Mathl. Comput. Modelling Vol. 16, No. 4, pp. 115–129, 1992
Printed in Great Britain.

0895-7177/92 $5.00 + 0.00

AN INTERACTIVE GRAPHICS BASED VISUAL MODELLING TOOL

SUKUMAR RATHNAM*
Management Science and Information Systems Department, CBA 5.202
University of Texas at Austin, Austin, Texas 78712-1175, U.S.A.
T. MADHAVAN
Production and Quantitative Methods
Indian Institute of Management, Ahmedabad, India

(Received September 1991)

Abstract — Interactive computer based graphics systems are increasingly being used in modelling environments. In these situations the main design goals for software systems, which support visual output display, are: the ease of prototyping, an easy-to-use interface, a large set of graphic primitives and a powerful image composition scheme. This paper first presents the theoretical basis, and then some of the design issues, for a system we built to aid the visualization of the graphical output from a modelling process. The paper concludes by describing applications of the software in various modelling contexts. The main contribution of our research is that it introduces new and powerful image composition techniques as opposed to providing geometric primitives.

The graphical output of any formal mathematical model can be described in terms of either an iterated function system, or a recursive function. We have, in our research, used the recursive function approach to describe graphical objects. This is achieved by creating a context free grammar to describe any recursive function. A picture grammar, that is equivalent to the context free grammar, is then developed. The software implementation section of our work focuses on building an interpreter for the resulting language.

The main theoretical issues that we deal with in the paper are the design of the recursive functions and context free grammars for visual modelling tools. The main implementation issues that we deal with in the paper are the design of the dynamic data structures, the LOGO-like command nature of the input interface, the context sensitive help system, the design of complex graphic primitives like splines and panelled polygons, and the creation of composite images using the notion of recursive replacement terminating in primitive geometric structures.

The current implementation of the tool has been used in several different contexts. Among them are: the modelling of crystal growths, the animation of dynamic processes (for slides in commercials) and the generation of fractal images (to aid in the creative design of wallpaper and textile patterns).

1. INTRODUCTION

Interactive computer graphics are one of the best available tools for both visualizing output from a modelling process and creative design. The availability of high resolution, bit mapped graphics workstations has made it possible to create high quality images in real time. There has, consequently, been an explosion in the number of tools available to use as display routines.

In a modelling environment there are three main requirements for visual modelling tools. These are:

1. The capability of a system has to support image composition rather than in drawing CAD/CAM-like wire diagrams and geometric figures.

*We would like to thank Mr. J.R. Varma, currently a faculty member at the IIM Ahmedabad for his help. Thanks are also due to Prof. Nitin Patel of the IIM Ahmedabad for his constructive suggestions. This project would not have been feasible without the computational resources of the IIM Ahmedabad and the University of Texas at Austin. We would like to thank everyone at the computation center for their help and co-operation.

This work was supported in part by an IBM doctoral fellowship and in part by a University of Texas at Austin Graduate Fellowship.

2. The support for the incremental specification of models.
3. The ability to show how closely-related models are related and differentiated along a set of dimensions. An example of this would be to compare the underlying structural similarity of curves like the "C curve" or the "Dragon curve" to that of curves like snowflakes.

These three requirements motivate the need to develop tools that support the visualization of modelling, through visualization of generic functional forms rather than mere geometric specification. Several attempts have been described in both, the popular press [1] and the academic literature [2] to create images by plotting recursive functions or recursively defined point sets. Unfortunately none of them have resulted in general image composition schemes. The focus so far has been to model each image individually rather than as a special case of a general system or on providing a toolkit that gave a modeller a large set of primitive drawing features with no image composition functions.

The pioneering work of Mandelbrot [3] has shown that a geometry (fractal geometry), which can be used to describe any graphical object, exists. Further, he showed that such a geometry leads to the construction of fractals. It is this theory that supports the possibility of creating a very general purpose visual modelling tool. The creative use of the general purpose concepts of Mandelbrot's fractal geometry in interactive computer graphics can be seen in [4].

There have been, in the past, two general approaches to the problem of the interactive generation of composite images using functional' image-composition techniques:

- Barnsley [2] used the concept of an iterated function system (IFS) to plot fractals. An IFS consists of a metric space (X, d) together with a finite set of contraction mappings w_n: $X \rightarrow X$ with contractivity factors s_n, $n \in [1, N]$.
- Smith [5] investigated the use of L-grammars to describe images but the grammar he developed concentrated only on graftals and figures that represent natural trees.

Any image that has to be defined can always be described as a point set or as a function. Every computable function is computable in the domain of recursive functions. Hence, to describe any image, all that is required is an equivalent definition for it in terms of recursive functions. As recursive functions can always be expressed in terms of an equivalent grammar, it is possible to describe any image in terms of a grammar.

The basis for our tool, to support the visual modelling, lies in the specifications of fractals and IFS as the sentences of formal language. For our system we have chosen an unrestricted context-free graph grammar. The role of the grammar is in merely generating sentences of the language. It is the interpretation of the sentence that results in the plotting of the final image.

Section 2 of this paper presents the formal specifications for the grammar that we developed. Section 3 describes the operation semantics for the resulting language. Section 4 deals with some of the major implementation details of the tool. Section 5 contains our conclusions and directions for future research. To show the ease with which the tool can be used, a set of sample figures, along with their parameterization, is presented in Table 1. The Appendix presents a short overview (for those readers who wish to understand the mathematical basis) of the integration of fractal theory with that of recursive functions.

2. FORMAL SPECIFICATIONS

The specification of the visual output from a modelling process, in terms of recursive functions, results in the creation of fractals [6]. While fractility itself is interesting, it is not the main reason for this paper. The database amplification properties of such systems, controlled by a finite and small set of grammar rules [5], can result in a wide variety of images. (The sample figures whose specifications are given in Table 1 illustrates this.) It also provides the basis of a visual modelling tool which meets the three criteria set in the the previous section.

Table 1. Some sample figures and their key parameters.

