UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO

(Bacharelado)

PROTÓTIPO DE UMA FERRAMENTA BASEADA EM SISTEMAS DE LINDENMAYER PARA MODELAGEM DE ESTRUTURAS FRACTAIS

TRABALHO DE CONCLUSÃO DE CURSO SUBMETIDO À UNIVERSIDADE REGIONAL DE BLUMENAU PARA A OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA COM NOME EQUIVALENTE NO CURSO DE CIÊNCIAS DA COMPUTAÇÃO — BACHARELADO

JOSÉ CARLOS DOS SANTOS

BLUMENAU, DEZEMBRO/1999

1999/2-20

Dedico este trabalho à minha família, especialmente em homenagem à minha sobrinha Laiza Cristina da Costa. Um agradecimento especial a todos, professores e amigos, que de alguma forma contribuíram para a sua realização.

PROTÓTIPO DE UMA FERRAMENTA BASEADA EM SISTEMAS DE LINDENMAYER PARA MODELAGEM DE ESTRUTURAS FRACTAIS

JOSÉ CARLOS DOS SANTOS

ESTE TRABALHO DE CONCLUSÃO DE CURSO, FOI JULGADO ADEQUADO PARA OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA DE TRABALHO DE CONCLUSÃO DE CURSO OBRIGATÓRIA PARA OBTENÇÃO DO TÍTULO DE:

BACHAREL EM CIÊNCIAS DA COMPUTAÇÃO

_____

Prof. Dalton Solano dos Reis — Orientador na FURB

_____

Prof. José Roque Voltolini da Silva — Coordenador do TCC

BANCA EXAMINADORA

_____

Prof. Dalton Solano dos Reis

_____

Prof. Antônio Carlos Tavares

_____

Prof. Sérgio Stringari

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

RESUMO

O escopo desse trabalho é apresentar um protótipo de uma ferramenta baseada em Sistemas de Lindenmayer, resumo de Sistemas-L, para modelagem de estruturas fractais. Entende-se aqui por estruturas fractais a formação topológica e geométrica da ramificação de plantas e arbustos, criação de padrões fractais, curvas de preenchimento e ladrilhos. O protótipo é capaz de gerar modelos determinísticos e estocásticos em duas dimensões, permitindo que o modelo gerado possa ser gravado em disco no formato BMP (*Bit Map* – Mapa de Bits) e impresso independente de dispositivo.

ABSTRACT

The goal of this work is to present a Lindenmayer System based prototype tool, in short L-Systems, to model fractal based structures. Fractals based structures is understood here by the topological and geometrical formation of plants and bushes, fractal patterns, filling espace curve and tiles. The prototype is able to genarate deterministic or estochastic models in a two dimensions space, allowing the images generated to be saved on disk in BMP (Bit Map) format or device independent printing .

1 INTRODUÇÃO

Há séculos que o desenvolvimento das estruturas e formas intrincadas naturais vêm atraindo a atenção dos matemáticos [PRU90], simplesmente pelo fato da natureza apresentar-se de uma maneira que a matemática comum não consegue descrever devido as limitações impostas por Euclides e Pitágoras [DEV90]. Ao contrário do que muitos pensam, a matemática não parou no tempo e há poucos anos criou-se uma nova extensão dirigida ao estudo dos Sistemas Dinâmicos.

Sistemas Dinâmicos é uma área da matemática que estuda os processos que se movem e apresentam mudanças através do tempo [DEV90]. Está presente em todos os ramos da ciência, como por exemplo: nos padrões de mudança do tempo, na variação econômica, no desenvolvimento da população, no movimento dos planetas e galáxias. Um dos aspectos dos Sistemas Dinâmicos é a utilização da iteração. A teoria matemática dos Sistemas Dinâmicos tem como objetivo básico determinar ou caracterizar o comportamento dos sistemas ao longo do tempo [WRI96]. Existem vários tipos de Sistemas Dinâmicos, cada um possui métodos matemáticos para representá-los. Este trabalho está mais voltado aos Sistemas Dinâmicos Simbólicos, que utilizam caracteres para determinar ou caracterizar o comportamento de processos dinâmicos naturais ou artificiais.

