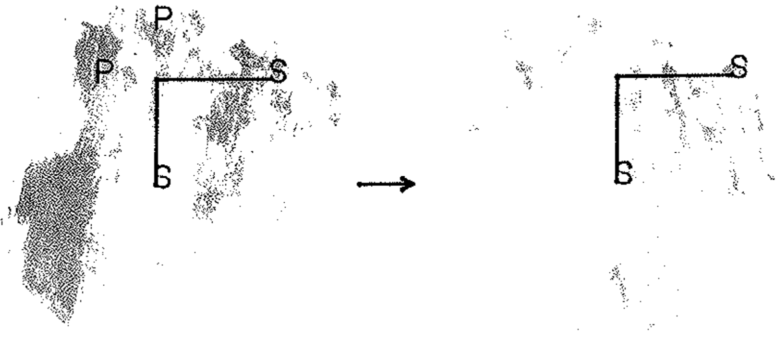
Figure 11 (continued)



Rule 7



Rule 8



Rule 9

Figure 11 (continued)

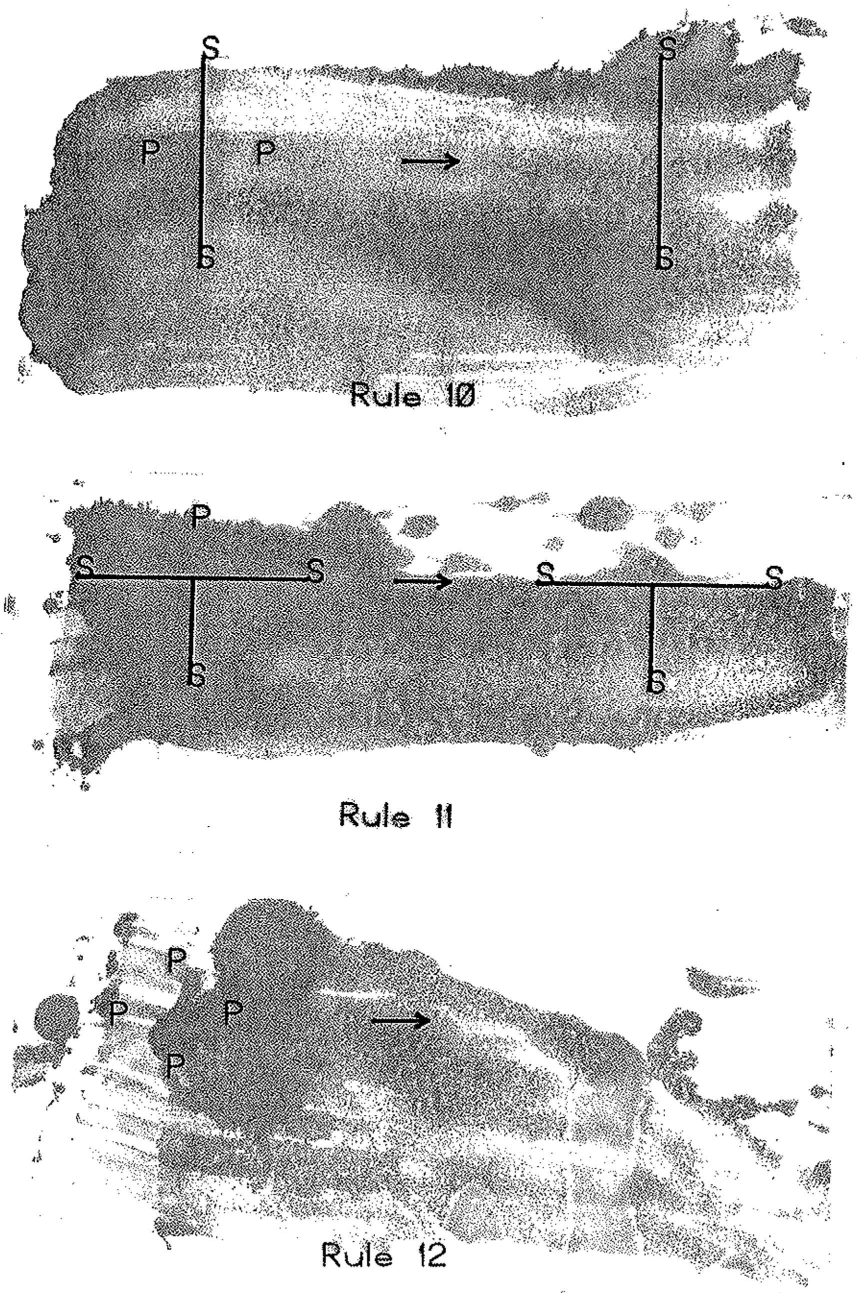


Figure 11 (continued)

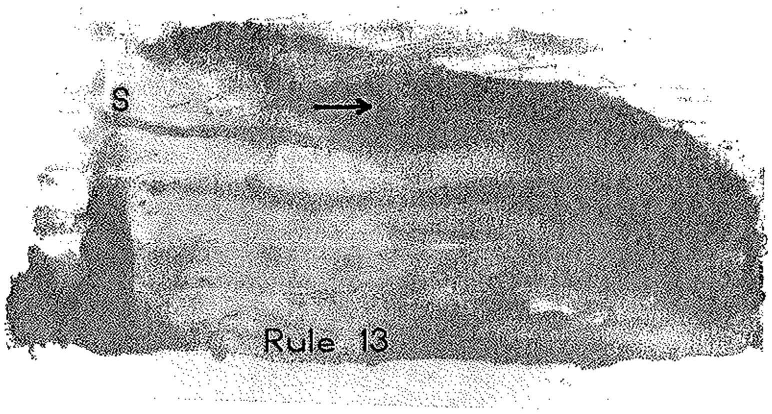


Figure 11 (continued)