Sample Fig.	Name	Scheme	Primitive	Depth
1	Peano	Complex	Line	4
2	Cross	Complex	Stars <4>	4
3	Snowflake	Diagonal <6,3>	Stars <8>	2
4	Tree	Complex	Line	5
5	*C* Curve	Complex	Line	13
6	*L* curve	Complex	Line	5
7	Dragon Curve	User	Line	11
8	Close Pentagon	Diagonal <5,1>	Polygon <6>	4
9	Koch	Complex	Line	4
10	Buckle	Edge <8,1>	Blocks <3>	3
11	Snowflake	Diagonal <7,2>	Polygon <4>	2
12	Hexagrill	Diagonal <6,1>	Polygon <3>	4
13	Mountain	Diagonal <3,1>	Polygon <3>	5
14	Lace	Diagonal <12,3>	Stars <8>	2
15	Mat	Diagonal <4,1>	Stars <12>	4

2.1. Requirements for the Generation of Recursive Objects

All type-0 grammars [7] have an equivalent representation in Turing Machines and, as there always exists a Turing Machine representation for every recursive function, we can conclude that

> "If we want a generalized image drawing system, capable of drawing any fractal and hence any object, it would be necessary and sufficient to have one which has graphics operators, expressed in terms of grammars, which have a congruence with their equivalence in recursive functions."

The functions for which equivalent graphic operators are required are:

1. **ZERO**: **ZERO** $(X) = 0$
2. **SUCC**: **SUCC** $(X) = X + 1$
3. **SEL**: **SEL** $(n, k) = I_k^n(X_1, X_2, \ldots, X_n) = X_k \quad (k \leq n)$
4. **COMP**: The composition strategy is defined as: If f is an n-ary function (that is, it is a function with n arguments) and g_1, g_2,..., g_n are k-ary functions, the composition or substitution $f(g_1, \ldots, g_n)$ is a k-ary function where:

$$f(g_1, g_2, \ldots, g_n) = f(g_1(x_1, x_2, \ldots, x_k), g_2(x_1, x_2, \ldots, x_k), \ldots, g_n(x_1, x_2, \ldots, x_k))$$

5. **PREC**. The primitive recursion strategy defined as: Given any n-ary total function g and an $(n+2)$-ary total function h, then f the $(n+1)$-ary total function defined by primitive recursion, is:

$$\begin{aligned} f(0, x_n, \ldots, x_1) &= g(x_n, \ldots, x_1) \\ f(m+1, x_n, \ldots, x_1) &= h(m, f(m, x_n, \ldots, x_1), x_n, \ldots, x_1) \end{aligned}$$

where h is a composite function.

6. **MINIMALISATION**: This operation introduces, for each total function $f(y, x_1, \ldots, x_n)$, a new function $h(x_1, \ldots, x_n)$, whose value for given $(x_1, \ldots, x_n)$ is the least y such that

$$\begin{aligned} f(y, x_1, \ldots, x_n) &= 0, \qquad \text{and} \\ h(x_1, \ldots, x_n) &= \min y \qquad \text{such that} \quad [f(y, x_1, \ldots, x_n) = 0]. \end{aligned}$$

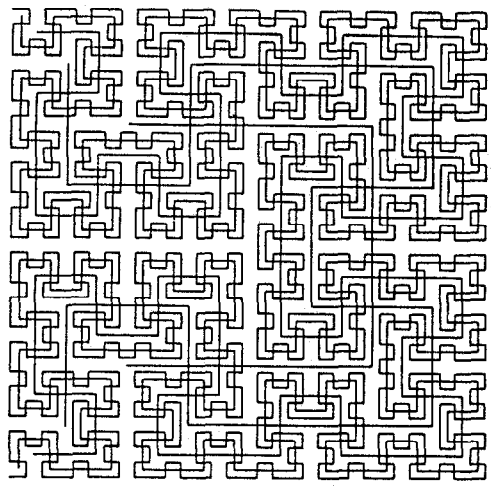

Figure 1. Peano's Curve.

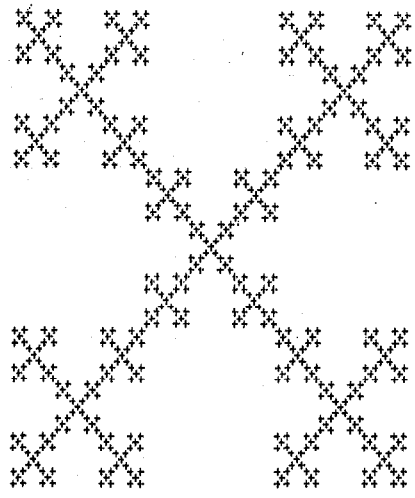

Figure 2. Cross Curve.

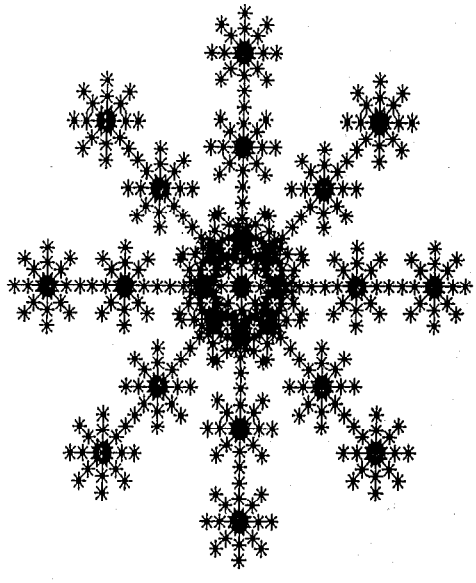

Figure 3. The Snowflake Curve.

