# Structure and Expressiveness of Visual Computer Languages

*Clarisse Sieckenius de Souza*
*Departamento de Informática - PUC-Rio*
*Rio de Janeiro, RJ - Brazil*
*email: clarisse@inf.puc-rio.br*

**Monografias em Ciência da Computação — MCC06/94**

## Abstract

One of the meanings that can be assigned to the expression Visual Programming Language (VPL) is that of a programming language whose tokens are visual, rather than textual. Despite the linguistic parallel, however, VPL's can hardly challenge their textual counterparts in the construction of even very simple software programs. But Visual Command Languages (VCL's) seem to have nearly overtaken their textual counterparts as a protocol for user-system interaction. Thus, the very nature of software programs and interface commands appear to be differentially suited for visual representation.

In this paper, a classification of visual languages is proposed with respect to the dimensions of structural complexity and interactivity. Whereas languages deprived of a cognitively clear conceptual model in the semantic component seem to be necessarily low on the interactive side, languages deprived of formally clear structuring rules in the syntactic component do not. But, the lack of a powerful grammar seems to affect the language's ability to represent adequately the elements involved in even the simplest computer processes.

In order to integrate the adequacy of VCL's to a programming environment, an end-user VPL schema is presented, in which visual parametric and syntactic variations of a classical fractal image — the Koch Curve — are assigned perceptually motivated procedural meanings. Although images in this VPL schema are abstract, and not semiotically iconic or indexical as the majority of VCL images, underlying structural regularities seem to accommodate consistently some semantic notions related to data, control and modular abstractions.

A global linguistic and semiotic analysis of the proposed VPL schema suggests that there may be important expressive limits to the quality of genuinely visual programming languages. Nevertheless, the limits of successful visual command languages may apparently be extended to incorporate abstractions that are desirable for end-user programming applications.

## Keywords

Visual Programming Languages, Visual Grammars, Computer Semiotics, Iterated Function Systems, L-Systems

## 1. Visual Computer Languages

The meaning of the expression VISUAL LANGUAGE in the computer community covers a wide variety of languages, ranging from graphic user interface protocols to specialized programming languages that support visual applications [Chang'90]. In this paper, we will concentrate on those languages whose tokens are images or image components, as opposed to textual computer languages (TL's), whose tokens are alphabetic or numeric signs.

Many visual computer languages (VL's) do not have a formal grammatical description for the combination of signs into sentences or even texts. Their combinatorial rules are often based on spatial constraints suggested by analogies with the physical world or they are implicit in events and states allowed by Graphic User Interface (GUI) systems. In the first case, the interpretation rules applied to visual combinations of signs are projected from common sense knowledge VL users have. The assignment of meanings to visual representations, and conversely the assignment of visual form to intended meanings, is reputed to be cognitively intuitive. In the second case, interpretation rules are projected from a set of visual events and states supported by GUI's in a number of computer applications. Meaning and form assignments are decided based on VL users practice with GUI systems in general and the major cognitive resource is knowledge transfer from one application to another.

Nevertheless, there are other VL's whose visual sentences may accommodate formal descriptions. Interpretation rules are projected from such abstract domains, as computer science (e.g. algorithm visualization languages), logic (e.g. proof visualization languages), or geometry (e.g. fractal visualization languages). As these examples suggest, however, such languages are of a totally different nature. They belong more naturally to the field of scientific visualization, than to that of interactive protocols. At the interface level, they are characterized as output languages which can only be interpreted by highly specialized users.

One of the formalisms that can visualization languages is L-Systems [Lindemeyer'87]. Although this formalism has been originally proposed to characterize developmental models of cellular growth, it supports the description of images other than biological [Prusinkiewicz et al.'88]. For example, a classical fractal image — the Koch Curve — can be syntactically described by means of axiom, parameters, and production rules of a (0)L-System. However, the Koch Curve can also be described by parametric, rather than syntactic, formalism: Iterated Function Systems (IFS) [Barnsley'88]. Since the IFS method can be used to compress images in general, this approach may be taken as an alternative formalism to visual language description.

In this paper, VL's will be contrasted with each other along two major dimensions — structure and expressiveness. A general linguistic framework will be proposed to characterize different classes of VL's. Each class will be examined in terms of meaning assignment at input, output, and input-output levels. As a result, semiotic connections between a cognitively intuitive conceptual model and interactivity, on the one hand, and expressiveness and structural complexity, on the other, will suggest that genuine VISUAL PROGRAMMING LANGUAGES must incorporate features of interactivity and structural complexity to express important data-, control-, and module-level procedural abstractions [Appleby'91]. To illustrate this point, a VPL schema based on the Koch Curve image will be presented.

Section 2 describes the general linguistic framework for the analysis of VL's. Four classes of languages are identified and characterized in communicative terms. Section 3 introduces the Koch Curve VPL schema, providing parametric (IFS) and syntactic (L-Systems) approaches to formalize the language. Section 4 discusses the assignment of meaning to parametric and syntactic variations on the Koch Curve theme within a visual programming environment, and reports some of the directions suggested by a semiotic approach to the topic.

## 2. A Linguistic Framework for the Analysis of Visual Computer Languages

The general linguistic framework for the analysis of computer languages is represented in Figure 1. In it, a number of perceptible signs (lexical items) are combined according to established structuring rules (grammar) to mean or represent some entity (semantic model) for the computer (as input), the user (as output), or both (input-output). Typically, programming languages are designed to be interpreted primarily by computers, although they must obviously be intelligible humans as well. Command languages are also designed to be interpreted by computers, but they differ from programming languages in that they are not used to represent problem-solving procedures, but rather a set of imperatives for immediate action. Being command-level interface languages, their structural complexity is much smaller than that of programming languages, and some of the aspects of human cognition may be accounted for at the expense of computer performance (e.g. contrast the free order of parameters in some operating system languages with the rigid order of parameters in procedural calls of structured programming languages).

Output languages, in this context, are to be interpreted by humans, and so they must favor human cognition. However, they must also be processed by computers, in that output languages are generated by machines. It is unusual to have the same language as input and output precisely because human and computer interpretation mechanisms and resources differ so widely. In general, the interpreter's capacity is driving the design of the language: if the interpreter is the computer, human programmers adapt to rigid syntax and data types; if it is an end-user, the computer processes text, graphics, and sound, in order to optimize message understanding.

One exception to the above dichotomy between input and output languages is natural language interfaces, where computers are able to understand and generate human language texts. Another exception to having different input and output languages is direct manipulation of system objects [Schneiderman'83]. In this case, users directly operate upon output objects, which in turn become input objects. The most popular example of this paradigm is the XEROX Star interface [Smith et al.'82], which not by chance provides an example of a visual language. The connection between direct manipulation and visual languages is intrinsic, in that the basic metaphor in direct manipulation is that of a model world [Hutchins,Hollan & Norman'86].