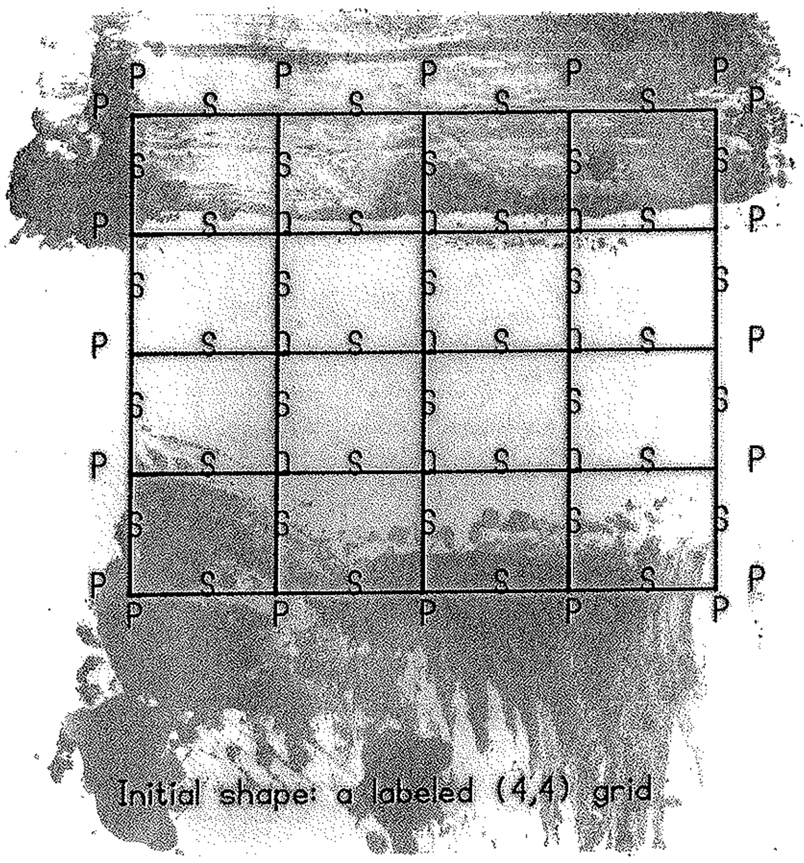


Figure 12

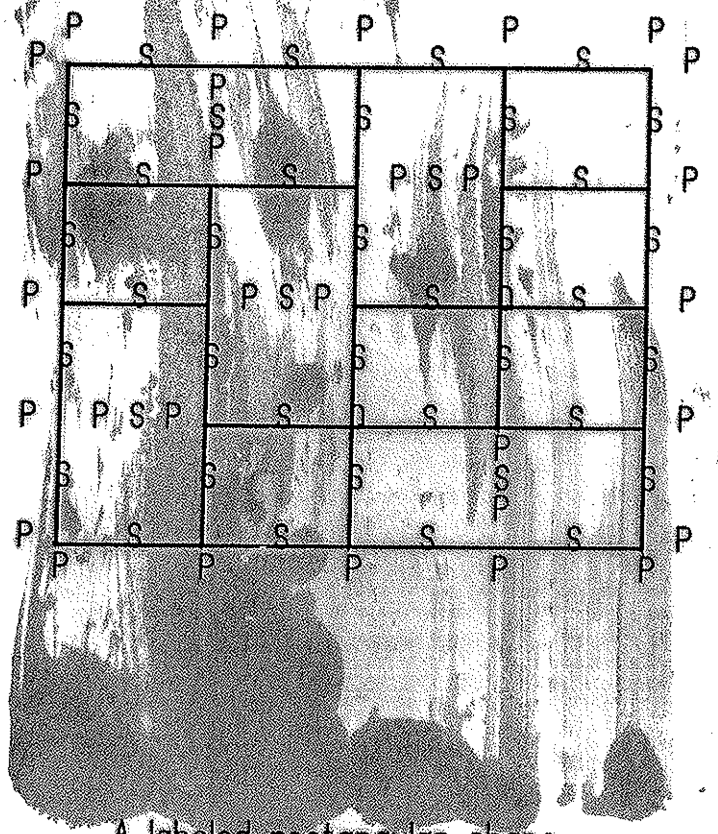

A labeled rectangular shape

Figure 13

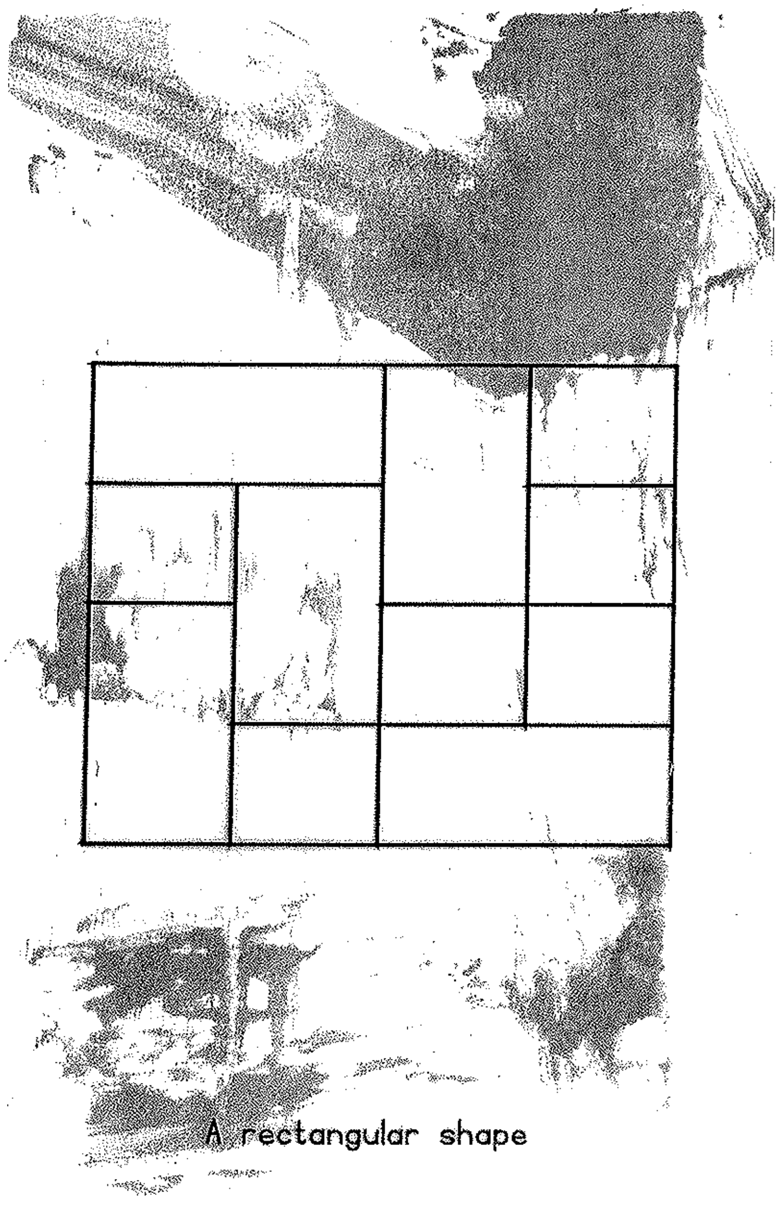
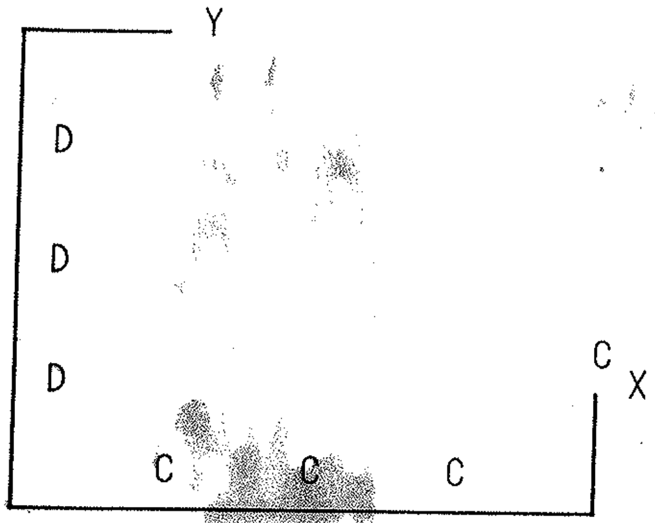


Figure 14

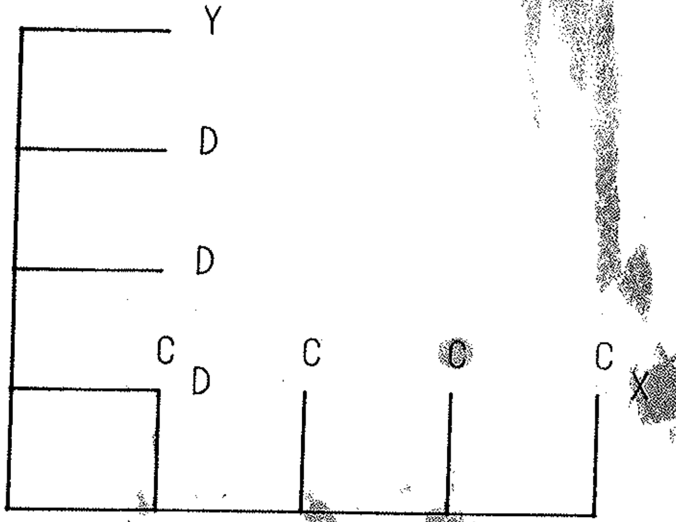
Example 3: Grid grammar

We now present a shape grammar for generating rectangular grids. A grid is a shape which consists of maximal lines all parallel to the cartesian coordinate axes, four of which share end points (the boundary rectangle) and the rest have their end points coincident with the boundary rectangle. An (m,n) grid has m+1 maximal lines parallel to the y-axis and n+1 maximal lines parallel to x-axis. Parametric shape schemas for the generation of various classes of (m,n) grids can be found in (Stiny and Mitchell, 1978,1980; Weissman Knight, 1981).