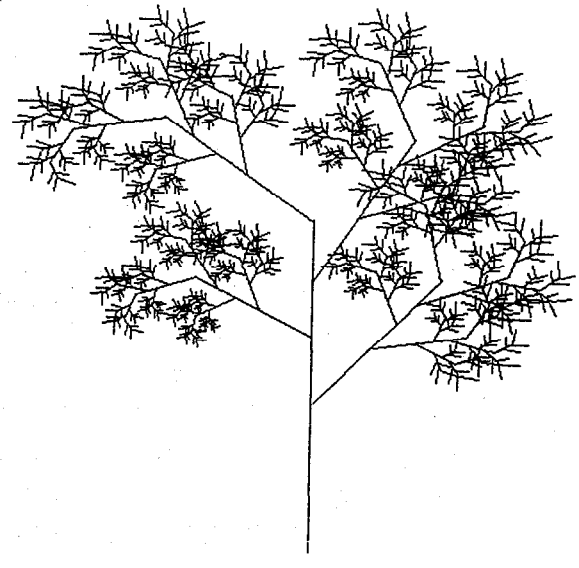

Figure 4. The Tree Curve.

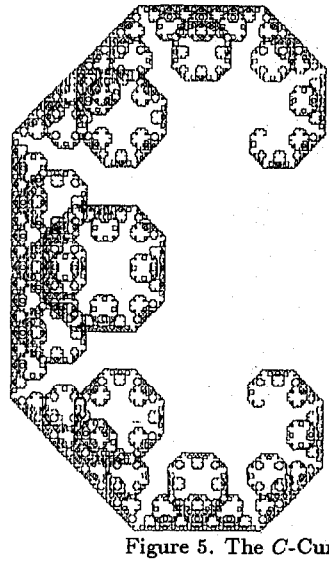

Figure 5. The *C*-Curve.

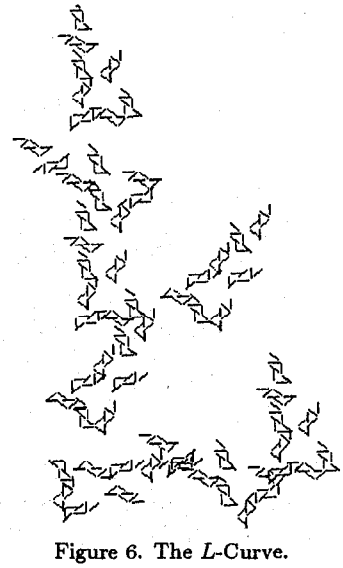

Figure 6. The *L*-Curve.

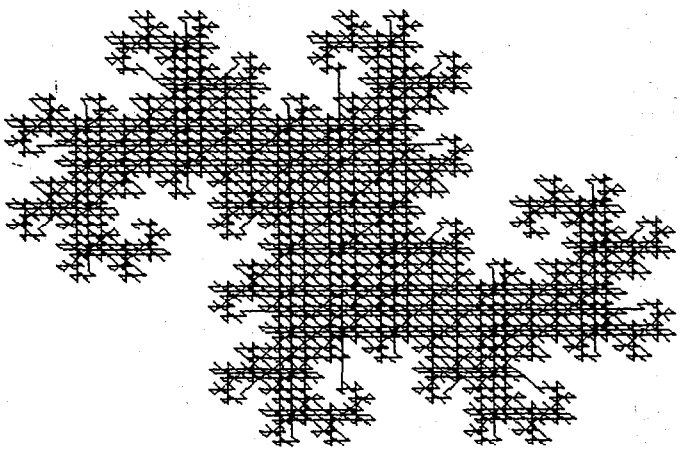

Figure 7. The Dragon Curve.

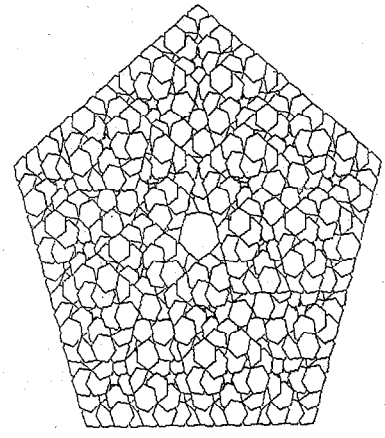

Figure 8. The Close Pentagon Curve.

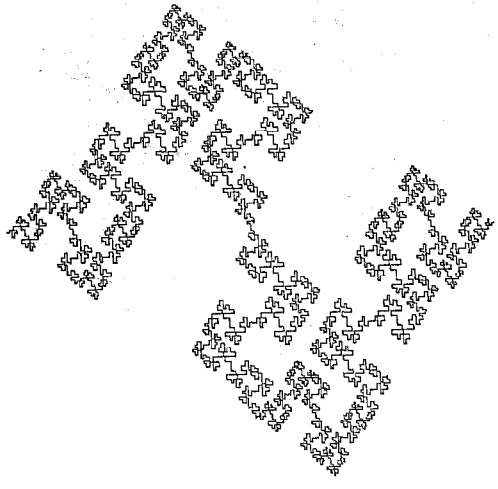

Figure 9. The Koch Curve.

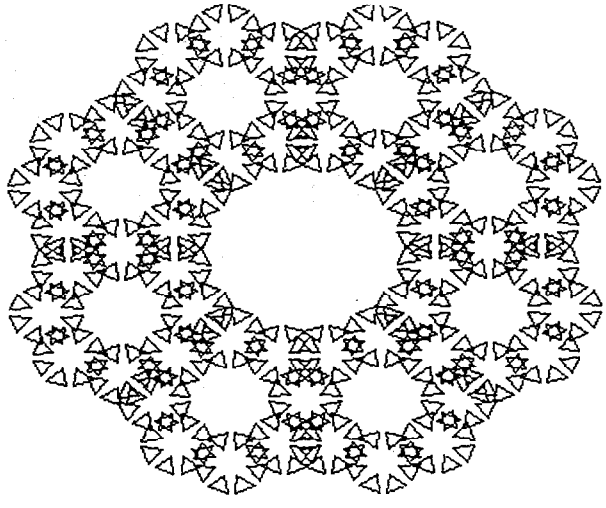

Figure 10. The Buckle Curve.