Sistema-L é um tipo de Sistema Dinâmico Simbólico [WRI96] que foi introduzido em 1968 pelo biólogo Aristid Lindenmayer com o propósito de modelar estruturas biológicas, em particular os padrões de ramificação das plantas, árvores e arbustos. Em 1986, Prezemyslaw Prusinkiewicz incorporou Sistemas-L na computação gráfica. De acordo com [PRU90], a evolução algorítmica dos Sistemas-L assemelha-se com o desenvolvimento genético dos organismos vivos, onde propriedades são adquiridas com o decorrer do tempo. Sistemas-L modelam estruturas fractais tendo como principal função explorar a auto-similaridade, característica visível nos padrões naturais [PRU90]. Pode-se gerar imagens complexas partindo de um pequeno conjunto de dados, a isso dá-se o nome de amplificação de dados.

De acordo com [PRU90] a aplicação da computação gráfica às estruturas biológicas é somente um dos muitos fatores que contribuem para a característica interdisciplinar deste

estudo. Por exemplo, a noção de Sistemas-L é uma parte da teoria da linguagem formal, com base na teoria dos algoritmos. A aplicação de Sistemas-L na descrição de plantas foi estudada por muitos biólogos e envolve vários métodos da matemática geral. A auto-similaridade liga as estruturas das plantas com a geometria fractal. A visualização de imagens dessas estruturas auxiliada por computador e os processos utilizados para criá-las juntam arte e ciência. Podemos concluir então que neste estudo estão envolvidos as seguintes ciências: computação, matemática e biologia.

A geometria clássica é limitada na representação de formas naturais, como já foi dito anteriormente. De acordo com [PEI86], a natureza não exibe simplesmente um grau maior, mas um nível de complexidade totalmente diferente, com padrões, escalas e tamanhos praticamente infinitos. A existência desses padrões desafia os matemáticos a estudar essas formas que Euclides ignorou como sendo “sem forma”. É, com certeza, de responsabilidade da Computação Gráfica apresentar soluções para a representação e validação dessas teorias.

Algumas característica deste protótipo são extensões a um trabalho feito, em 1993, por Vitor César Benvenutti [BEN93] que sugeriu uma interface gráfica, estudo de outros tipos de Sistemas-L e utilização de uma estrutura de dados mais dinâmica.

1.1 OBJETIVO

O objetivo desse trabalho é implementar um protótipo de ferramenta baseada em Sistemas de Lindenmayer ou Sistema-L determinísticos e estocásticos para modelagem de estruturas fractais em duas dimensões, assim como: ramificação de plantas e arbustos, estruturas fractais, texturas e curvas de preenchimento. O protótipo é baseado em teorias de Aristid Lindenmayer com técnicas de reescrita paralela e interpretação gráfica para geração das imagens.

1.2 ORGANIZAÇÃO DO TEXTO

O texto está organizado em seis capítulos de forma a permitir um melhor entendimento do assuntos apresentados. Cada capítulo apresenta inicialmente uma visão geral do que irá ser abordado e no final um breve resumo do que foi apresentado.

O capítulo 2 introduz uma noção básica aos Sistemas Dinâmicos, Caos e Fractais. São apresentados elementos básicos que formam as imagens fractais e um breve histórico das pessoas envolvidas neste estudo.

Já o capítulo 3 apresenta os fundamentos dos Sistemas de Lindenmayer, resumo de Sistemas-L, envolvendo a noção de Sistemas Dinâmicos Simbólicos, reescrita, interpretação gráfica e Sistemas-L determinísticos e estocásticos. A formação de agrupamentos e ramificação também será abordada.

O capítulo 4 apresenta detalhes do protótipo implementado. É explicado o que é possível de ser feito com a implementação, demonstra a arquitetura na qual foi desenvolvido, a modelagem, as estruturas de dados utilizadas e, finalmente, como usar o protótipo.