The shape grammar is shown in Figure 3 and consists of ten shape rules. Shape rules 1 and 2 define the x-axis for the grid and likewise, shape rules 3 and 4 define the y-axis. Rule 5 extends the internal maximal lines parallel to the y-axis by a unit distance. In same fashion, shape rules 6 and 7 extend by a unit distance the internal maximal lines parallel to the x-axis. Rule 8 is similar to rule 7 except that it affects just the top most maximal line of the boundary rectangle. Rule 9 affects the right most maximal line of the boundary rectangle and extends it by a unit distance when a row of grid cells have been constructed. Rule 10 is the termination rule which only applies when the grid has been generated. Notice that in order to construct an (m,n) grid, shape rule 1 has to be applied m-1 times and shape rule 3 has to be applied n-1 times. Figure 15 illustrates some stages in the generation of a (4,4) grid.

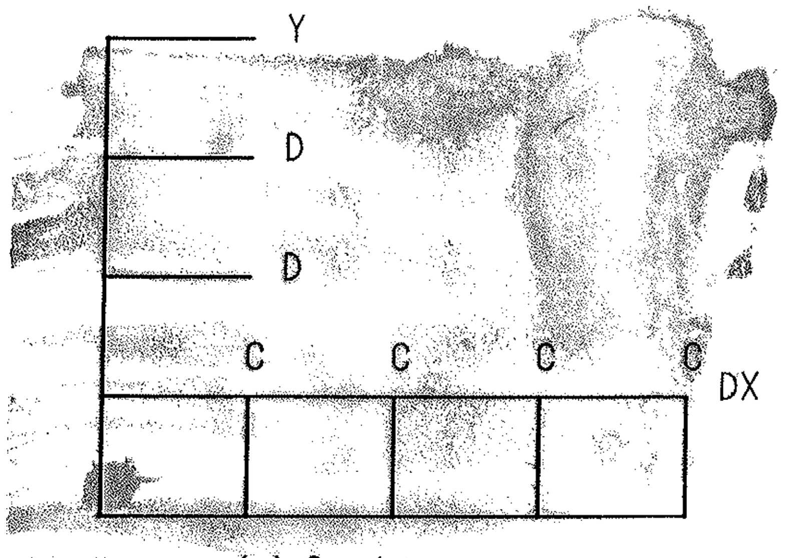


(a) Define the size of the grid

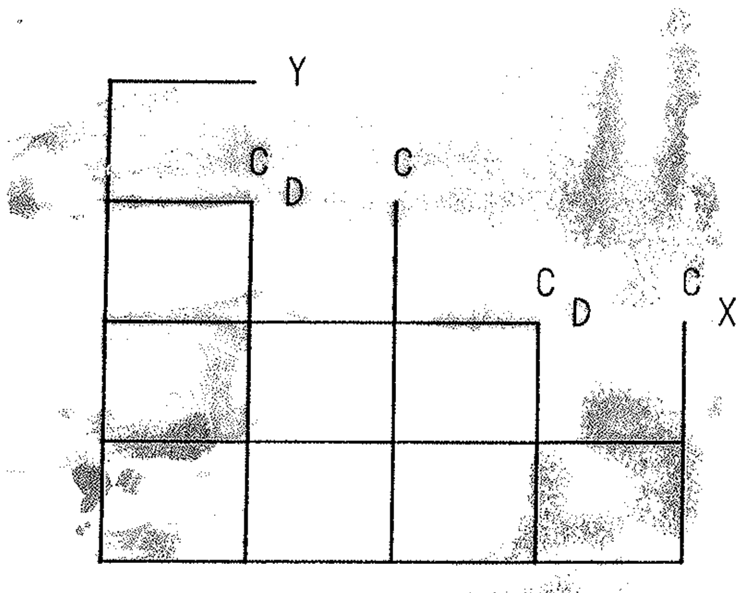


(b) Start the internal maximal lines

Figure 15

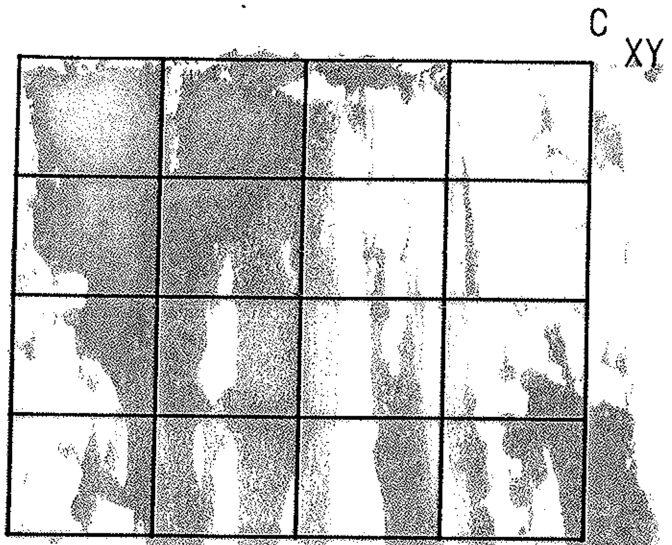


(c) Complete a row

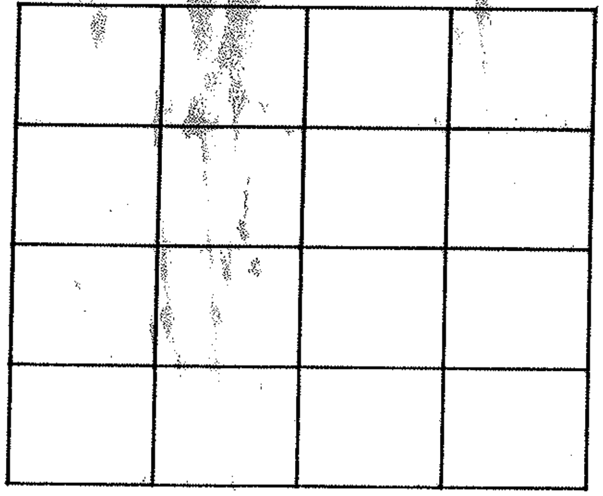


(d) An intermediate shape

