Graph Grammar Algebraic Approach For Generating Fractal Patterns

H. K. Hussein

A. E. Hassanien

M. Nakajima

Graduate School of Information Science & Engineering
Tokyo Institute of Technology
2-12-1 0-okayama, Meguro-Ku
152-8552 Japan
{huss,abo,nakajima}@cs.titech.ac.jp

ABSTRACT

This paper presents a new approach to fractal image generation by translating the evolution of graph grammar into graphics output. It uses the algebraic approach of graph grammar as a powerful abstract modeling tool for the generation of self-similar fractal images via its graph production, derivations and double-pushout construction. We focus on three proposed methods for image generation which exhibit a certain analogy with fractals. Validation of our approach is given in terms of experimental results. An investigation of the relationships between the generated images and their corresponding graph grammar is also discussed.

Keywords: Graph Grammars, Fractal Geometry, Escape-time algorithm, Strange attractors, Graph productions, Double-pushout construction, Pattern Generation

1 Introduction

The generation and recognition of artificial pictures and patterns are challenging tasks in computer science and other applied areas. This paper will be consider a graph grammars [23,23] as a key solution for such challenging via three proposed methods. Graph grammars are the generalization of string grammars to graphs. Among the various formulations of graph grammars, the so called "algebraic approach" [11,12,22] is one of the most successful because it comprises many results concerning image understanding [7], recognition [6], image synthesis [16,24], analysis and processing [1]. On the other hand, fractal methods are quite popular for the modeling of natural phenomena in computer graphics ranging from random fractal models of terrain [20], to deterministic botanical models such as L-systems, iterated function system and recurrent iterated function system [21,3,4]. These models allow representation of a large class of objects [2,9,25]. Unfortu-

nately, the application of L-systems is fairly limited due to their lack of support for time-varying representations, it doesn't take the fractal nature of the sets into account, processing it as ordinary sets and inefficient geometry specification, whereas the disadvantage of both iterated and recurrent function system are the difficulty in fractal generations and limitation for the class of image generation. Compared with these models, graph grammar is suggested to use because it is independent of geometry specification, simple for image generation, easy to use due to graphics visualization, and has the ability to generate unlimited class of fractal patterns. In this paper we present a new formalism for the concept of fractal iteration in the escape-time algorithm [13,14,19,26] and chaos game [5,18] applied to the graph grammars evolution. We will regard a graph grammar as a filter for image generation via three proposed methods, namely GG-escape-time, GG-simple, and GG-chaos filter methods. Validation of our approach is given by discussion and an illustration of some experimental results.

The paper is organized as follows. Section 2 introduces the basic notions of graph grammar, and in particular of the algebraic Double-Pushout (DPO) approach. Section 3, points out the relationship between fractal image generation methods and their corresponding graph grammar by introducing our three proposed methods. Finally, some experimental results with discussion and conclusions are given in sections 4 and 5 respectively.

2 Mathematical Background and Fundamental Concepts

A graph grammar allows one to describe finitely a collection of graphs, i.e. those graphs which can be obtained from a start graph through repeated applications of graph productions. Each production can be applied to a graph by replacing an occurrence of its left-hand side with its right-hand side. In order to generate fractal patterns, we will manipulate and generalize the notion of graph grammars using the DPO approach [10,11,22] into what we shall call typed graph grammars as a tool for such generation. Here, by type we mean a concept that include what usually is called a label, or a color, or similar. A typed graph grammar is a combination of typed graph and typed graph productions defined as follows:

Definition 2.1 (Typed Graph (TG)) *Given two fixed alphabets Ω_E and Ω_V for edge and vertex labels respectively, a typed graph ($over(\Omega_E, \Omega_V)$) is a tuple $G = (E, V, s, t, \ell_E, \ell_V)$, where, V is a finite set of nodes with three kinds (base node (bn), mother node (mn), and parent node (pn)), E is a finite set of edges, which connect either (mn) and (pn) or (bn) and (mn), $s, t : E \to V$ are the source and target functions, and $\ell_E : E \to \Omega_E$, $\ell_V : V \to \Omega_V$ are the edge and the vertex labeling functions respectively.*