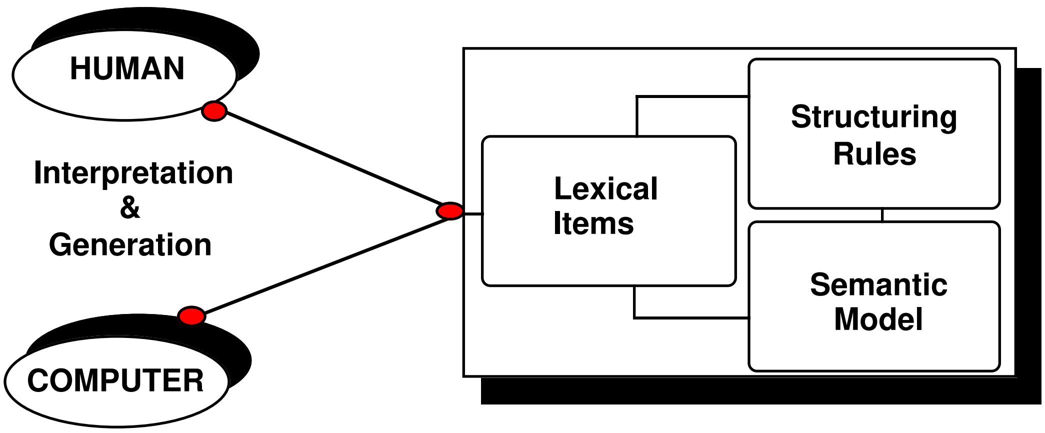


Figure 1: General Linguistic Framework

In order to illustrate the lexical, syntactic, and semantic components of VL's appearing in Figure 1, we can take an instance of the Macintosh OS interface language [Apple Computer'87], in which the images of a file (lexical item 1), a hard disk (lexical item 2) and a trash can (lexical item 3) appear on the desktop (lexical item 4). A visual grammar rule says that objects may be dragged and dropped onto other objects. The basic interpretation to this kind of spatial sentence is that the dropped object is moved into the limits of the recipient object (e.g. a file into the hard disk or into the trash). Both user and computer have a "common understanding" of the sentence, and for this to be possible the role played by the semantic component is crucial.

If we examine the meanings of the visual sentence FILE INTO DISK, we see that the semantic component of the visual language is actually reinterpreted by the computer in a cascade of symbolic levels [Winograd and Flores'86] until eventually some operating system action results (e.g. move or copy the values of given storage locations into other given storage locations). Thus, the semantic component actually includes that level of symbolic representation that shares the largest set of objects, relations, and actions, with the user's conceptual model of the task (e.g. put file into a new organization slot or reproduce file contents in a backup file).

The syntax of the example above is extremely simple, as that of most visual command languages (VCL's). Just as a textual programming language syntax is much more complex than a textual command language syntax, visual programming languages (VPL's) could be expected to offer users richer syntax and greater generality than visual command languages. However, unlike their textual counterparts, VPL's cannot count on natural language meaning associations to such notions as DATA TYPES (e.g. "string" and "real"), CONTROL STRUCTURES (e.g. "repeat [...] until" and "if [...] then [...] else [...]"), or MODULARITY (e.g. "procedure [...]" and "program [...]"). So, in order to design real general-purpose problem-solving visual representation languages, which must be intelligible to human programmers, some sort of conceptual model supporting the semantic component must be motivated by a visual equivalent of natural language words.

The contrast above suggests that the general linguistic framework proposed in Figure 1 may be differentially instantiated for four classes of visual languages. The ideal language where lexical, syntactic and semantic components are equally rich is one instance. The second instance is that exemplified by the Macintosh VCL, where the syntactic component is poor, but the semantic one is rich and motivated. The third instance is that of hypothetical VPL's, with rich syntax and poor motivation for a conceptual model. This is actually the case of fractal geometry visualization languages supported by L-Systems and IFS formalisms, as mentioned in Section 1. In Figure 2 are shown the configurations of the general linguistic framework for each of these examples. A fourth configuration is shown, where both the syntactic and the semantic components are relatively poor. This is a case of VL in which visual images are directly associated to computer actions. We envisage this situation as the equivalent to button pressing and depressing in control panels in order to trigger processing in the machine. If such buttons were associated to images (icons) with no connection to a consistent conceptual model, then the resulting "language" would be an instance of the fourth class.

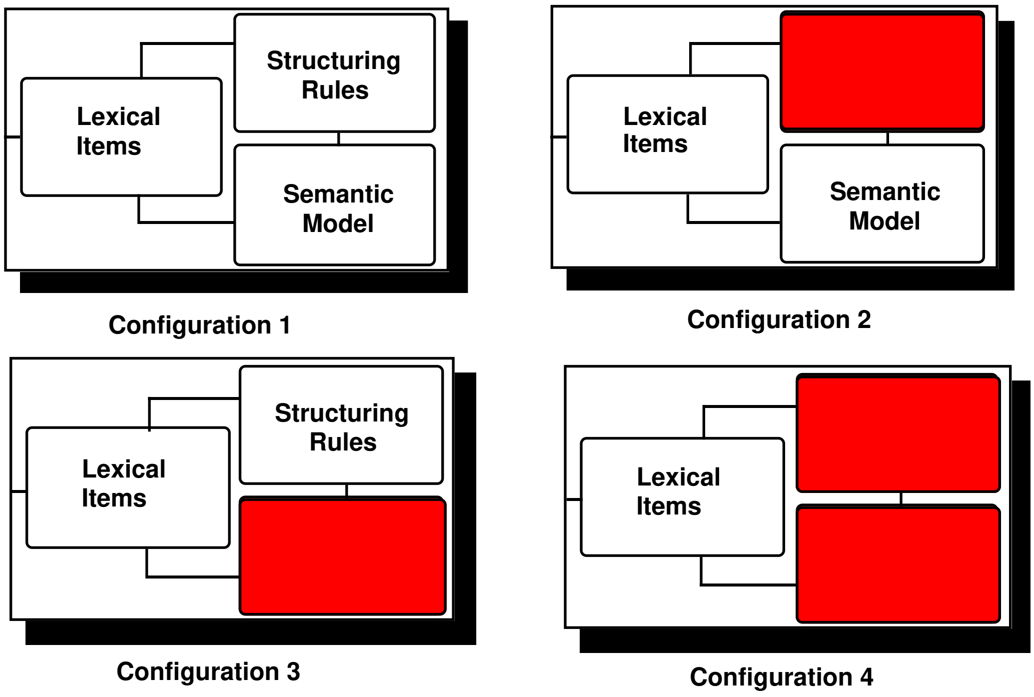


Figure 2: Different Configurations of VL Linguistic Framework

The consolidation of this approximate taxonomy of VL's reveals the situation pictured in Table 1. Configuration 1 languages, having a rich conceptual model associated to the semantic component, are high on the interactive side because users share meanings with some symbolic system the computer is prepared to interpret. Because such languages are also high in terms of structural complexity, they can be expected to accommodate notions of data, control, and module-level abstractions, being suited to represent problem-solving methods as well as immediate action imperatives. Configuration 2 languages also offer a good conceptual model as a basis for the semantic component, but they lack structural complexity to represent structures involved in programming. Configuration 3 languages are low in interactivity, because the underlying conceptual model in terms of which images must be interpreted are not cognitively motivated (as they are in the Macintosh example), but rather formally motivated (as they are in the algorithm, proof or fractal geometry visualization examples mentioned in Section 1). These languages are therefore high in structural complexity, because image token combinations are the direct consequence of syntactic operations upon computer layers of symbolic processing. Configuration 4 languages cannot be properly classified as languages, since there isn't the least regulation for the combination of visual tokens. They are only visual representatives of computer symbols, which humans cannot relate to any conceptual model, or analyze in terms of any set of structuring rules. For this reason we have called them "visual protocols".