Figure 15 (continued)



(e) Complete the grid



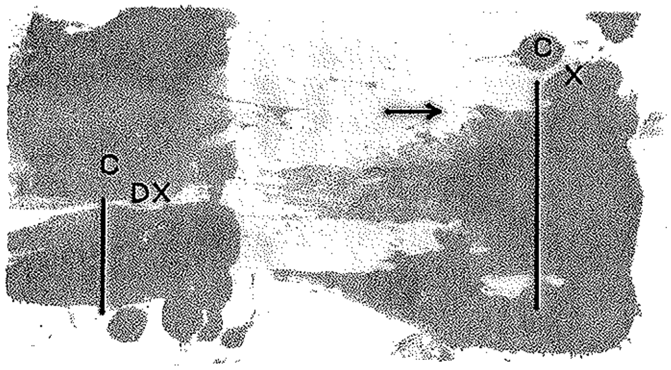
(f) Terminate: the (4,4) grid

Figure 15 (continued)

Notice that the shape rules can be modified in order to construct, say all (2m,n) grids. For that, shape rules 1 and 2 are changed to those shown in Figure 16. Clearly, any class of (m,n) grids can be constructed by suitably changing shape rules 1 through 4.

Notice too that the shape rules can be easily modified to construct the initial shape for the shape grammar described in the preceding section. This shape grammar is shown in Figure 17.

It is worth making one last comment. The grammar shown in Figure 3 was implemented without modification. This resulted in several unexpected subshape instances to be discovered when shape rule 9 was applied. Rule 9 was subsequently modified to that shown below:



This eliminated the surprise (though perfectly valid) subshape instances. This simple grammar provides an example where the tradeoff between time and space costs must be considered when using the program, by supplying more than the minimum necessary information in the shape rules, the computation required to search for subshape instances is substantially reduced.