From a graphic point of view, base node, mother node, and parent node will be represented by the symbols $\ominus$, $\bigcirc$ and $\circ$, respectively. An illustrative example for a typed graph is given in figure 1. It consists of three mother nodes namely a, b and c, one base node t, and three parent nodes, two of them connected with a and the other connected with b. The mother nodes may or may not connected with parent nodes as node c has no connection with any parent nodes. For a base node t, the mother nodes a and b are called its preconditions and c is its postcondition. The number of arc weights from a to t, b to t and t to c are 2, 1 and 2 respectively. In TG, no arc weights from the mother nodes to the parent nodes. Parent nodes represent the state of a graph after some application of both graph production and graph derivation.

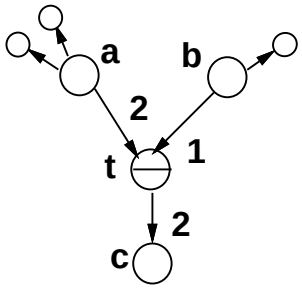


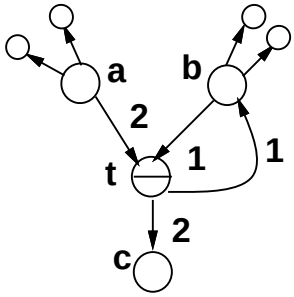
Figure 1: Example for TG

Definition 2.2 (TG-Production) *A typed graph production* $p = (L \leftarrow^{l} K \rightarrow^{r} R)$ *is a pair of injective typed graph morphisms* $l : K \rightarrow L$ *and* $r : K \rightarrow R$. *The typed graphs* L, K, *and* R *are called the left-hand side, the interface, and the right-hand side of* p, *respectively.*

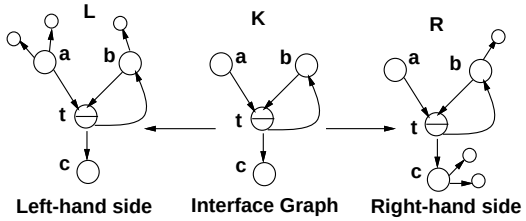
Definition 2.3 (TGG) *A typed graph grammar* $TGG = (\{P_t\}_{t \in B}, G_0)$ *is a finite set of TG-productions, together with a start typed graph* G_0. B *is the set of all base nodes in* G_0 *and for each* $t \in B$ *there is an applicable production* $p_t = (L \leftarrow^{l} K \rightarrow^{r} R)$ *to a graph* G_0 *defined as follows:*

- L *is a typed graph assigning* n *parent nodes for each input mother nodes of the base node* t, *where,* n *is the the number of arc weight connecting a base node with the input mother nodes.*
- K *is the empty graph interface, consist of only the base and mother nodes.*
- R *is a typed graph assigning* n *parent nodes for each output mother nodes of the base node* t, *where,* n *is the the number of arc weight connecting the base node with the output mother nodes.*

A TG-production consumes the parent nodes in the preconditions and generates parent nodes in the postconditions of the base node t, while the interface graph is always empty. Figure 2(b) illustrates the corresponding typed graph production of the base node t for a given typed graph G of figure 2(a).



(a) Typed Graph G with base node t



(b) Corresponding TG-production

Figure 2 Corresponding production for t

In figure 2(b), the number of connected parent nodes with the mother nodes of both L and R depend on the number associated with each arc weight from the base node t to its corresponding in-coming mother nodes a and b for L and out-coming mother nodes b and c for R. According to the DPO constructions [11,22], for a base node t, the typed graph H is called directly derivable from a given typed graph G via a TG-production p_t if and only if the diagram of figure 3 can be constructed. Such derivation is denoted by $G \rightarrow^{p_t} H$.