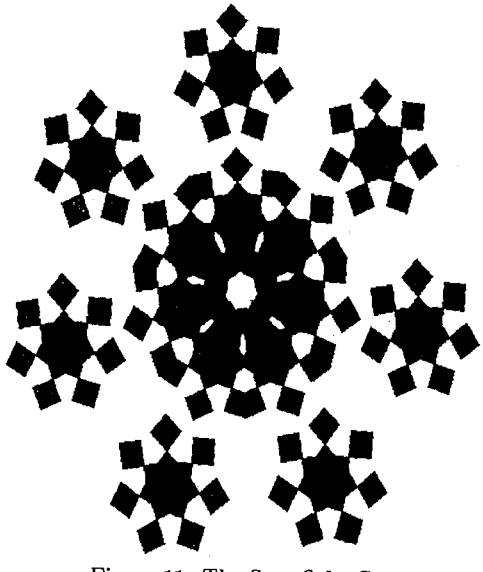

Figure 11. The Snowflake Curve.

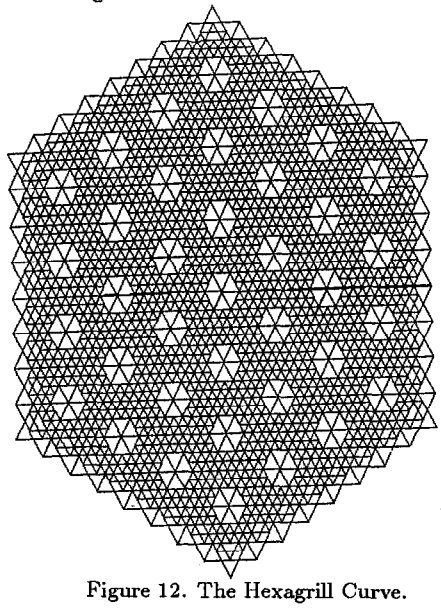

Figure 12. The Hexagrill Curve.

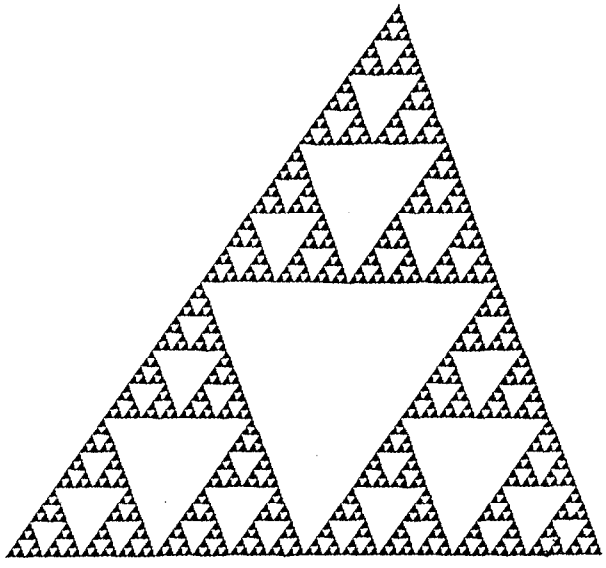

Figure 13. The Mountain Curve.

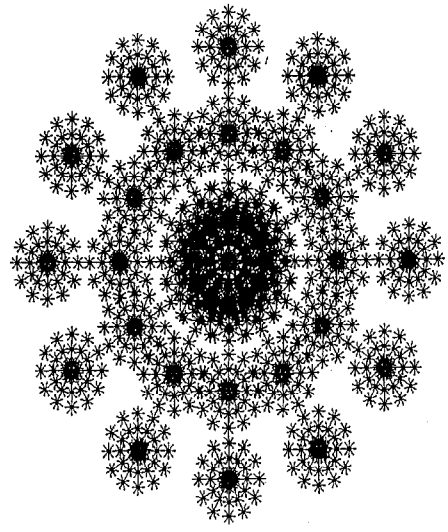

Figure 14. The Lace Curve.

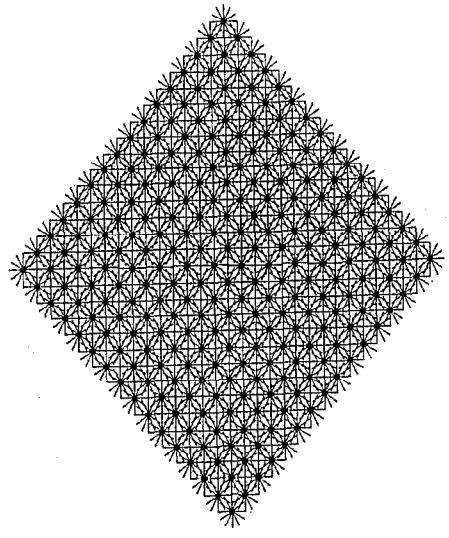

Figure 15. The Mat Curve.

The class of functions generated by the repeated application of the base strategies (**COMP**, **PREC** and **MINIMALISATION**) starting with the base primitive recursive functions (**ZERO**, **SUCC** and **SEL**) results in the set of partial recursive functions. Exclusion of the minimalisation operator leads to the class of general recursive functions. General recursive functions provide a necessary and sufficient condition required to model deterministic images. Hence, these are the functions with which our grammar has an equivalence.

2.2. The Formal Specification for the Language

We now present the formal language for our picture grammar in three parts: a definition of the grammar, a summary of the Extended Backus-Naur Form (EBNF) notation used, and the formal syntax of the grammar using EBNF.

2.2.1. The Following Type 0 Grammar is Used in the Specification:

$$\mathtt{G} = (\mathtt{N}, \mathtt{T}, \mathtt{P}, \mathtt{S})$$

where:
G = The picture grammar
N = A finite set of non-terminal shape patterns (the intermediary shapes)
T = A finite set of terminal shapes (or primitive shapes)
P = A set of production rules (the replacement scheme)
S = An initial symbol (the starting replacement figure which is assumed to be a straight line)

2.2.2. EBNF Notation Used in the Description of the Picture Grammar

Symbol	Interpretation
=	is defined to be
|	alternatively
.	end of production
{X}	0 or more instances of X
(X|Y)	a grouping: either X or Y
"XYZ"	the terminal symbol XYZ
Meta_id	the non-terminal symbol Meta_id
Scheme	the start symbol