| **Configuration** | **Structural Complexity** | **Degree of Interactivity** |
|---|---|---|
| 1 | **HIGH**<br>Input-Output Interface Language or Programming Language | **HIGH**<br>Input-Output Interface Language or Programming Language |
| 2 | **LOW**<br>Input-Output Interface Language | **HIGH**<br>Input-Output Interface Language |
| 3 | **HIGH**<br>Output Interface Language | **LOW**<br>Output Interface Language |
| 4 | **LOW**<br>Input Protocol | **LOW**<br>Input Protocol |

Table 1: Attributes of VL Structural Configurations

The theoretic background for our classification in terms of degree of interactivity is Norman and Draper's approach to User-Centered System Design [Norman & Draper'86]. One of the factors determining success in human-computer interaction is the users ability to formulate reasonable intentions of action upon the backend system. In domain-specific applications, for instance, the formulation of intentions is driven not only by the problem situation encountered in the real world, but also by the conceptual model expressed by interface signs. The authors show that the absence of such conceptual model severely challenges the success of the interaction as a whole, since all steps in input specification and output evaluation are interdependent. As for the theoretic background for our classification in terms of structural complexity, we resort to Chomsky's classical characterization of formal languages [Chomsky'63] into a hierarchy of increasing complexity. Although this characterization requires adaptations when applied to visual languages, the examples we will be dealing with in the next section involve the classical notions of syntactic rewriting rules.

## 3. The Koch Curve Visual Programming Schema

The Koch Curve is a classical fractal image constructed from a line segment (the initiator), which is then divided into three equal parts. The middle third is replaced by an equilateral triangle and the base is taken away. The resulting image (the generator) repeatedly appears in every line segment of the curve as the process is iterated $\underline{n}$ times. When $\underline{n}$ tends to infinity, the image contains no straight lines, but the curve does not admit a tangent in any of its points [Peitgen et al'92]. Two methods can be used to generate the Koch Curve: Iterated Function Systems (IFS) and L-Systems.

In the IFS method [Barnsley'88], iterated affine contraction transformations determine a unique final image, known as the IFS attractor. Each transformation present in the system has specific parameters relative to primitive transformations (rotation, scaling, and translation), so that different images are obtained if parameters are changed. Thus, the Koch Curve in this method is generated by a set of functions $w_i$ that apply to a point (x, y) and determine a second point (ax+by+e, cx+dy+f), according to the following schema:

| | | |
|---|---|---|
| a= sx cosθ | b = -sy sinθ | e = E tx |
| c= sx sinθ | d= sy cosθ | f = E ty |

θ is the rotation angle relative to x, counterclockwise.
sx and sy are scaling factors relative to x and y, respectively.
tx and ty are the translations along x and y, respectively.
E is an external drawing factor.

The Koch Curve is generated by IFS(λ) = $\{w_i;$ i = 1,2,3,4}, with the following set of parameters, for:
s1 = 1/3 cos(λπ/3),  s2 = 1/3 sin(λπ/3),  λ = 1

| **i** | **a** | **b** | **c** | **d** | **e** | **f** |
|---|---|---|---|---|---|---|
| 1 | 2s1 | 0 | 0 | 2s1 | 0 | 0 |
| 2 | s1 | -s2 | s2 | s1 | 2s1 | 0 |
| 3 | s1 | s2 | -s2 | s1 | 3s1 | s2 |
| 4 | 2s1 | 0 | 0 | 2s1 | 4s1 | 0 |

The alternative generation method uses L-Systems. They have been introduced by the application of grammars to the study of growth in cellular organisms [Lindemeyer'87]. This family of grammars differs from Chomskyan grammars because there is no difference between terminals and non-terminals, and because at each step in a derivation, a production is applied to each symbol of a sentential form (and not just a substring or short symbol) [Hopcroft & Ullman'79].

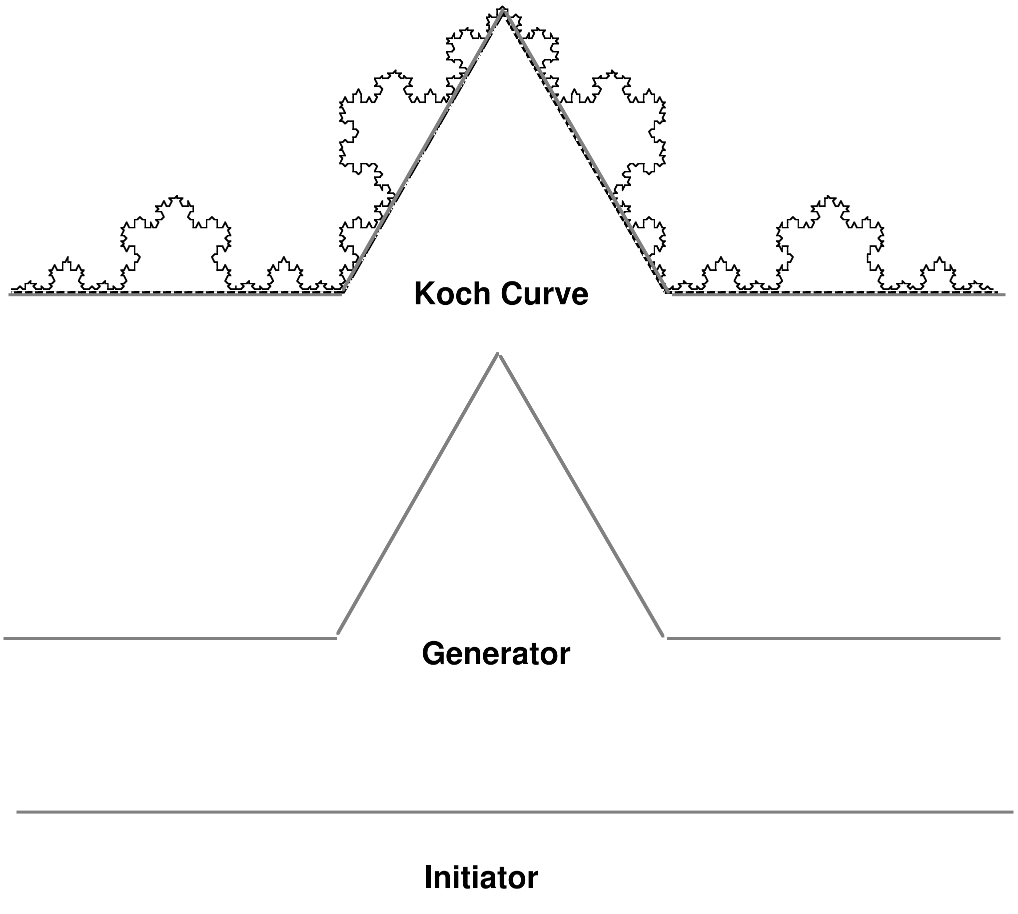


Figure 3: The Koch Curve

The graphic interpretation of L-Systems associates symbols in the grammar to the drawing of line segments based on Seymor Papert's turtle graphics [Papert'80, Peitgen et al.'88]:

**F** is a forward move in which a fixed-length line segment is drawn;
**f** is a forward move, but without drawing the same fixed-length line segment
**+** is a left turn (counterclockwise) by a fixed angle θ
**-** is a right turn (clockwise) by a fixed angle θ
**[** pushes current drawing state onto stack
**]** pops current drawing state from stack

Extensions to this basic set of structure-related symbols, appearing in computer graphic systems such as FRACTINT [FRACTINT] and L-Systems [Bourke'90], may include arbitrary parameter-related symbols to:

increments/decrements of the line length by a fixed factor
increments/decrements of the turning angle by a fixed factor

L-Systems consist of an axiom (the initial string) and a set of production rules. Context-free production rules define (0)L-Systems (for 0 context), whereas context-sensitive production rules define for example (1,1)L-Systems (for 1 context string left and 1 context string right of the symbol to be rewritten). The Koch Curve is defined by the following (0)L-System [Peitgen' et al.92]:

axiom : F
productions: F —> F+F--F+F; - —> -; + —> +
turning angle parameter: 60 degrees

Our goal is to explore some **perceptive** features associated to parametric and syntactic variations on the original Koch Curve image, in order to identify structural elements that could be incorporated to a conceptual model of a Visual Programming Language. We will try to make changes to the Koch Curve VL, which has been classified as a Configuration 3 language, so that it approximates the ideal of a Configuration 1 language. And because domain-specific conceptual models are contrary to the generality desired in programming languages, we will resort to perceptually-motivated conceptual models, and see to what extent perception can contribute to facilitate meaning assignment in visual programming languages.

Visual grammars [Lakin'87,Chang'90,Crimi et al.'90] emphasize the notion of spatial parsing. It means that the location of images can be assigned different meanings. The Koch Curve VPL will have to account for meanings related to problem-solving methods, and the first issue is to see if different locations <u>of</u> or <u>on</u> the image can be associated to problem-solving steps. As shown in Figure 3, there is a perceptual invariant at all stages of the curve generation — the location of the two extremes of the initiator, generator, and the Koch Curve itself. Since processes evolve along time, the transposition of a time scale onto a space scale allows us to assign two distinct meanings to the extremes of the image: the BEGINNING and the END of the problem-solving process. Thus, the process is represented as a varied PATH from BEGINNING to END, just like in structured textual programming languages (TPL's) the process is represented by a unit (<u>program</u>) that extends across procedures, from beginning to end.

If we choose PASCAL as a paradigmatic TPL, there is another meaning association that is perceptually motivated by the Koch Curve image. In PASCAL, programs can be composed of procedures, in a recursive block structure where each subpart has a beginning and an end. Two notions emerge here: the one of PARTS (or STEPS) within a whole program, and the one of STRUCTURAL SYMMETRY (or recursive block structure for programs and procedures). Again in Figure 3, we notice that the generator of the Koch Curve clearly introduces the perception of SEGMENTS (or parts, or steps) within the path leading from beginning to end of the curve. Moreover, as iterations apply, the structure of the generator is reproduced (by definition) in each of the preceding generator segments. Thus, there is a perceptual motivation for the Koch Curve VPL to associate the meaning of PROCEDURE to each segment of the generator, the meaning of SEQUENCE to the spatial increment of points along the **x** coordinate, and the meaning of RECURSIVE ITERATION to the transformation of each segment into a Koch Curve image by itself.

Therefore, regardless of any variations applied to the Koch Curve generation process in either methods, there is a perceptual motivation to the association of meanings proposed in Table 2.

| | |
|---|---|
| INITIATOR | PROGRAM |
| INITIATOR'S ORIGIN | BEGIN PROGRAM |
| INITIATOR'S END | END PROGRAM |
| GENERATOR | SEQUENCE OF PROCEDURES IN THE PROGRAM |
| GENERATOR'S LINE SEGMENTS | PROCEDURES |
| LINE SEGMENT'S ORIGINS | BEGIN PROCEDURE |
| LINE SEGMENT'S END | END PROCEDURE |
| ITERATION IN GENERATION | RECURSIVE ITERATION OF PROCEDURES |

Table 2: Perceptually-motivated association of procedural meanings to the Koch Curve image.

Associations presented in Table 2 allow us to build an interpretive grid for the parametric and syntactic variations that will be applied to the original Koch Curve image. As seen in Figure 4, the grid identifies 4 distinctive regions in the Koch Curve, constraining our VPL to problem-solving methods composed of 4 intermediary steps.

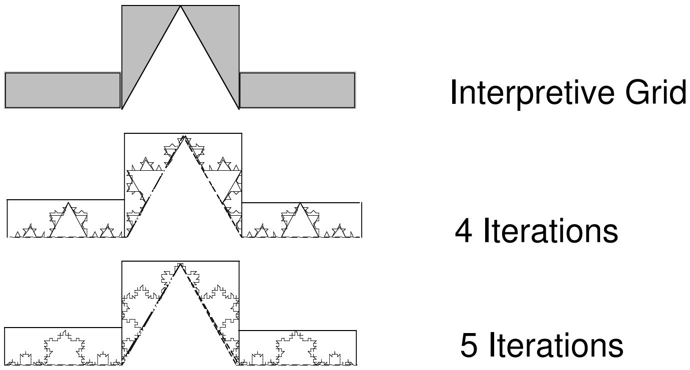


Figure 4: Koch Curve VPL Interpretive Grid

In the following, parametric and/or syntactic variations on the generation methods of the Koch Curve will show that perceptions will vary considerably, and so will the possibilities of procedural meaning assignment to the resulting images.

### *Parametric Variations*

The FRACTINT [FRACTINT] program allows users to specify parameters for the generation of fractal images using the IFS method. Probabilities are assigned to each function, so that the IFS parameters presented in this section assume the following input values:

| a | b | c | d | e | f | probability |
|---|---|---|---|---|---|---|
| .3333319 | 0 | 0 | .3333319 | 0 | 0 | 0.25 |
| .166666 | -.2886755 | .2886755 | .166666 | .3333319 | 0 | 0.25 |
| .166666 | .2886755 | -.2886755 | .166666 | .4999979 | .2886755 | 0.25 |
| .3333319 | 0 | 0 | .3333319 | .6666639 | 0 | 0.25 |

Figure 5 shows arbitrary parameter variations performed on (a), (c), and (a) and (c), so that the values assumed for the generation are, respectively:

| | | | | | | |
|---|---|---|---|---|---|---|
| (I) | 2(a), | b, | c, | d, | e, | f |
| (II) | a, | b, | 2(c), | d, | e, | f |
| (III) | 2(a), | b, | 2(c), | d, | e, | f |