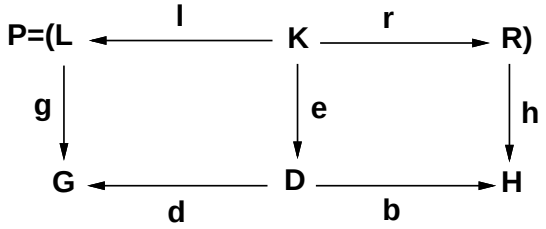


Figure 3. Derivation as DPO construction

The derived graph H is constructed from G by deleting every parent nodes for each preconditions of G which matches parent nodes of L that has no corresponding parent nodes in R. Symmetrically, we add to G each parent nodes of R that has no corresponding parent nodes in L. All remaining parent nodes of G are preserved. Thus,

roughly spoken, the derived graph H is constructed as $H = G - (L - R) \cup (R - L)$. The context graph D is obtained from G by deleting all parent nodes of G which have a pre-image in L, but none in K. Roughly spoken, D can be constructed as $D = G - L + K$. For example, a corresponding direct derivation obtained from the TG-production of figure 2(b) for a base node t applicable to a typed graph G of figure 2(a) is shown in figure 4.

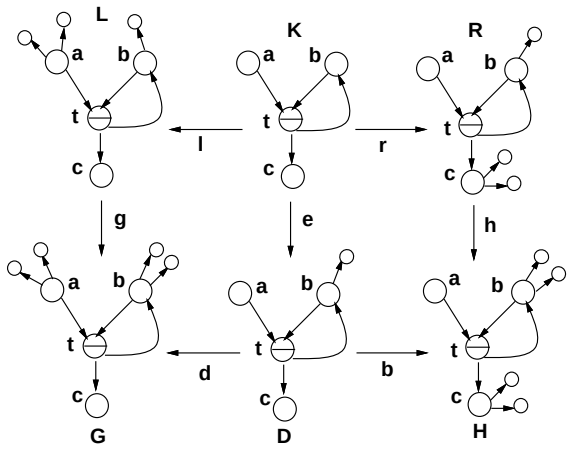


Figure 4. DPO direct derivation diagram

3 *TGG* Tool for Image Synthesis

In this section three different proposed methods for generating fractal images are described. The three methods depend on the graph production and the DPO construction for a given base node of a labeled typed graph G discussed in section 2. In principle, graph grammars have several degrees of freedom: in particular, the resulting labeled typed graph after some applicable production, the sequence of direct derivation or the number of derivations could be selected for display via the distribution of the parent nodes in G. On the other hand, the procedure for generating fractal images are given as an iterative mapping function from pixel coordinate on the screen to pixel color. Therefore, the relationship between the graph grammar and the generated images is to search for a given labeled typed graph for two functions g_1 and g_2 such that g_1 is a mapping from the spatial coordinates (x, y) on some region D on the screen to the initial labeled typed graph G_0 then, after some direct derivation of a TG-production, the function g_2 maps the obtained labeled typed graph G_1 into a color $C(x, y)$. For each of the three proposed methods, we will discuss some suggested rules for such mapping functions.

3.1 Suggested Synthesis Rules

To map the space of pixel coordinates (x, y) onto the space of labeled typed graph we need mapping functions $g1 : N^2 \to N^m$ and $g_2 : N^m \to N^3$ where, m is the number of mother nodes in the typed graph. We have tested two possible classes of choices for g_1:

- $g1 = A \circ B$, is the composition of two functions say, A and B, where, $A : N^2 \to N$ and $B : N \to N^m$. The function $A(x, y)$ is strongly characterizes the image pattern, while B acts on colors. For instance, B may be selected in such a way as to put the same number of the parent nodes in a labeled graph into each mother node that is, $B_j(y) = y, j = 1, \cdots, m$. The other case is to choose the function g_1 as a projection function, that is $g_1(x, y) = (x, y, 0, 0, \cdots)$. This means that in order to obtain the initial labeled graph G_0, the values of the point coordinates are respectively assigned as connecting parent nodes to two mother nodes (called pixel nodes), chosen from the m mother nodes.
- For the function g_2, an RGB representation of colors is chosen, $g_2 : N^m \to N^3$, where we map the connected parent nodes of three selected mother nodes into RGB color components respectively.