2.2.3. Formal Syntax of the Picture Grammar Using EBNF

```
Scheme           =  (Edge | Diagonal | User | Complex).
Edge             =  "edge_fractal" integer integer Edge {Edge}.
Edge             =  "edge_fractal" integer integer Primitive.
Diagonal         =  "diag_fractal" integer integer Diagonal {Diagonal}.
Diagonal         =  "diag_fractal" integer integer Primitive.
User             =   Figures {Figures}.
Figures          =  (User | Line).
Figures          =   Primitive.
Complex          =   Objects {Objects}.
Objects          =  (Complex | Primitive).
Objects          =   Primitive.
Primitive        =   Simple_prim | User_prim.
User_prim        =   Simple_prim {Simple_prim}.
Simple_prim      =   Circle | Line | Star | Polygon | Block.
Line             =   Straight_line | Smooth_line.
Polygon          =   Unfilled_polygon | Filled_polygon.
Circle           =  "draw_a_circle" Specs.
Star             =  "draw_a_star" Specs Sides.
Straight_line    =  "draw_a_straight_line" Specs.
Smooth_line      =  "draw_a_smooth_line" Specs B.
Filled_polygon   =  "draw_a_filled_polygon" Specs Sides Code.
Unfilled_polygon =  "draw_a_unfilled_polygon" Specs Sides Code.
Specs            =   X Y Size Angle Color.
X                =   real.
Y                =   real.
Size             =   real.
Angle            =   real.
Color            =   integer.
Sides            =   integer.
Code             =   integer.
B                =   matrix.
Depth            =   integer.
```

The matrix B is the constant matrix

$$\begin{bmatrix} 1 \\ - \\ -6 \end{bmatrix} * \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix}.$$

For simplification purposes the syntax of the terms integer, real and B have not been defined.

2.3. Proof of Grammar Type and Equivalence

The following condition for type-0 grammars can be trivially shown to be satisfied:

If

V = N U T is the vocabulary of the language
and V* = The set obtained by the closure on V
and N ^ T = {} the null set

Then

P is a finite set of ordered pairs (x,y) where x,y are strings
over V* and x has at least one symbol in N
and S = the starting symbol and S ∈ N

3. THE SEMANTICS OF THE LANGUAGE

This section qualitatively deals with the operational semantics of our grammar.[1] The grammar creates both sub-fractals (curves which look like fractals but are not) and pure fractals. In congruence with the languages described by Mandelbrot [3], we assume that the initial symbol for our production rules is a straight line. At this point it must also be noted that we have used a system of finite interpretation which does not allow randomness.[2]

3.1. The Replacement Scheme <Scheme>

The replacement scheme specifies the interpretation of the non-terminals in the grammar. This is accomplished by specifying the pattern along which recursive replacement will take place. Hence the replacement scheme creates the final form of the fractal image. Images drawn with the same replacement scheme but different primitives will have the same structure and form. However, their building blocks will be different (as shown by the 2 "Snowflake curves"). Conversely, images drawn with the different replacement schemes but the same primitives will have different forms and structures but will be composed of the same building blocks. (This is shown by examining the "*C*-curve" and the "dragon-curve.")

The system has an important property in that inter-connectivity during replacement and replacement itself do have an orientation attached to them. As this implies that the geometry is carried with the production rules, they must work under affine transformations [5].

The replacement scheme can be specified by:

EDGE <Edge>. Selecting this option implies that the replacement figure(s) are computed by the following steps:

1. The number of sides of a regular polygon (m) is selected. The next level of recursive figures will be strung along the edges of this polygon.
2. The number of replacement figures (n) for each edge of the original regular polygon is then selected.

[1] We choose to present this section using operational semantics rather than axiomatic or denotational semantics mainly for the purposes of edification.

[2] In this section the use of the notation <ID> refers to the token whose properties are being defined.

3. An initial polygon will be oriented such that one of the lines joining its center to a vertex is coincident with the given initial orientation vector.
4. The vertices of the initial polygon are computed and the equations of the straight lines joining adjacent pairs of vertices i.e., the edges of the polygon are determined. Along these lines are the centers of the $n*m$ replacement figures placed and their orientations are those of the slopes of these edges. The procedure edge is recursively called for each of the $n * m$ replacement figures until a termination condition is reached.
5. On termination a primitive is drawn.

DIAGONAL **<Diagonal>**. Selecting this option implies that the replacement figure(s) are computed by the following steps:

1. The number of sides of a regular polygon (m) is selected. The next level of recursive figures will be strung along the diagonals of this polygon.
2. The number of replacement figures (n) along each of the lines joining the center of the polygon to the vertices (the diagonals of the regular polygon) is then selected.
3. An initial polygon will be oriented such that one of the lines joining its center to a vertex is coincident with the given initial orientation vector.
4. The vertices and the equations of the diagonals are determined. Along these diagonals are placed the centers of the $n * m$ replacement figures and their orientation is that of the corresponding diagonal. The procedure diagonal is now called recursively for each of the $n * m$ figures until the termination condition is reached.
5. On termination a primitive is drawn.

USER-DEFINED **<User>**. Under this option the user after choosing a data entry mode explicitly states the relationships between the number and positional parameters of the lower order figures and the higher order figure. Each lower order figure can be either a straight line or a recursive call. On termination of recursion, primitives get drawn. In this scheme for every lower order element, the same primitive is drawn at the termination of recursion and all lower order elements recurse down to the same level. This option can create those images which require a tree-replaces-segment strategy [5]. Patterns that can be created using this are those defined by the grammar of which **User** is a non-terminal.

COMPLEX **<Complex>**. This is an extension of the above scheme except that each lower order figure can recurse to different levels and each figure can have a different primitive drawn at the termination of each branch of recursion. Additionally this option allows any primitive (not just straight lines) to be drawn in linking the lower order figures. This is the most general case and under suitable parameters can be used to create the same effect as any of the replacement schemes **[<Edge>,<Diagonal>,<User>]**. This option is a case of a graph replacing graph form and can be used to create figures like the Carpenter mountain [5]. Figures that are created using this option have a grammar in which **Complex** is a non-terminal.