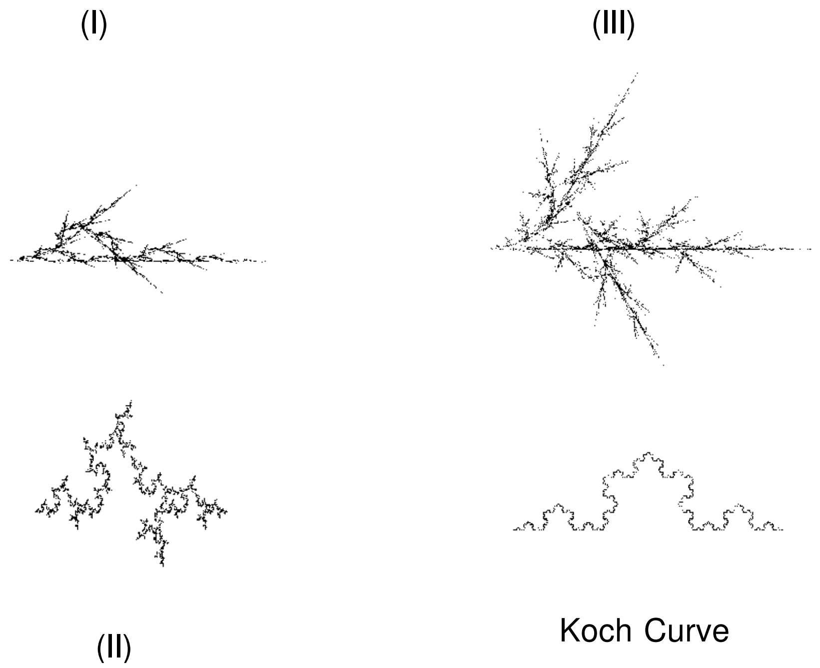


Figure 5: Arbitrary Variations of IFS Parameters

In terms of the interpretive grid proposed in Figure 4, the new images generated by parametric variations suggested above do not motivate any direct association. So, instead of working locally on each of the functions, the other kind of parametric variation we can perform is centered on the $\lambda$ factor. We know that the Koch Curve is generated by the IFS stated above when the value of $\lambda$ is 1. Stepwise variations of this factor within a [1.0] interval generate images that lose some of the characteristics of the Koch Curve semantics we are trying to build. Examples appearing in Figure 6 (for 2 iterations only) show that the "continuity" of the path leading from beginning to end of the process is lost. This perception disturbs a model in which different continuous segments of a polygonal line can stand for procedures in a program.

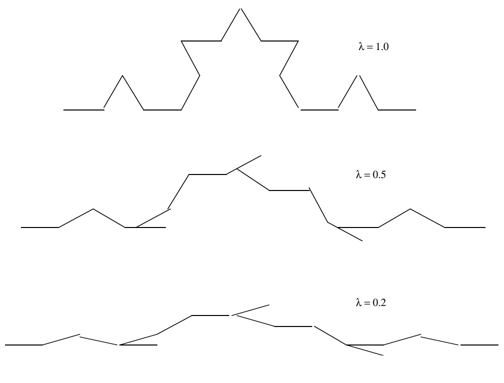


Figure 6: Stepwise Variations of IFS Value

However, the very fact that such disturbance occurs and is visually perceived points in the direction that $\lambda$ may be assigned meaning, if some perceptual regularity is found. An observation of images generated by a decimal point decrement of λ between 1 and 0 reveals that as λ decreases, the image is "bent forward", trespassing the limits of the interpretive grid. An interesting special condition of λ values is the interaction it presents with the number of iterations in the image generation process. The perception of "discontinuity" is only obtained from the second iteration onwards (the first iteration renders the image generator, which is continuous). This feature shows that there is a special case of λ values that can still be interpreted within a procedural framework — those associated to the image generator (i.e. for the first iteration only).

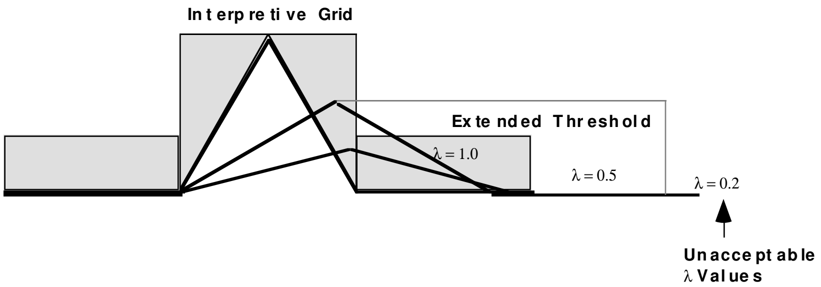


Figure 7: Regions of Critical λ Values on the Interpretive Grid

Therefore, we propose that this special condition be used in the semantic component of our Koch Curve VPL to account for a SELECTION structure. IF the variation is performed within a given range THEN the program path is of one type, ELSE it is of another. To illustrate the point, suppose that the interpretive grid is extended to show two limits: the original one and another threshold marking, as shown in Figure 7. When the values of λ are varied within [1,0.5] the image generator stays within the new grid markings. As λ falls below 0.5, the end of the polygonal line trespasses the acceptable limits. The stepwise decrement we are considering for λ actually opens 10 slots of "meaning" for SELECTION structures, which suggests that a control structure like CASE can also be accommodated. Additionally, all other "discontinuous" images obtained when the number of iterations is greater than 2 can be assigned meanings for ERROR HANDLING routines (since the process path is disrupted due to improper value assignment).

One last observation about parametric variations based on λ values is that in the IFS method, such variations have fixed SCOPING. They are all global, since as values are changed in the parameter matrix, the whole image is affected. So, in sum, Table 2 above can be extended to include λ variation-based meanings just presented.

| INITIATOR | PROGRAM |
|---|---|
| INITIATOR'S ORIGIN | BEGIN PROGRAM |
| INITIATOR'S END | END PROGRAM |
| GENERATOR | SEQUENCE OF PROCEDURES IN THE PROGRAM |
| GENERATOR'S LINE SEGMENTS | PROCEDURES |
| LINE SEGMENT'S ORIGINS | BEGIN PROCEDURE |
| LINE SEGMENT'S END | END PROCEDURE |
| ITERATION IN GENERATION | RECURSIVE ITERATION OF PROCEDURES |
| PARAMETER-BASED<br>IFS FUNCTION DEFINITION | GLOBAL VARIABLE SCOPING |
| GLOBAL PARAMETER VARIATION | SELECTION STRUCTURES & ERROR HANDLING |

Table 3: Perceptually-motivated Semantics of Parametric Image Generation Methods

### *Syntactic Variations*

The L-SYSTEMS program [Bourke'90] allows users to write (0)L-Systems (context-free), (1,1)L-Systems (context-sensitive) and Stochastic production rules to generate images. The Koch Curve in Figure 3, as mentioned before, is generated by the following (0)L System:

axiom : F
productions: F —> F+F--F+F; - —> -; + —> +
turning angle parameter: 60 degrees

An immediate formal observation is the convenience of the syntactic representation in which the initiator is elegantly the axiomatic line segment (F), and the generator appears as a continuous polygonal line composed of 4 line segments (F —> F+F--F+F). However, these line segments have in theory the same length as the initiator, which means that an automatic scaling factor must apply in computer graphics applications (e.g. L-SYSTEMS, FRACTINT) so that the drawing scale is recomputed relatively to screen dimensions after each iteration in the process. In what follows, we will take this resizing operation for granted and images will be compared against a fixed 2D grid.