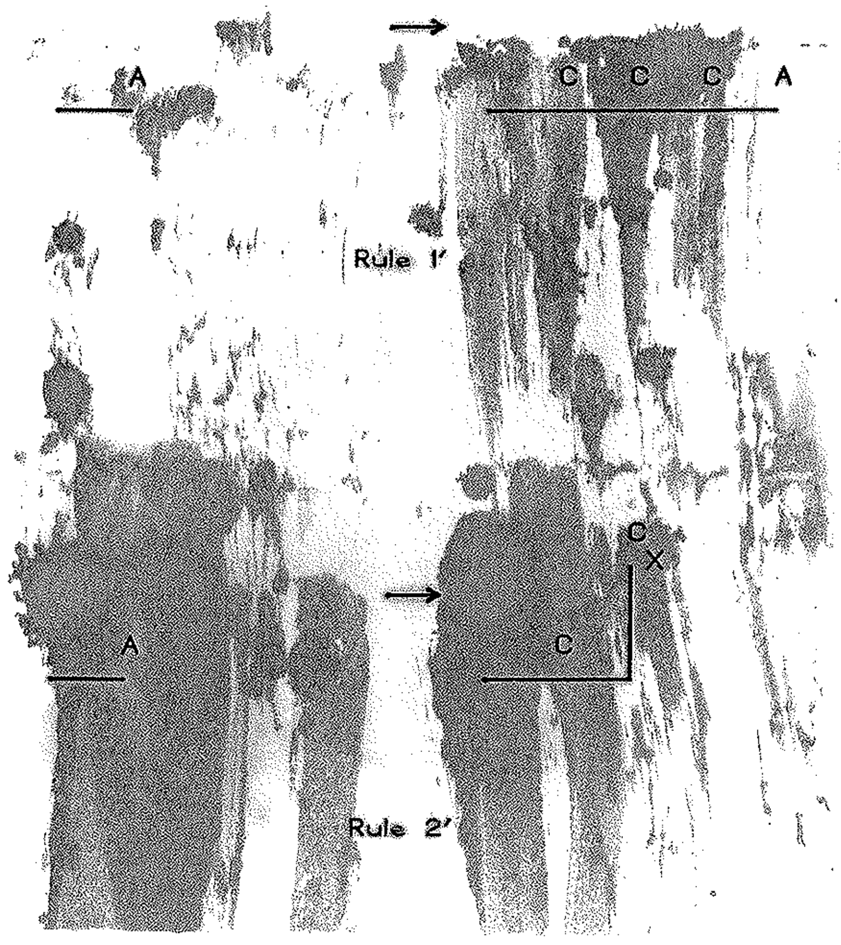


Figure 16: Modified shape rules 1 and 2 to generate all (2m,n) grids

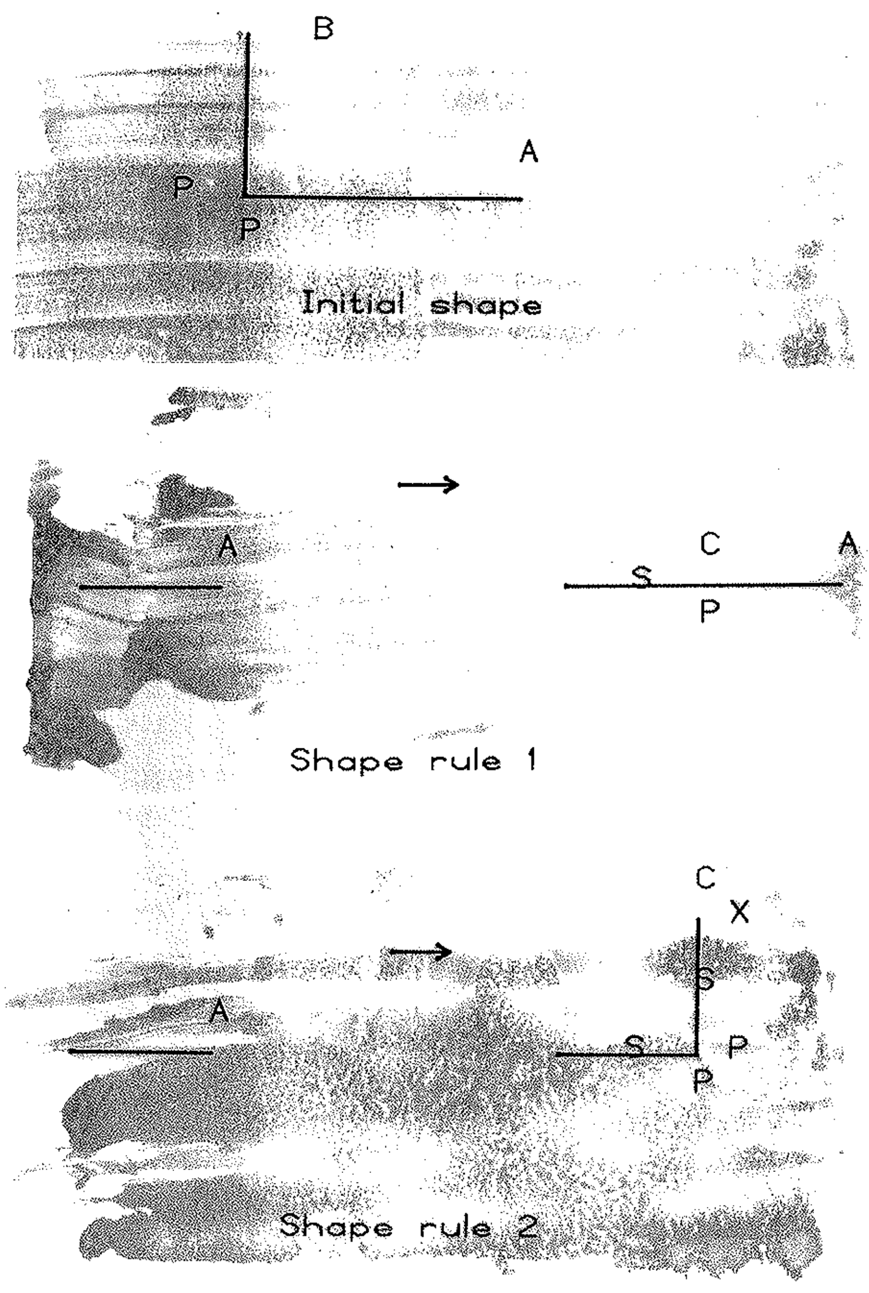


Figure 17: Changes to shape grammar to generate the initial shape for Earl's grammar