3.2. The Primitive Specification **<Primitive>**

The primitive specification specifies the interpretation of the terminals of the grammar. This is accomplished by specifying the type of the graphical object to be drawn at the termination of recursion. While the position, orientation and size of this figure are obtained from the replacement scheme the form of the object is derived from this primitive specification. This parameter can be specified as:

CIRCLES **<Circle>**. Under this choice circles are drawn as the primitive on the termination of recursion.

LINES **<Line>**. This option makes the primitive drawn at the termination of recursion a straight line.

STAR `<Star>`. Here the primitive shape drawn at the termination of recursion is a figure which looks like a star or the diagonals of a polygon.

REGULAR POLYGONS `<Polygon>`. If we choose this option we have to further characterize it with:

1. The number of sides in the polygon.
2. The choice as to whether we want to fill the interior of the polygon or not. If this is set to true, then the polygon has its interior filled with a pre-defined pattern.
3. The choice of a library from which we choose the pattern to fill the interior of the polygon.

When these are chosen, the primitive drawn is a regular polygon whose shape and interior pattern are determined using the above parameters. The polygon will be oriented such that a diagonal is coincident with the initial orientation vector.

BLOCKS `<Block>`. This is equivalent to the previous option except that the polygons are drawn such that their vertices are rotated. Hence the effect is the same as in the above case except that for polygons of even sides an edge of the polygon will be orthogonal to the initial orientation vector and for polygons of odd sides a diagonal of the polygon will be orthogonal to the initial orientation vector.

USER-DEFINED `<User-prim>`. Here the user specifies as to what form the primitive should take. Primitives are composed as an arbitrary set of closed polygons and inter-connecting lines which can have any of these properties:

1. Panelling: This causes the primitive that is drawn to appear as a polygon with its interior filled. The details of the panelling features are identical to those for `<Polygon>`.
2. Line smoothing: Here, instead of joining successive points with straight lines, a cubic spline is drawn to fit them.

3.3. The Depth of Recursion

This parameter indicates how many times the self duplication process continues before primitives are drawn. As such, this is not part of the semantics. It merely controls the way the final sentence is interpreted. For complex schemes this has two implications:

1. After what depth of recursion the primitive is to be drawn for those elements of the grammar which are non-terminals and have either `Complex` or `Objects` as part of their grammar.
2. For how many levels of recursion the interconnecting figure, which is described in the same way as a primitive is, must be drawn for those elements of the definition which are primitive-defining ones.

4. SYSTEM FEATURES AND IMPLEMENTATION ISSUES

This section deals with the options the user has in terms of controlling the system environment and several of the key implementation issues that came up during the course of our research. Figure 16 carries a schematic of the system architecture.

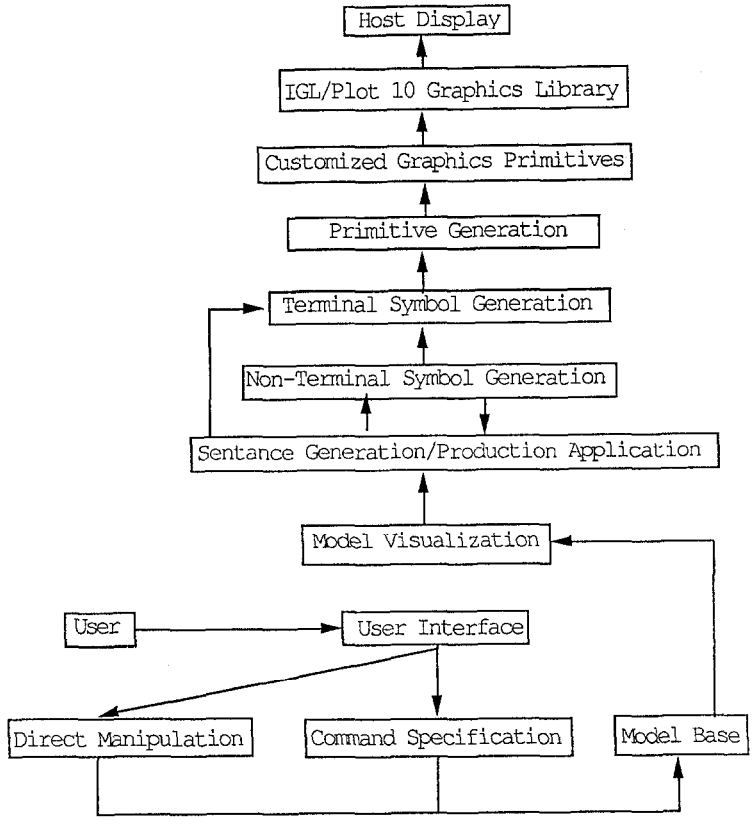

Figure 16. Architecture of XXXRUM.

4.1. The Data Structures Used in Representing Recursive Descriptions

The parameters which go into composing standard and pre-defined forms and primitive shapes are represented as integers and real numbers. The major issue in the design of the data structures was in the case of user-defined parameters. As the processing of the data structure was, at each level of recursion, sequential and continuous starting from the first element to the last, with only one pass being made, the choices were between an array, a stack and a singly-linked linear list.

The choice of a singly-linked linear list was made because the algorithm was intrinsically recursive and dynamic and, hence, required a recursively-defined dynamic data structure. The interpreter was implemented using recursion rather than iteration and an explicit stack. This design chosen as an analysis, indicated that most of the running time was taken up by the plot procedures and the baud rate limitations between the host system and the terminal. Hence some inefficiency in running time and memory usage could be afforded for gaining algorithimic clarity and ease of implementation. An empirical analysis indicated that even in the worst case, the running time would not go up by more than 2%.

4.2. Graphic Control Options

These options describe how users can control the actual graphical display.

COLOR. Different elements of the picture definition, at termination, can draw different primitives. To show the true nature of the construction of the final image and to increase the quality of the images produced, color has been made a parameter in both the primitive definition and the composition schemes.

Users can choose colors from a palette specified in a color table. This pre-defined color table is loaded during the configuration stage. The table definitions of each color is in terms of three co-ordinates. Either the red, blue, green or the cyan, magenta, yellow or the hue, lightness, saturation standard can be used to specify the axis of these co-ordinates.