The right-side of L-Systems rewriting rules determine the drawing of a number of line segments. According to Turtle Graphics principles, line segments may be actually drawn ("pen down") or just traversed without any drawing ("pen up"). The drawing can be modified by angle and/or line length increments, as well as by stack controls for the initial coordinates of the segment. Consequently, syntactic variations on the original Koch Curve (0)L-System grammar can be of any kind (just like arbitrary parametric variations observed in the IFS method). The issue is to find some type or types of syntactic variations that can lead to perceptually-motivated procedural meanings against the Koch Curve VPL interpretive grid.

The basic semantic associations presented in Table 2 are maintained in the L-Systems generation method. It should be noticed that the syntactic representation allows for an additional association between the number of **F** constituents in the generator production rule and the number of procedures along the program path. Interesting variations are those that, similarly to the Koch Curve generation process, preserve the perception of a segmented "path". In particular images in Figure 8 (a) and (b) characterize perceptions that can be associated to procedural steps within procedures (e.g. command statements within a procedure) in a linear sequence (a) or in a loop structure (b).

Image (a) motivates the notion that the first procedure in the program is composed of 5 steps (possibly 5 STATEMENTS) in linear sequence, whereas image (b) motivates the notion that the procedure is composed of 4 (LOOPING) steps. The difference between this sort of iteration and the ones previously mentioned in parametric variations (associated to the number of iterations in the generative process of the curve) is that in image (b) the LOOP is not associated to recursion. As the arrows suggest, there is a decision determining the end of the iteration along the sides of the square. The relevant point in this syntactic variation is that, unlike what has been obtained with $\lambda$ value manipulations, self-similarity is not mandatory. In other words, syntactic variations allow for LOCAL SCOPING of both data and control structures. The locality of transformations in syntax-driven variations is precisely the mechanism by which line segments can be interpreted alternatively as PROCEDURES or STATEMENTS. In global (parametric) variations, as images reproduce themselves on each line segment of the image, once assigned the meaning of "procedure" at one level of iteration, it sounds inconsistent to assign the meaning of "statement" to the same (resized) image at the next level of iteration. But, when self-similarity is suspended, then different line segment shapes may be consistently assigned different meanings.

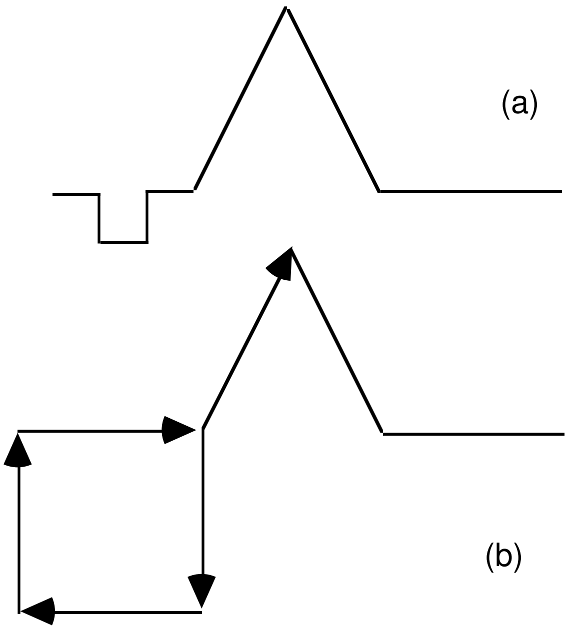


Figure 8: Syntactic Variations in L-Systems Production Rules

Additionally, this very principle broadens the expressiveness of the VPL, in that there may be a number of visual representations for a procedural path. For example, programs can be shown to lead to different endings, as input data are varied. Figure 9 illustrates one of such situations. Before the first procedural step is performed, a decision structure dispatches execution along the upper (original Koch Curve) path or along the lower (arbitrary) path. On the left side, both paths have an equal number of steps; but, on the right side, the alternative (arbitrary) path has a smaller number of steps. As can be seen, a syntactic approach to the Koch Curve VPL produces an underlying graph grammar schema, commonly used to visualize computer processes [Shu'88].

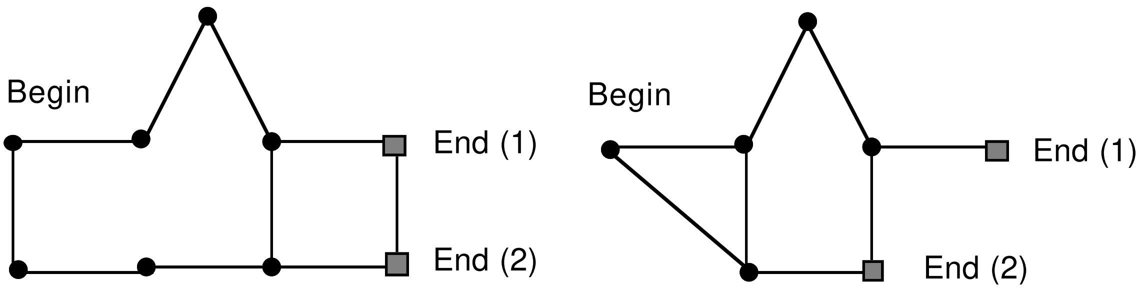


Figure 9: Different Procedural Paths

Finally, the perception of disruptions along the path, previously associated to error handling conditions, can be equally obtained in L-System representations. Just for illustration, consider the following (1,1)L-System:

axiom:

F+[>F+F--F+F]f--F+F (Locality of transformations marked by "[ ]")

context-sensitive production rule:

$\geq$ F $\pm$ -> [>F+F--F+F]f (The size of F continually decreases with iterations)

turning angle:

60 degrees

line length increment:

0.333

Since global variations can also apply to L-Systems angular parameters (60 degrees may be decreased by a fixed factor), we conclude that the expressiveness of the above suggested syntactic variations can be summarized as shown in Table 4.

| | | |
|---|---|---|
| INITIATOR | IFS & L-SYSTEMS | PROGRAM |
| INITIATOR'S ORIGIN | IFS & L-SYSTEMS | BEGIN PROGRAM |
| INITIATOR'S END | IFS & L-SYSTEMS | END PROGRAM |
| GENERATOR | IFS & L-SYSTEMS | SEQUENCE OF PROCEDURES IN THE PROGRAM |
| GENERATOR'S LINE SEGMENTS | IFS & L-SYSTEMS | PROCEDURES |
| LINE SEGMENT'S ORIGINS | IFS & L-SYSTEMS | BEGIN PROCEDURE |
| LINE SEGMENT'S END | IFS & L-SYSTEMS | END PROCEDURE |
| ITERATION IN GENERATION | IFS & L-SYSTEMS | RECURSIVE ITERATION OF PROCEDURES |
| PARAMETER-BASED IFS FUNCTION DEFINITION | IFS & L-SYSTEMS | GLOBAL VARIABLE SCOPING |
| GLOBAL PARAMETER VARIATION | IFS & L-SYSTEMS | SELECTION STRUCTURES & ERROR HANDLING |
| SYNTAX-BASED DEFINITIONS FOR GENERATION | L-SYSTEMS | LOCAL VARIABLE SCOPING<br>STATEMENT VS. PROCEDURE DISTINCTIONS<br>NON-RECURSIVE LOOPS<br>ALTERNATIVE PROGRAM PATHS |