Análise dos resultados e comentários são apresentados no capítulo 5. Imagens que lembram padrões naturais, fractais, curvas de preenchimento e texturas bem como suas definições são apresentadas neste capítulo. Também são comentados os pontos positivos e as limitações do protótipo.

O capítulo 6 apresenta conclusões, melhoramentos e sugestões para novos trabalhos.

1.3 CONCEITOS BÁSICOS E TRABALHOS CORRELATOS

Os conceitos mais básicos envolvem a noção de fractais, suas origens e principais características: auto-similaridade, dimensão fractal e iteração. Todos esses conceitos são apresentados no capítulo 2.

O estudo dos Sistemas de Lindenmayer são baseados principalmente nas teorias de reescrita paralela e interpretação gráfica. São adequados para implementação em sistemas computacionais, tais como os chamados “Laboratórios Virtuais”. São grandes sistemas utilizados em laboratórios de botânica para o estudo de plantas extintas, desenvolvimento celular e para gerar paisagens virtuais para cenários em filmes. Alguns trabalhos feitos utilizando-se Sistemas-L estão listados abaixo [PRU90].

- *Plant and fractal generator (PFG)*: foi implementado por Przemyslaw Prusinkiewicz e J. Hanan. Gerador de estruturas fractais com saída para arquivo ou

impressora. Pode ser configurado para trabalhar com modelos paramétricos ou não.

- *Modeling program for phyllotactic patterns (Spiral)*: implementado por D. R. Fowler. Sistema interativo desenvolvido que modela padrões filotaxômicos, ou seja, o modo como as folhas estão dispostas no caule de certas plantas.

- *Interactive surface editor (Ise)*: implementado por J. Hanan. Pode ser utilizado para modificar ou definir superfícies bicúbicas consistindo de uma ou várias partes conectadas arbitrariamente. A saída pode ser utilizada pelo PFG ou Spiral.

- *Modeling program for cellular structures (Mapl)*: implementado por F. D. Fracchia. Esse sistema lê uma definição bidimensional de uma camada celular capturada por um mapeamento Sistema-L e gera a seqüência do desenvolvimento celular utilizando o método da interpretação do mapeamento.

Embora seja dada uma ênfase maior na computação gráfica [PRU90], os Sistemas-L são utilizados em grande escala em diversas sub-áreas das Ciências da Computação. Em [ROZ92] são mencionadas vários trabalhos nesta área.

2 SISTEMAS DINÂMICOS, CAOS E FRACTAIS

Este capítulo introduzirá de forma geral o que é Sistema Dinâmico, um tópico atual para pesquisadores matemáticos ou não. Apresentará também um resumo da teoria do Caos e fractais com suas definições e principais características, bem como sua relação entre fenômenos e padrões naturais e artificiais. Também será apresentado um breve histórico das pessoas envolvidas no estudo dos fractais, pois muitas imagens apresentadas possuem o nome de seus criadores.

2.1 SISTEMAS DINÂMICOS

Um sistema dinâmico, basicamente, é qualquer processo que evolui no tempo [PEI88]. Os sistemas dinâmicos acontecem em todos os ramos da ciência, como por exemplo: na mudança do padrões climáticos (meteorologia), nos altos e baixos na balança comercial (economia), no desenvolvimento da população (ecologia), no movimento dos planetas na galáxia (astronomia), no desenvolvimento de organismos vivos (biologia) e reações de elementos químicos (química), entre outros [DEV90]. Um sistema dinâmico pode estar presente tanto em processos naturais quanto em processos artificiais.

Os sistemas dinâmicos lidam com processos físicos, químicos, biológicos e matemáticos com o objetivo básico de prever eventuais resultados dos processos de evolução. Existem vários tipos de sistemas dinâmicos, exibindo características comportamentais naturais ou artificiais, deste modo, a matemática possui meios de representar a evolução de tais sistemas. Nesse trabalho será utilizado o tipo sistema dinâmico simbólico, que utiliza caracteres para representar evolução de um processo, o qual adquire complexidade de acordo com um processo particular de evolução.