LINE SMOOTHING. When this option is exercised then every line joining a pair of points is drawn such that it is not a straight line but a line drawn by fitting a cubic spline. This gives the impression of a smooth line joining the points (like a geographical contour) rather than that produced by the point to point joining by straight lines.

The algorithm used to perform line smoothing is:

$$C(t) = [t^3\ t^2\ t^1\ 1] * \begin{bmatrix} 1 \\ - \\ 6 \end{bmatrix} * \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} * \begin{bmatrix} P_{i-1} \\ P_i \\ P_{i+1} \\ P_{i+2} \end{bmatrix} \quad \text{for } 2 \leq i \leq n.$$

SEGMENT CREATION. This causes the output device to shift to a raster scan mode. Raster scan mode display will allow a user to perform geometric transformations on the final output. In addition, it allows hard copies to be taken. This also permits the user to perform zooms on the final image.

4.3. The User Interface

We now describe the interface that was built for our system. In modelling systems as well as creative design environments the main features that an interface should possess include: ease of use, the ability to take direct, incremental, reversible actions and a mechanism that allows the system to be easily hooked to other systems. Our interface satisfies these requirements through the use of a sophisticated input description language and the data retrieval system.

Commands are the primary mode of specifying grammars in the system. To ease the use of the language, command completion is built in as a part of the system. Syntax errors are trapped at input and error messages displayed in a status window. Simple semantic errors are also trapped at input. Such errors include specifying polygons with less than 3 edges.

The interpreter works by parsing (a standard LA-LR strategy) its input into a sequence of tokens and then constructing a sentence in the appropriate grammar. Parameters for the non-terminal symbols and the recursive schema can be specified in a variety of modes. The use of a grammar, as the basic unit for modelling, greatly helped in the interface design as the interface can now be looked on as a language input mechanism.

Three input mechanisms have been implemented. These include:

4.3.1. Direct Functional Specification Using a Textual Interface

This choice results in the user giving his input through the terminal keyboard. There will be prompts for specific parameters as to the relative size, the relative orientation and the relative starting co-ordinates that define the scheme or the sequence of move and draw operations that go into describing a primitive shape. Incorrect parameters values and other syntactic mistakes will be trapped immediately and the user will have to re-input the parameters. The interface mode is basically parametric and uses command completion. We felt that given the nature of the application this was the most appropriate way to design the interface.

4.3.2. Incremental Functional Specification Using a Direct Manipulation Graphical Interface

In this mode the user gives his input in the form of digitized information using a device like the mouse/tablet or a joystick. As the input is received the corresponding figure is drawn on the terminal screen. This mode is very similar to keyboard mode input except that the parameters are computed through the direct manipulation of graphical objects by the user instead of co-ordinate values. The second difference is that this mode of specification is incremental in terms of building the underlying functional description for models. To aid the easy use of a graphic input device, menus appear indicating as to what key on the mouse/joystick stands for what parameter. A problem with this mode of input is directionality and orientation for graphical objects for it becomes difficult to interpret the slope of a line (does the user want to create a slope theta or a slope of $\theta + 2\pi$).

4.3.3. Model-Base

This choice results in the data for the functional parametric characterizations being computed from a model base. This stores the model and its underlying representation as a recursive function. An interactive customizable model-base browser has been built. The motivation in providing this facility was that incremental changes could be easily built and a form of versioning provided. The interactive editor was built using the concept of a structure editor generator. The use of tools like the Cornell synthesizer generator are facilitated because of the underlying representation of model descriptions in terms of grammars and recursive functions.

4.4. The Help Procedure

The help routines are available at two levels. At the first level, the help is available only when the system is initialized. This help is a continuous page-driven help. The main feature of the help routines is at the second level. This is a context-sensitive help. In this mode the system will automatically display information about the command, what it does semantically and what its side-effects are.

5. APPLICATIONS AND DIRECTIONS FOR FUTURE WORK

The system we have built is now being used in computer-aided design. The ease of use and the symmetry of the fractals that can be produced makes the system highly suitable for use in a creative design environment, like in textile and wall-paper design, where the emphasis is on creating pure symmetric patterns or symmetrically asymmetric patterns. Objects like mountains, clouds, trees, snowflakes, and dragons can be easily drawn. The samples show some of the applications in which the system has been used.

The easily understandable syntax and semantics of the language we have developed make it suitable for use as the basis of a visual modelling tool. The modelling of crystal growths is being carried out on an experimental basis. The main advantage that ensued was that the basic recursive nature of the output was easily presented allowing incremental changes by the selective change of design parameters. The sample figures show how this incremental change affects the final output thus providing a useful tool for sensitivity analysis.

While the previous applications involved the use of the system as static entities, the animation features built in can be used in a dynamic environment to describe a continuously changing scene. This stems from the fact that images are continuously described as they grow giving the impression of animation. A typical example of this kind of use would be in the design of slides for commercials.

An unusual application of the system is in computer-aided instruction (CAI). Wirth [8] has argued that the choice of an initial programming language and the examples used to illustrate concepts, significantly affect the way people think about algorithms and data structures. Recursion is a concept fundamental to mathematics and programming. There is a lack of tools, in general, to teach programming and specifically to teach the concepts of recursion and grammars. This package, through its easy-to-comprehend output interface makes the task of visualizing the process of recursion easy. Figures like the tree help in illustrating how tail-recursive functions behave.

One weakness of the system is that it can model only objects whose Hausdorff dimension is ≤ 2.0. We are currently extending the system to model objects whose fractal dimension is ≤ 3.0. We are extending our grammar to come up with context-sensitive productions which are required to support the higher dimension visualization of models.

From a theoretical perspective we are attempting to formally define the axiomatic semantics of the language. This would allow us to reason about the effects of incremental changes in models.