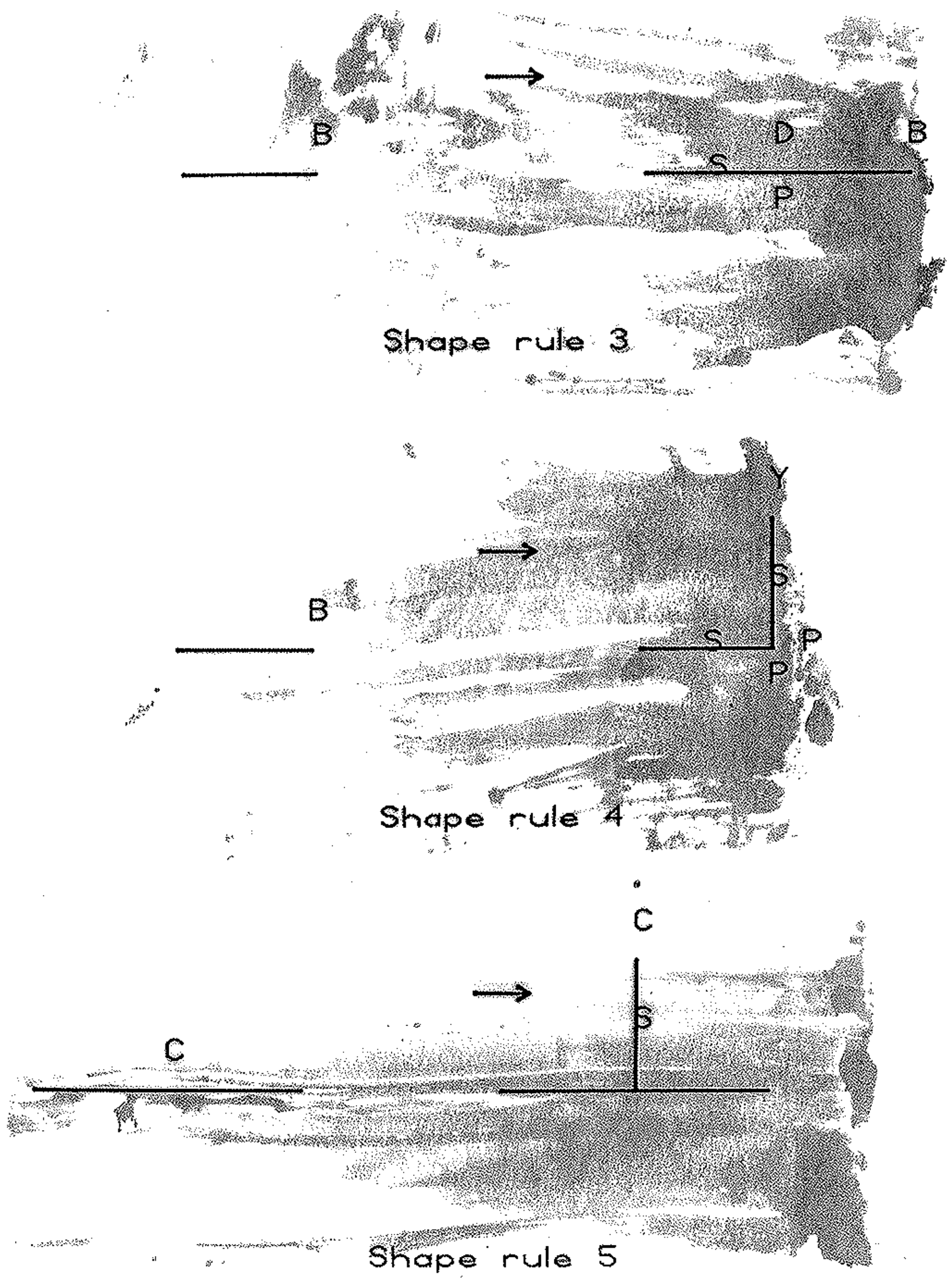
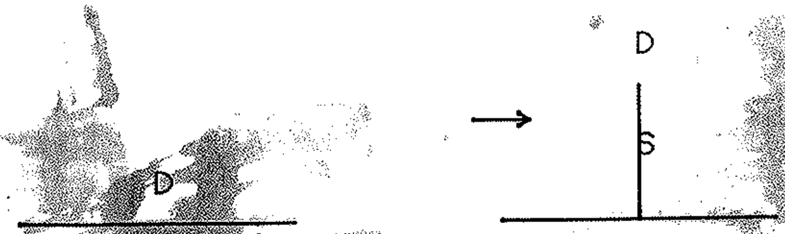
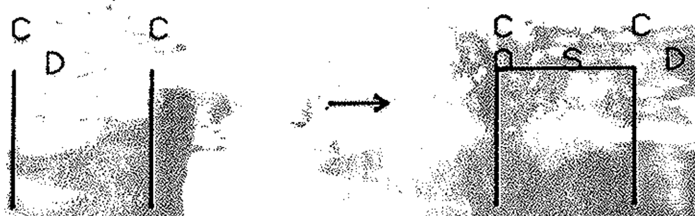


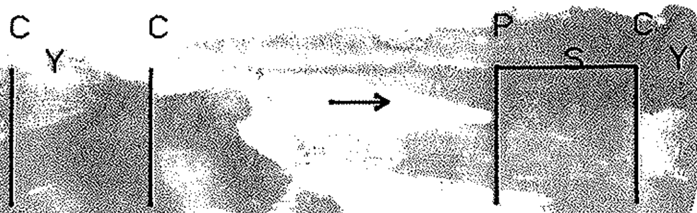
Figure 17 (continued)



Shape rule 6



Shape rule 7



Shape rule 8

Figure 17 (continued)