3.2 GG-escape-time Filter Method

The escape-time algorithm was originally developed as a method for visualizing the dynamics of complex quadratic fractals. It consists of testing how fast points z outside the attractor diverged to infinity while iterating in the complex plane the function:

$$f_c(z) = z^2 + c \qquad (1)$$

where, c is some constant. Given an initial point z_0 and a function f_c, the points in the orbit $(z_1, z_2, \cdots)$ are defined recurrently as:

$$z_i = f_c(z_{i-1}) \tag{2}$$

By analogy with the escape-time algorithm for generating both Julia and Mandelbrot sets, the GG-escape-time method assigns pixel colors on the basis of the number of iterations of the graph grammar derivation before a particular condition on the labeled typed graph is fulfilled. The resulting image produced by these method performs the same procedure as the escape-time algorithm because color counters indicate the time required for the labeled typed graph derivation to be terminated after some satisfied condition chosen on the parent nodes. As a termination condition we consider a boundary value for a function (i.e. the sum of parent nodes) of some output mother nodes. In a similar way, G_0 can be calculated from G_1 by simply moving the parent nodes from the output mother nodes into the input mother nodes and resetting the others. The flowchart of this method is given in figure 5.

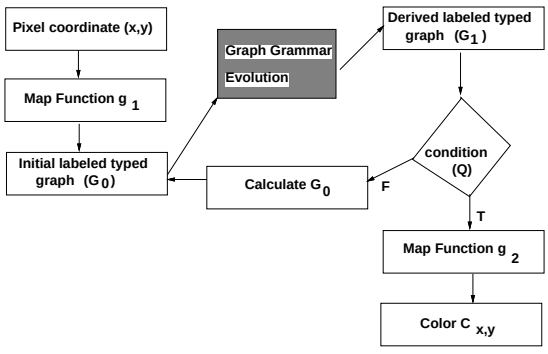


Figure 5. GG-escape-time flowchart

Assume the monitor has a graphical resolution $a \times b$ points and K colors then steps for generating image patterns of this method are given as follows:

Step1 (Initialization)

• Select some suitable attractor domain D for drawing.

• Fix some number of iterations M_{max}.

• Fix the number of derivation sequence.

Step2 (Loop)

• For all pixels (i,j) in the domain D with $i = 1 \rightarrow a$, $j = 1 \rightarrow b$ do the following:

• Calculate the initial labeled graph as $G_0 = g_1(x, y)$.

• set the number of iteration $n = 0$.

Step3 (iteration)

• Construct the production p applicable to a graph.

• Apply p on G_0 to get G_1.

• $n = n + 1$.

Step4 (evaluation)

• If $(not\, Q)$ and $(n < M_{max})$ then calculate G_0 from G_1 and go back to step 2.

• If $(n = K)$ then choose the red as a color c, and goto step 5.

• If (Q) and $(n < M_{max})$ then calculate color c as
$c = g_2(n)$ and goto step 5.

Step5 (Color Assignment)

• Assign color c to the pixel (i, j), and goto the next pixel starting with step 2.

3.3 GG-Simple Filter Method

The mathematical principle behind the generation of the fractal shapes involves the iteration of algebraic transformation. Consider a mapping $T : R^n \rightarrow R^n$ of n-dimensional Euclidean space onto itself. If x is a point in R^n, successive applications of T define a sequence of points in R^n as:

$$\left.\begin{array}{c} x_0 = x, \\ x_1 = T(x), \\ x_2 = T(T(x)), \\ x_3 = T(T(T(x))), \cdots \end{array}\right\} \tag{3}$$

The result of such iteration ordinary depends on the starting point x. When that is the case points on R^n may be classified according to the results of such an iteration. Equation (3) forms a basis for using graph grammar as powerful modeling tool for simulating such iteration via the derived labeled typed graph process of some base node. In this methods a graph grammar is viewed as a black box that evolves by moving the parent nodes among the mother nodes: given an initial

labeled typed graph G_0 it returns a derived labeled typed graph G_1 after some DPO direct derivation construction diagrams is performed. In this case a picture can be thought as a mapping function from pixel coordinates on the screen to pixel colors. Thus, the function g_1 maps the pixel coordinates (x, y) to the initial labeled typed graph G_0, and g_2 mapping the resulting derived labeled typed graph G_1 into a color $C(x, y)$. The flowchart of such method is shown in figure 6.