REFERENCES

1. A.K. Dewdney, Computer recreations, *Scientific American*, (August 1985).
2. M.F. Barnsley, *Fractals Everywhere*, Academic Press, (1988).
3. B. Mandelbrot, *The Fractal Geometry of Nature*, W.H. Freeman, (1983).
4. H.O. Peitgen and P.H. Richter, *The Beauty of Fractals*, Springer-Verlag, (1986).
5. A.R. Smith, Plants, Fractals and formal languages, *Computer Graphics*, **18**(3), (July 1984).
6. B. Mandelbrot, Stochastic models for the earth's relief: The shape and fractal dimension of coastlines and the number area rule for islands, *Proceedings of The National Association of Science United States of America*, (1975).
7. N. Chomsky, *Aspects of the Theory of Syntax*, MIT Press, (1965).
8. N. Wirth, Programming by stepwise refinement, *Communications of the ACM*, (1974).
9. B. Mandelbrot, How long is the coast of Britain ?: Statistical self similarity and fractional dimension, *Science*, (1967).
10. M. Gardner, Mathematical games, *Scientific American*, (December 1976).
11. W. Hurewiezc and H. Willman, *Dimension Theory*, Princeton University Press, (1962).
12. F. Hausdorrf, *Set Theory*, Chelsea Publishing Co., (1962).

APPENDIX

A Note on Fractal Geometry and Dimension Theory

In this appendix we present the basic results from fractal geometry and the link to dimension theory as background reference.

Basic Concepts Derived From Fractal Geometry

Fractals are recursively defined objects. They can be created by plotting graph grammars which are equivalent to partial recursive functions. Traditionally fractals were used to model dynamic situations [9] but because they are abstractions of recursive functions they can be used wherever recursive functions can be used. This explodes the possibility of their use in computer graphics [4].

Fractals have an interesting topological property associated with them. They have a Hausdorrf-Bescovitch dimension (discussed later), formally denoted as $m_p(X)$, which is non-integral. They are continuous at all points but are not differentiable at any point. They have an infinite perimeter, without being space filling in the limit and have properties in between those of the bounding integral dimensions. We notice that for fractals, in the limit, not only is the perimeter infinite but also that the distance between any two points infinite. The curves are not differentiable at any point because we can never be sure of the direction of motion of a particle tracing the 'curve.' We see that for the Peano Curve not only does the contour, in the limit, trace through every point on the plane but also that it passes through some points an uncountable infinite number of times.

Simplistically the fractal dimension of an object can be determined by the following procedure [10]: Imagine that a figure is magnified X times and that Y copies of the original figure fit into the enlarged figure. If the dimension of the object is D then:

$$D * (\log X) = \log Y$$

These procedures give an intuitive definition of fractal geometry based on the growth pattern of recursive objects. Trying the above procedure for straight lines and cubes we get:

1. Magnifying a straight line twice gives us two copies of the original figure. Thus, the fractal dimension $m_p(X)$ is 1.0
2. Magnifying a cube by a factor of two on all dimensions, we get 8 copies of the original figure $\geq m_p(X)$ of a cube is 3.0

Space filling curves in the limit have integral solutions and hence integral dimensions but recursive curves like the "Snowflake" and the Cantor middle third set do not have an integral Hausdorrf-Bescovitch dimension.

Basic Notions Derived From Dimension Theory

Intuitively dimension can be thought of by the following process: Consider a square. It can be covered by arbitrarily small bricks in such a way that no point of the square is contained in more than 3 bricks, but if the bricks are sufficiently small at least 3 have a point in common. Similarly a cube in Euclidian n-space can be decomposed into arbitrarily small bricks so that not more than $n + 1$ of these bricks meet at a point. Lebesgue conjectured that $n + 1$ is a minimum leading to an intuitive definition of dimension.

Formally dimension can be inductively defined [11] as:

- The empty set and only the empty set has dimension -1.
- A space has dimension $\leq n$ ($n \geq 0$) at a point p if p has arbitrarily small neighborhoods whose boundaries have dimension $\leq n-1$.
- X has dimension $\leq n$, DIM $X \leq n$, if X has dimension $\leq n$ at each of its points.
- X has dimension n at a point p if it is true that X has dimension $\leq n$ at p and it is false that X has dimension $\leq n-1$ at p.
- X has dimension n if DIM $X \leq n$ is true and DIM $x \leq n-1$ is false.
- X has dimension = infinity if DIM $X \leq n$ is false for all n.

Hausdorff [12], for arbitrary metric spaces, was able to assign a p-dimensional measure for each non-negative real number p. This is a metric concept, while dimension is purely topological. There is a strong connection, for a space of dimension n must have positive n-dimensional measure. But if we consider not merely the metric space X but X together with all the metricies that can be put into it, or, the class of all spaces homeomorphic to X, then if all these spaces have positive n-dimensional measure, X itself must have dimension $\geq n$. Hausdorff's formal definition runs as:

The diameter of a set is defined as:

supremum $D(x,y)$, where x, y are all points in the set and D is the distance function.

Let X be a space and p an arbitrary real number such that $0 \leq p \leq \infty$, given eps > 0.

Let

$$m_p^{eps} = \inf \sum_{i=1}^{i=\infty} [D(a)_i]^p$$

where $X = a_1 + a_2 + \ldots$ is any decomposition of X in a countable number of subsets of diameter $<$ eps and the superscript p denote exponentiation.

Then

$$m_p(X) = \sup m_p^{eps}(X), \qquad eps > 0$$

and $m_p(X)$ is the Hausdorff-Bescovitch p-dimensional measure of X.

Hence, given an arbitrary metric space X, we denote this by the Hausdorff-Bescovitch dimension of X, the supremum of all real numbers p such that $m_p(X) > 0$, and it can be proved that Hausdorff DIM $X \geq$ DIM X.

EXAMPLE. Non-integral dimension:

Elements of the Cantor Middle Third set are defined as:

$$\sum_{n=1}^{\infty} \left(\frac{a_n}{3}\right)^n, \qquad \text{where } a_n = 0 \text{ or } 2.$$

The set is obtained by recursively dropping the middle third of a set and hence it has a Hausdorff-Bescovitch dimension of $\log 2 / \log 3 = 0.6309$.