De acordo com [PEI88], alguns sistemas dinâmicos são previsíveis outros não pelo fato de possuírem grande número de variáveis envolvidas, até mesmo aqueles com uma variável podem apresentar comportamento aleatório ou imprevisível. Geralmente esse

comportamento é proveniente dos sistemas dinâmicos não lineares, onde pequenas mudanças nas variáveis de controle acarretam grande mudança em todo sistema [CRI91]. O comportamento imprevisível dos sistemas dinâmicos fez com que os matemáticos criassem a noção de Caos.

2.2 CAOS

De acordo com [WOO95], caos é a irregularidade imprescindível do comportamento nos sistemas dinâmicos determinísticos e não lineares. A incapacidade de se prever o resultado de um sistema dinâmico não está na quantidade de variáveis envolvidas no processo, e sim, na grande sensibilidade das condições iniciais destas variáveis, onde pequenas mudanças podem acarretar resultados totalmente inesperados [DEV90].

O avanço dos estudos relacionados ao caos é conseqüência do avanço da capacidade de processamento dos computadores. Inovações nas técnicas de computação gráfica fizeram com que cientistas e matemáticos progredissem nos estudos dos sistemas dinâmicos que é de onde o Caos deriva [CRI91].

Um exemplo de um sistema dinâmico que possui comportamento caótico é a função $f(z) = z^2 + c$, gerando um dos objetos mais fascinantes e mais difícil da matemática que é o conjunto de Mandelbrot representado na Figura 1 [PEI88]. Outro exemplo de sistema caótico pode ser observado na Figura 2 através do método de Newton para se achar raízes de equações com grau superior a dois, ou seja, valores para *x* que anulam equações sob a forma:

$$f(z) = a_0 + a_1 z + a_2 z^2 + a_3 z^3 + \cdots a_m z^m = 0$$

Este método consiste em se estipular um valor inicial para a resposta e, a partir deste valor, iterar a função até que se aproxime o máximo possível da mesma, obtendo-se uma resposta numérica e um fator de erro (aproximação) [BIR94]. Imagens fractais como a Figura 1 e Figura 2 nos dizem que nem tudo sobre equações ainda é conhecido e que não é um simples gráfico de coordenadas cartesianas que define os seus comportamentos [DEV90]. O conjunto de Mandelbrot e o método de Newton são exemplos de funções que exibem comportamento caótico, exibindo características fractais (auto-similaridade) através de

iterações em um plano complexo. Para conhecer mais à respeito ver [PEI86], [PEI88] e [DEV90].