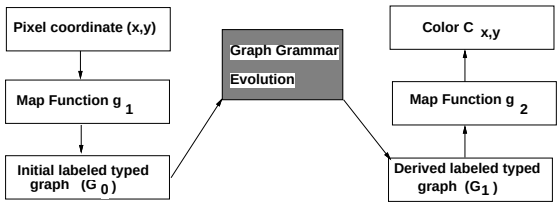


Figure 6 GG-simple method flowchart

The algorithm process is given as follows:

Step1 (Initialization)
• Select some suitable attractor domain D for drawing.
• Fix the number of derivation sequence.
Step2 (Loop)
• For all pixels (x,y) in the domain D do
Step3 (Iteration)
• Calculate the initial labeled typed graph as $G_0 = g_1(x, y)$.
• Construct the production applicable to a graph.
Step4 (Color Assignment)
• Assign color to the pixel (x, y) using the final derived labeled typed graph G_1 as: $Color(x, y) = g_2(G_1)$.

3.4 GG-Chaos Filter Method

Fractals in general have an equally valid existence as a limit of random processes. The chaos algorithm generates a sequence of points which fall onto or near the attractor[15]. Mathematically, the chaos process is described by an IFS and the rate of convergence of points towards the attractor is determined by the contractivity of the IFS mapping [8]. IFS is defined as a pair $\{X; T_n, n = 1, 2, \cdots, N\}$ where, X is a complete metric space and each T_n are affine contractions. According to a theorem by Hutchinson [17], there exists for each IFS a single compact non-empty set $\hat{A}$, called its attractor. The attractor of a random algorithm can be viewed as a limit of a random process. It starts with a point x_0 in a metric space X and generates a sequence of points as

$$x_{t+1} = (T_j)(x_t) \tag{4}$$

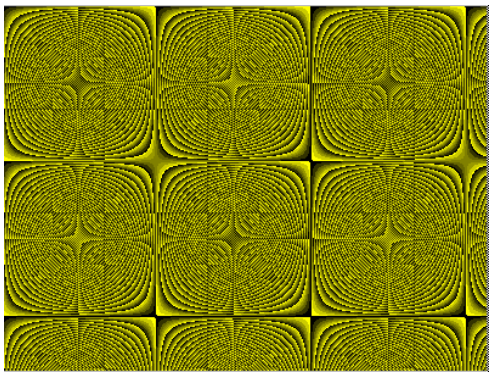
where j is an integer from one to N chosen randomly for each new point in the sequence. In GG-Chaos filter method the new points are calculated in terms of the direct derivation of a TG-production for a given base node instead of the contraction mappings $\{T_j, j = 1, 2, \cdots, N\}$ of the IFS. This method uses information derived from the derivation of a graph grammar in the generation procedure. Given a labeled typed graph G, let the graph grammar evolve by plotting a point on the screen each time a derivation occur. The coordinate of the points of the resulting image are calculated as a function for the derivation of a given base node and its color as a function of the resulting labeled derived typed graph. The algorithm of such method is given as follows:

• Assign to each base node b a point $u_b = (x_b, y_b)$ in the plane which forms a vertex of some regular polygon.
• Set some initial point $p_0 = (x_0, y_0)$.
• For some fixed number of times do
• Randomly, apply a corresponding direct derivation for a base node b_k for some k via a DPO construction.
• Calculate the pixel color from the resulting labeled derived typed graph.
• Plot $\grave{x} = (u_k + x_0)/2$, $\grave{y} = (u_k + y_0)/2$.
• Set (x_0, y_0) to the new points: (i.e. $x_0 = \grave{x}$, $y_0 = \grave{y}$).

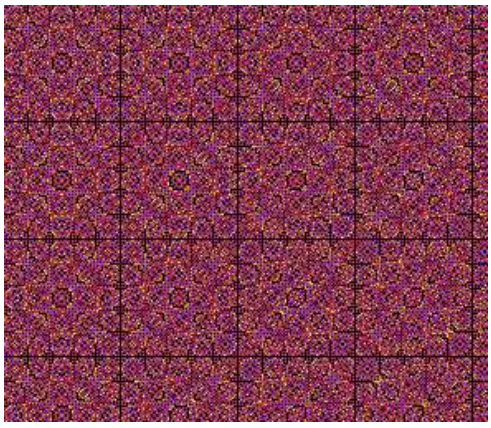
According to the above procedure, the derivation is assigned to the spatial information (i.e. pixel coordinates) and the derived labeled typed graph to color information.