Table 4: Method-dependent expressiveness in the Koch Curve VPL

| | | |
|---|---|---|
| INITIATOR | parametric or syntactic | PROGRAM |
| INITIATOR'S ORIGIN | parametric or syntactic | BEGIN PROGRAM |
| INITIATOR'S END | parametric or syntactic | END PROGRAM |
| GENERATOR | parametric or syntactic | SEQUENCE OF PROCEDURES IN THE PROGRAM |
| GENERATOR'S LINE SEGMENTS | parametric or syntactic | PROCEDURES |
| LINE SEGMENT'S ORIGINS | parametric or syntactic | BEGIN PROCEDURE |
| LINE SEGMENT'S END | parametric or syntactic | END PROCEDURE |
| ITERATION IN GENERATION | parametric | RECURSIVE ITERATION OF PROCEDURES |
| PARAMETER-BASED IFS FUNCTION DEFINITION | parametric | GLOBAL VARIABLE SCOPING |
| GLOBAL PARAMETER VARIATION | parametric | SELECTION STRUCTURES & ERROR HANDLING |
| SYNTAX-BASED DEFINITIONS FOR GENERATION | syntactic | LOCAL VARIABLE SCOPING<br>STATEMENT VS. PROCEDURE DISTINCTIONS<br>NON-RECURSIVE LOOPS<br>ALTERNATIVE PROGRAM PATHS |

Table 5: Parametric and syntactic variation expressiveness in the Koch Curve VPL

We are suggesting that L-Systems are a more powerful formalism to characterize the syntax of the Koch Curve VPL schema we have just proposed. This could be expected from the very fact that "parametric grammars" are actually non-standard structuring systems, once the classical **rewriting rule** principle is formally absent However, the important issue is to contrast the expressiveness of parameters to that of syntactic structures as far as our basic image is concerned. Table 5 provides relevant indications for the ideas discussed in the following section.

## 4. Linguistic and Semiotic Analysis of the Koch Curve VPL Schema

The major issue in this approach is to assess the complexity of (i) assigning meanings to perceived visual representations and (ii) assigning visual representations to intended procedural meanings. In a visual programming environment, by analogy with textual ones, programmers (VPL users) must be able to (i) read visual programs and (ii) write visual programs.

The first characteristic of programs is that they constitute **texts**, in linguistic terms. They must have a cohesive and consistent internal structure, organizing data, controls and modules into a representation of methods for solving a certain kinds of problems. Consequently, VL's that do not have powerful structuring mechanisms are poor candidates for programming applications.

As mentioned in Section 1, and more extensively discussed in previous work [de Souza'93], analogies with real world physical rules often play the role of a "proviso grammar" in visual languages. However, such analogies strongly constrain the kinds of computational operations a visually specified program can contain [Halazs & Moran'82]. For example, the trash can icon on the Macintosh Desktop interface is semantically ambiguous, if we think of the effects of dropping file icons into it and compare to dropping disk icons. Such ambiguity would be exponentially harmful in a full-fledged VPL based on the desktop metaphor. Moreover, one should question how far this metaphor can be extended when it comes to programming structures like the ones presented in Section 3. Other metaphors could be found, but the association of physical reality phenomena to abstract computational models do not seem to guarantee generality of purposes.

We have claimed that our VPL schema accounts for some interesting data-, control-, and module-level abstractions. To illustrate the difficulty in performing meaning assignment tasks (i) and (ii), consider a program schema which does the following:

```
Main program
        reads input value for Global Variable GV1
        reads input value for Global Variable GV2
        evaluates if global condition GC is true
        selects the case of GV1 values:
                in case of GV1 = val1: repeats GV2 calls to procedure1 passing parameter GV2
                in case of GV1 = {val2,val3,val4,val5}:
                        if GC is true
                        calls procedure1 passing parameter GV2 with specific value v
                        else
                        calls procedure1.5
                        call procedure1.6
                in all other cases of GV1 values
                        sends an error message

Procedure1
        reads received GV2 value
        reads global GV1 value
        calls procedure1.1 passing parameters GV2 and GV1
        call procedure1.2 passing parameters  GV2
        call procedure1.3 passing parameters  GV2
        call procedure1.4 passing parameters  GV2
```

Procedure1.1
reads received GV2 value
reads received GV1 value
if GV1 = val5 executes instruction 1.1.1
else repeats execution 1.1.2 n times

Procedure1.2
reads received GV2 value
executes instruction 1.2.1

Procedure1.3
reads received GV2 value
executes instruction 1.3.1

Procedure1.4
reads received GV2 value
executes instructions1.4.1

The program schema above is sufficiently generic to support the claim that a VL accounting for this situation deserves the qualification of a Visual Programming Language, as opposed to a Visual Command Language. Also, all structures involved in this program's code in PASCAL, for instance, appear in Table 5, Section 3, which means that our Koch Curve VPL schema should support this kind of program coding with visual representations.

The difficulty in assigning meanings to visual representations of the programs content can be assessed by the difficulty in making the associations shown below, from left to right of the page. The difficulty in assigning visual representations to intended program data, control and structures can be assessed by the difficulty in making the associations from right to left. Apparently, if a programmer had in mind the above program schema, **it is unlikely** that this person would express his or her conceptions with the visualizations proposed in Section 3. This means that associations are not natural, even though they are perceptually motivated.

| | | | |
|---|---|---|---|
| different values of GV1 | different values of $\lambda$ | different regions in the interpretive grid | see Figure 7 |
| different values of GV2 | different # of iterations | different generative stages of the Koch Curve | see Figure 4 |
| selections involving condition GC | different production rules | different paths along the visible graph | see Figures 8 and 9 |
| procedures | $\lambda$ values or F constituents in L-Systems rules | line segments in Koch Curve generator | see Figure 4 |
| instructions | F constituents in L-Systems rules | line segments outside the Koch Curve generator | see Figure 8 |

The next step in assessing the expressive quality of the Koch Curve VPL is to find out the difficulties involved in **learning and memorizing** the underlying associations between visual forms and program structures in the language. This is the point where perceptual motivation plays a leading role and provides distinctions between higher (motivated) and lower (unmotivated) quality VPL candidates. In order to gauge the extent of motivation's impact on the quality of VL's, consider a hypothetical language in which the notion of instruction sequencing had no perceptual (spatial) parallel and progressions in execution were signaled by a color hue gradation. The switch from spatial parsing paradigm into a chromatic paradigm to account for the semantics of sequential programming is clearly inadequate.

The considerations above leave no doubt that representing generic programs in VL's is an extremely complex (although not totally impossible) task. This finds theoretic support in Eco's Theory of Sign Production [Eco'76], which states that choosing non-established expression types for intended contents is a **semiotically difficult** process (*ratio difficilis*, in Eco's terms). Additionally, still abiding to Eco's theory, programs evidently need highly articulated expressions to convey highly articulated meanings. Since VL's are seldom articulated, the inappropriateness of this code for the task is considerable. Besides, the articulated systems we have chosen to support our VPL schema are not native to program codification (but to fractal image generation, image compression, cellular growth modeling, and the like), increasing the **cognitive labor** needed to produce signs [de Souza'93].