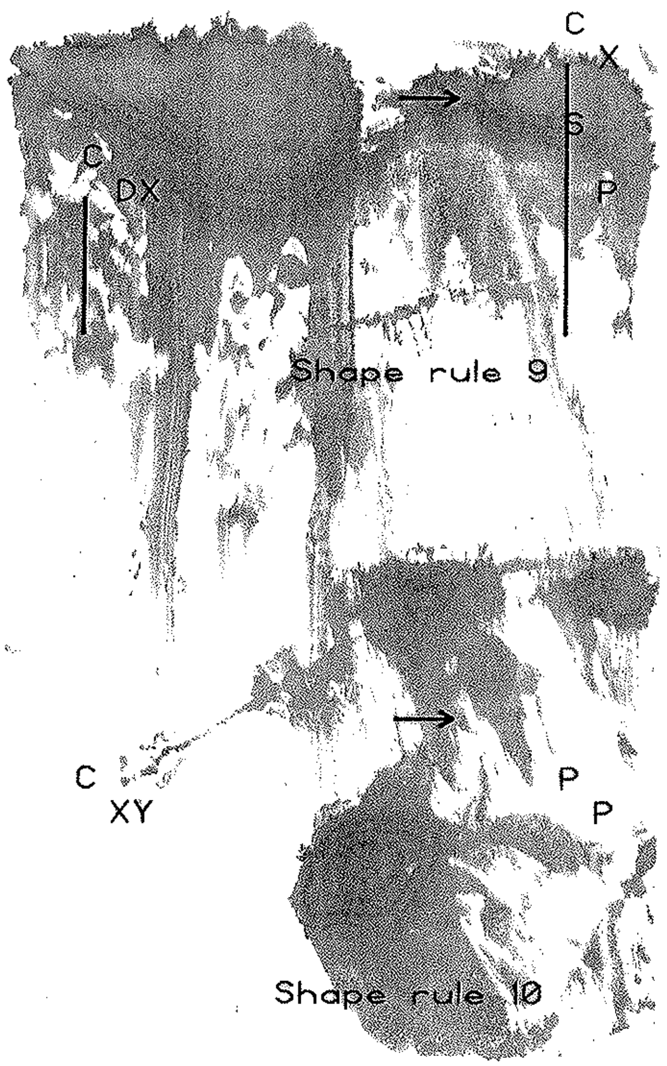


Figure 17 (continued)

Concluding remarks

The design of the shape grammar interpreter SGI has been presented. The program implements any shape grammar based on a complete and uniform representation of rational two dimensional shapes. Shapes can be generated by applying any user-specified sequence of shape rule application. SGI has been tested on a few sample shape grammars.

The correctness and efficiency of the shape algorithms employed by the program have been established in (Krishnamurti, 1980, 1981).

The program has been implemented in Fortran on a VAX11/780 computer. The program consists of 149 routines including the graphics and debug trace routines, and occupies approximately 240K bytes of memory excluding the storage required to house the list structures and shape rules. At present 200K bytes of memory have been allocated for this purpose. The program has been thoroughly tested. Each command has been verified to execute correctly for all possible spatial conditions. The response time for each command is, for all practical purposes, instantaneous except for the disk file read command which is dependent on the system disk driver routines.

The program fulfills a need for a general shape grammar interpreter that has existed for the past 5 or 6 years. To my knowledge, SGI is the second computer program ever designed to operate on shape grammars. The first program was written by Gips (1975). Gips' program was restricted to shape grammars with two shape rules and to shapes which are closed polygons.

Nonetheless, his experience with the compuational aspects of shape grammars through publications and personal communication, has had considerable influence on the design of SGI. While the shape algorithms and the data structures used by SGI differ radically from those employed by Gips' program, his design for a user-friendly shape grammar interpreter command set served as a model for the SGI language and helped to reduce what would otherwise have proved very hard work.

It is early days. There has been limited interaction with other users whose initial response has been encouraging. With greater user interaction one can expect more constructive comments and suggestions which will result in improvements and modifications to the program. This is precisely the reason for including the U command in the SGI language. One of the aims in designing a shape grammar interpreter is to extend our understanding of machine generation of shapes. The feedback from using SGI or any other shape grammar program can only lead to the design of better shape grammar programs with sufficient power for inclusion in large CAD and CAAD packages.

The program does have its limitations, stated earlier in the paper. It also lacks some other features which have become apparent through use such as commands for global deletion and substitution of subsets of labels from the labeled points. It is expected that these features along with others suggested by potential users will be incorporated in future versions of the program. I believe that SGI will prove to be a valuable tool which can contribute towards an appreciation of the joys and frustrations involved in the computer generation of shapes.

References

Earl C F, 198Ø,
"Rectangular shapes"
Environment & Planning B 7 pp311-342

Gips J, 1975,
Shape Grammar and their Uses: Artificial Perception, Shape Generation and Computer Aesthetics
(Birkhauser Verlag, Basel and Stuttgart)

Krishnamurti R, 198Ø,
"The arithmetic of shapes"
Environment & Planning B 7 pp463-484

Krishnamurti R, 1981,
"The construction of shapes"
Environment & Planning B 8 pp5-4Ø

Krishnamurti R, 1982,
"SGI User Manual"
Technical Report: Centre for Configurational Studies

Stiny G, 198Ø,
"Introduction to shapes and shape grammars"
Environment & Planning B 7 pp343-351

Stiny G, Mitchell W J, 1978,
"The Palladian grammar"
Environment & Planning B 5 pp5-18

Stiny G, Mitchell W J, 198Ø,
"The grammar of paradise: on the generation of Mughal gardens"
Environment & PLanning B 7 pp2Ø9-226

Weissman Knight T, 1981,
"The forty-one steps"
Environment & Planning B 8 pp97-114