4 Experimental Results and Discussion

Image patterns and colors depend both on the form of the labeled typed graph TG, the upper bound for the number of derivation sequence set, the domain D for drawing, and the class of mapping functions used in the generation process. For instance, the resulting images from the GG-simple filter method are mainly affected by the form of the typed graph TG, and dependent on the class of mapping function chosen for g_1, where different class of mapping chosen leads to different images. Geometric shape images have been created by the GG-filter method. Figure $7(a)$ shows some generated image with some chosen function for g_1. Figure $7(b)$ illustrates the case of choosing the function g_1 as a projection function. The corresponding TGG for the two cases is given in figure 8. For the GG-escape time filter method, the resulting images are mainly affected by the features of the typed graph TG, the number of iterations, the domain D for drawing, and the particular condition Q chosen for the parent nodes. Since infinity is always an attractor for escape-time process, we set ourselves to the goal of colors. The colors are to indicate how long it takes a point (x, y) to escape towards infinity. For these technical reason, we choose red color to distinguish between the escaping points from the non-escaping points. Escaping points will assigned a red color, while the non-escaping points will assigned a color depend on the iteration process. The process of our algorithm is quite similar to the generation process of the well known Julia and Mandelbrot sets, but in our work we use graph production, derivation and DPO construction instead of the original quadratic function used for image generation. By this way we can generate unlimited class of images. A lot of concentric shape images can be generated using our method. Figures $9(a)-(d)$ show some generated images for such method with its corresponding TGG given in figure 10. For the GG-chaos filter method the resulting images are mainly affected by the form of the typed graph TG and the number of its base nodes with the point coordinates assigned for each one. Different point coordinates will leads to different images. Many fractal images can be generated by this method. Figure $11(a)-(d)$ show some generated images. The corresponding investigated typed graph grammar TGG with different coordinate points assigned to each base node is shown in figure 12. In principle, investigating other typed graph grammar with different features, and different coordinate points assigned to each base node will leads to other generated images.



(a) $g_1(x, y) = 2xy$



(b) $g_1(x, y) = (x, y, 0)$

Figure 7. Some generated images by GG-simple filter algorithm

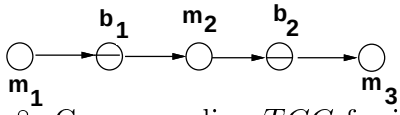
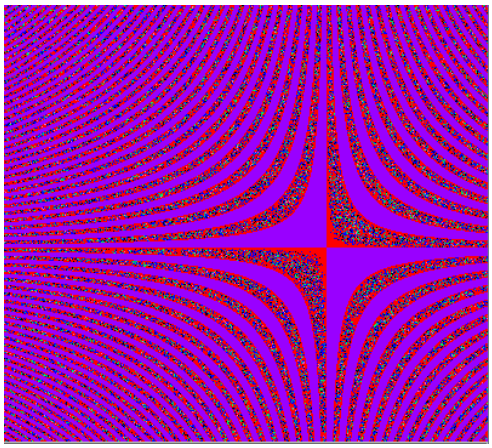
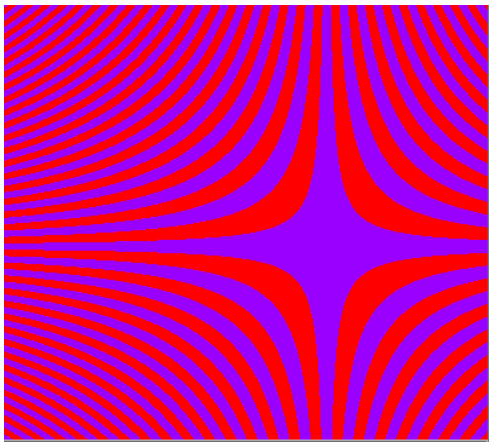


Figure 8. Corresponding TGG for images of figure 7



(a)



(b)