A comparison with successful visual command languages illuminates the fact that iconic and indexical images facilitate the interpretation and assignment of meaning. Because the images in our VPL schema are not iconic or indexical in the usual sense, the arbitrariness of signs must be overcome by an effective learning process. The same is true for textual programming languages; but the obvious semantic relations between tokens in many TPL's and English Language words do function as some sort of index to the appropriate meaning of structures in the programming language.

A contrast between syntactic and parametric variations on abstract images, as the ones exemplified in Section 3, suggests that the former have a greater expressive power than the latter for programming purposes. Without the L-Systems approach to the Koch Curve generation, extremely important phenomena such as local variable scoping, statement vs. procedure distinctions, non-recursive loops, and alternative path representations would be left outside the VPL schema. This is an important indication for the expressiveness of merely parametric transformations on generic images. Size, color, texture, location, and other attributes are often changed (to represent distinguished meanings) on the basis of parameter values. However, a homogeneous motivation for such parametric variation of image attributes across the whole spectrum of programming structures may become exceedingly difficult to maintain, and the lack of perceptual motivation in meaning associations may imply that the only semiotic design principle of a VPL is lost.

Finally, it could be argued that the IFS method for the Koch Curve image generation has been overtaken by the power of the alternative L-System method, especially because the latter supports a general graph grammars to represent programs in a visual language. In part, the argument is valid, in that L-Systems also include parameters that can be varied in the production rules. However, one point in favor of the IFS method is that the very nature of the method provides a more consistent notion of GLOBAL VARIABLE VALUES than L-Systems (in which production rules can be designed to neutralize effects of such values at local levels). Another point of a more prospective flavor is that the IFS method can be applied to compress images of any kind [Barnsley'88]. Even if radical VPL's as the one sketched here seem far away from semiotic adequacy (though maybe not so in strictly linguistic terms), one of the possibilities we have in the future is to incorporate grammatical structuring to iconic or indexical VCL's, so that they are extended to include some programming structures. In this case, certain icons or indices could undergo principled parametric variations based on the IFS compress/decompress operations. Of course, it is too soon to evaluate the usefulness of such approach, but the possibility remains that it could serve expressive goals for end-user VPL's.

As a conclusion, we can say that Configuration 1 languages for programming purposes still have a long way to go. Nevertheless, for interactive purposes, at immediate command level, the way is open to explore perceptually -motivated small-scale rule-governed variations on images. Semiotic adequacy could be found in a combination of icons or indices and abstract images, within a global framework. As Eco proposes, icons are like achieved texts that must be taken as an encapsulated whole. The icon of the hard disk on the Macintosh desktop, for example, cannot be analyzed into parts. Its components are depicted via **another** visualization — a window with new icons on it. And, as this example shows, there are levels of encapsulation in visual command languages that are perfectly appropriate for the task they are meant to support. The question is one of identifying encapsulation levels in visual programming languages, where cognitively (and not just perceptually) motivated images could be used to help programmers make sense out of what they see, or make images out of what they sense.

## Acknowledgments

Images presented in this paper have been generated by shareware software like FRACTINT and L-SYSTEM [Bourke'90], and by a special program written in Visual Basic. Marisa Ortegoza da Cunha has provided the program's specification and Christiano Braga has helped with the code.

## References

Apple Computer'87 — Apple Computer Inc. (1987) *Human Interface Guidelines: The Apple Desktop Interface*. Reading, Ma. Addison-Wesley.

Appleby'91 — Appleby,D. (1991) *Programming Languages*. New York, NY. McGraw-Hill Inc.

Barnsley'88 — Barnsley,M. (1988) *Fractals Everywhere*. New York, NY. Academic Press.

Bourke'90 — Bourke,P. (1990) *L System*. Program and User Notes, available in Internet via Public FTP.

Chang'90 — Chang,S.K. (1990) *Visual Languages and Visual Programming*. New York, NY. Plenum Press.

Chomsky'63 — Formal Properties of Grammars. *Handbook of Mathematical Psychology* Vol. 2. New York, NY. John Wiley and Sons.

Crimi et al.'90 — Crimi,C., Guercio,A., Pacini,G., Tortora,G., and Tucci,M. (1990) Automating Visual Language Generation. *IEEE Transactions on Software Engineering*, **16**(10). pp.1122-1135

de Souza'93 — de Souza,C.S. (1993) The Semiotic Engineering of User Interface Languages. *International Journal of Man-Machine Studies* (1993) **39**, 753-773.

Eco'76 — Eco,U. (1976) *A Theory of Semiotics*. Bloomington, In. Indiana University Press.

FRACTINT — *FRACTINT*. Program by FRACTINT, available in Internet via Public FTP.

Halazs & Moran'82 — Halazs,F. and Moran,T.P. (1982) Analogy Considered Harmful. In T.P. Moran (ed.) *Eight Short Papers in User Psychology*. Palo Alto, Ca. Xerox Corporation. pp. 33—36

Hopcroft & Ullman'79 — Hopcroft,J.E. and Ullman,J.D. (1979) *Introduction to Automata Theory, Languages and Computation*. Reading, Ma. Addison-Wesley.

Hutchins, Hollan and Norman'86 — Hutchins,E.I., Hollan,J.D. and Norman,D.A. (1986) Direct Manipulation Interfaces. In Norman,D.A. and Draper,S.W. (eds.) *User Centered System Design*. Hillsdale, NJ. Lawrence Erlbaum. pp.87-124

Lakin'87 — Lakin,R. (1987) Visual Grammars for Visual Languages. *Proceedings of AAAI'87* . pp. 683-688

Lindemeyer'87 — Lindemeyer,A. (1987) Models for Multicellular Development: Characterization, Inference and Complexity of L-systems. In A. Kelmenová and J. Kelmen (eds.): *Trends, Techniques and Problems in Theoretical Computer Science*. Lecture Notes in Computer Science 291, Springer-Verlag, Berlin. pp. 27-40

Norman & Draper'86 — Norman,D.A. and Draper,S.W. (1986) *User Centered System Design*. Hillsdale, NJ. Lawrence Erlbaum.

Papert'80 — Papert,S. (1980) *Mindstorms: Children, Computers, and Powerful Ideas*. New York, NY. Basic Books, Inc.

Peitgen et al'92 — Peitgen,H.O., Jürgens,H. and Saupe,D. (1992) *Chaos and Fractals: New Frontiers of Science*. Berlin. Springer-Verlag.

Prusinkiewicz et al.'88 — Prusinkiewicz,P., Lindemeyer,A. and Hanan,J. (1988) Developmental Models of Herbaceous Plants for Computer Imagery Purposes. *Computer Graphics* **22**(4). pp. 141-150

Schneiderman'83 — Schneiderman,B. (1983) Direct Manipulation: A Step beyond Programming Languages. *IEEE Computer,* **16** (8). pp. 57-69

Shu'88 — Shu,N. (1988) *Visual Programming*. New York, NY. van Nostrand Reinhold.

Smith et al.'82 — Smith,D.C.,Irby,C.,Kimball,R.,Verplank,W., and Harslem,E. (1982) Designing the Star User Interface. *Byte* **7**, pp. 242-282

Winograd and Flores'86 — Winograd,T. and Flores,F. (1986) *Understanding Computers and Cognition: New Foundations for Design*. Norwood, NJ. Ablex.