Figure 9. Some generated images by GG-escape-time filter algorithm

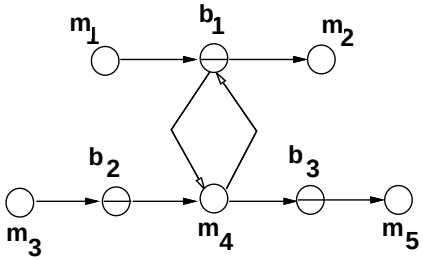
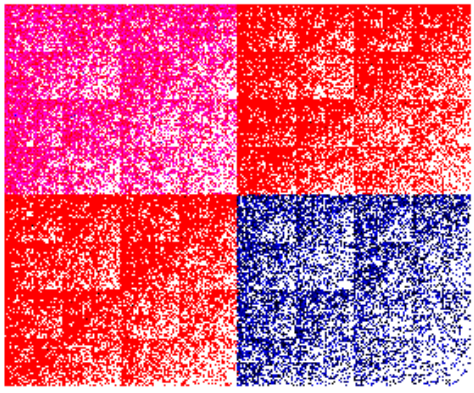
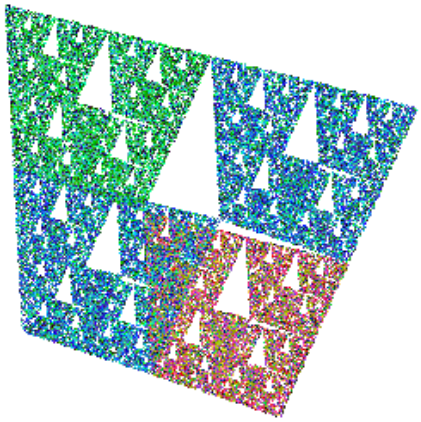


Figure 10. Corresponding *TGG* for images of figure 9.



(a)



(b)

Figure 11. Some generated images by GG-chaos filter algorithm

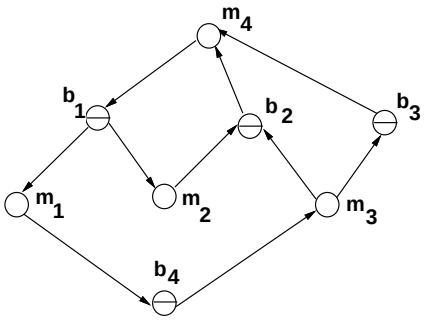


Figure 12. Corresponding *TGG* for images of figure 11.

5 Conclusion

In this paper we have presented a new approach for computer image generation using the algebraic approach of graph grammar. Three proposed methods are shown and discussed which exhibit a self-similarity with fractal iteration principle in escape-time algorithm and chaos game. We regarded a graph grammar as a filter for image generation via its graph production, derivation and *DPO* construction. In particular, we have shown how some techniques, commonly used in practice for fractal image generation, can be applied to a powerful modeling tool like graph grammar. The three methods proposed differ from each other in the graph grammar features displayed: the final labeled typed graph after a sequence of direct derivation, the number of the base nodes in the typed graph G and its TG-production applicable to G during a simulation. The

visual aspects of the resulting pictures are not independent of the generation process, which contributes to determine both patterns and colors: geometric shape images have been created by the GG-simple filter method, concentric shape images by GG-escape-time filter method and fractal images by GG-chaos filter method. Thus, a visual translation of the evolution of a graph grammar should reflect the behavior of the graph itself and in that case could be used as a new kind of analysis method. Further research effort is needed to derive more classes of images and to carry out a deeper analysis of the relationships between images and graph grammar.

REFERENCES

[1] Aizawa, K., and Akira N., " Path-Controlled Graph Grammars for Multiresolution Image Processing and Analysis". Volume 776 of LNCS, pp.1-14, Springer Verlag, 1994.

[2] Alastair N.Horn, "IFS and Interactive Image Synthesis", Comp. Graph. Forum, Vol. 9, pp. 127-137, 1990.

[3] Barnsely, M.F., and Demko S.G., "Iterated function systems and the global construction of Fractals", Proceedings of the Royal Society of London Series A 399, pp. 243-275,1985.

[4] Barnsely, M.F., "Recurrent iterated function systems", Constructive Approximation Vol. 1, No. 1, 1989.

[5] Barnsely, M.F., Malassenet F. and Jacquin A., "Harnessing Chaos for Image Synthesis", Proc. of SIGGRAPH '88, In Computer Graphics Vol. 22, No.. 4, pp. 131-140, 1988.

[6] Bartsck S., "Grammatical Interface of Graph Grammars for Syntactic Pattern Recognition". Volume 153 of LNCS, pp. 1-7, Springer Verlag, 1983.

[7] Bunke, H., "Graph Grammars as a generative tool in Image Understanding", Volume 153 of LNCS, pp. 8-19, Springer Verlag, 1983.

[8] Claude T., Lutton E., and Vehel J., Fractals in Engineering, "Modeling fractal shapes using generalizations of IFS techniques", by J. THOLLOT et al., Springer Verlage, pp.65-80, 1997.

[9] Casey S. D. and Reingold N.F., "Self-similar fractal sets: Theory and Procedure", IEEE Comp. Graph. & Appl., Vol. 14, No. 3, pp. 73-82, 1994.

[10] Corradini A., Ehrig H., Lowe M., Montanari U., and Rossi F., "Abstract Graph Derivations in the Double-Pushout Approach", Volume 776 of LNCS, pp.86-103, Springer Verlag, 1994.

[11] Ehrig H., Korff M., and Lowe M. "Tutorial introduction to the Algebraic Approach of Graph Grammars Based on double and single pushouts", Volume 532 of LNCS, pp. 24-37, Springer Verlag, 1991.

[12] Ehrig H., Pfender M., and Schneider H., "Graph Grammars : An algebraic approach". In Proceedings IEEE Conf. on Automata and Switching Theory, pp. 167-180, 1973.

[13] Hussein, K.H., Aboul Ella, H., M. Nakajima, "Fractal Implementation of Higher order Mapping Based on Escape-time Algorithm", ITE Annual Conference, D-12-112, 1998.

[14] Hussein, K.H., Aboul Ella, H., M. Nakajima, "Efficient Linear Fractal Algorithm for Computing The Continuous Escape-Time Classifications", Technical Report of IEICE, Vol. 97, No. 558, pp.125-130, 1998.

[15] Hussein, K.H., Aboul Ella, H., M. Nakajima, "Chaos Simulation Algorithm For Generating Fractal Images

Based on Graph Grammar Theory", ITE Annual Conf., D-11-84, 1998.

[16] Hussein, K.H., Aboul Ella, H., M. Nakajima, "A Novel Fractal Image Generation Technique Based on Graph Grammar Theory", VSMM'98, 4^{th} International Conference on Virtual Reality and Multimedia, Vol.2, pp.504-510, 1998.

[17] Hutchinson, J.E., "Fractals and self-similarity", Indiana University Journal of Math. Vol. 30, No. 5, pp. 713-747, 1981.

[18] Jeffrey, H. J., "Chaos Game Visualization of Sequences", Comp. & Graph, Vol. 16, pp. 25-33, 1992.

[19] Mandelbrot B.B., "The fractal geometry of nature", W. H. Freeman, New York, 1982.

[20] Peitgen H-O, and Saupe D., "The science of Fractal Images", Springer-Verlag, New York, 1988.

[21] Prusinkiewicz, P., "Graphical applications of L-systems" , Proc. of Graph. Interface, pp. 247-253, 1986.

[22] Rozenberg G., "The Handbook of Graph Grammars", Volume I: Foundations. World Scientific, 1997.

[23] Schneider H., and Erlangen, "Graph Grammars", Volume 56 of LNCS, pp. 314-331, Springer Verlag, 1977.

[24] Siromoney, R. and Subramanian, K., "Space-filling curves and infinite graphs", Volume 153 of LNCS, pp. 380-391, Springer Verlag, 1983.

[25] Smith, A. R., "Plants, Fractals, and Formal Language", Computer Graphics, Vol. 18, No. 3, pp. 1-10, 1984.

[26] Sprott J.C. "Automatic generation of general quadratic map Basins", Computer and Graphics, Vol. 19, No. 2, pp 309-313, 1995.