Figura 1 - Conjunto de Mandelbrot para $f(z) = z^2 + c$ [BOU91]**.**

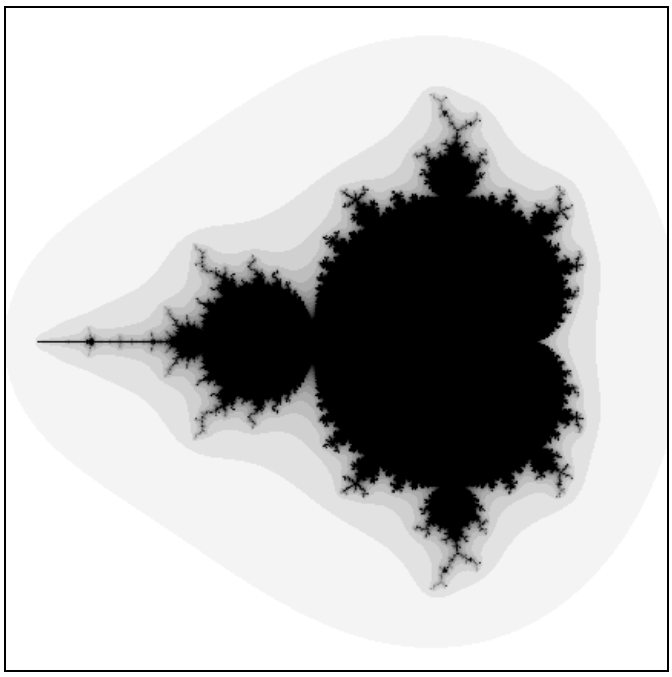
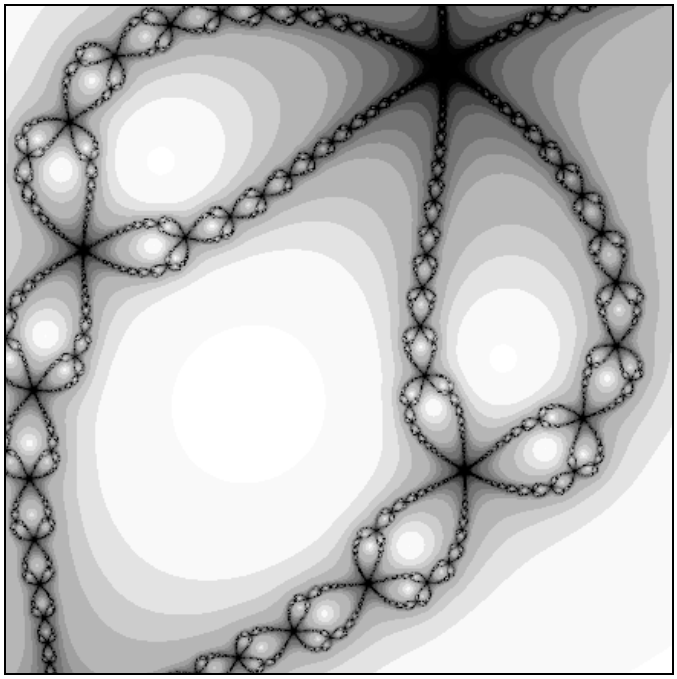


Figura 2 - Comportamento caótico da equação $z^3 - 1 = 0$ [BOU91].



Durante a década de 70, Benoit Mandelbrot, pesquisador da IBM (*International Business Machine*), um matemático não ortodoxo, preferia aplicar geometria na solução dos problemas, ao invés de álgebra. Segundo suas teorias geométricas, nem um pouco clássicas, a resposta gráfica de um sistema consistia em uma noção mais completa do problema. Foi pelo fato de amar a geometria, que percebeu que sistemas aparentemente caóticos possuíam padrões simples de resposta que eram repetitivos e continham, intrinsecamente, um algoritmo de auto repetição, capaz de gerar o todo. A essa forma geométrica chamou de *fractal*, que significa menor fração de um todo auto-similar capaz de gerar o conjunto. A partir desse momento os fractais e caos são utilizados como ferramentas matemáticas para a modelagem visual do comportamento caótico de processos naturais e artificiais.

2.3 FRACTAIS

A virada deste século coincidiu com drásticas transformações no mundo da matemática quando surgiram as primeiras formas geometricamente estranhas, que mais tarde seriam nomeados de fractais: Conjunto de Cantor, funções *Weierstrass*, Curva de Peano, curva de Von Kock, curvas sem diretivas e linha que podiam preencher espaços. Uma vez descobertos, foram mantidos como “monstros” da matemática, justamente por não ser coerente com conceitos já conhecidos pelos matemáticos. Não haviam idéias para aplicação e nenhum interesse em estudá-los. Benoit Mandelbrot, através da computação gráfica e cansativos experimentos provou a aplicação do que foi ignorado pela geometria clássica e achou na natureza razões suficientes para o estudo do geometria fracionária ou simplesmente, fractal.

Benoit Mandelbrot, em 1975 (data do lançamento de seu livro “*The fractal Geometry of Nature*), criou a palavra *fractal*, do adjetivo latim *fractus*, que significa irregular ou fragmentado. Nasceu a Geometria Fracionária, libertando a matemática das limitações impostas por Euclides e Pitágoras há mais ou menos 2000 anos.

Dependendo do ponto de vista, fractal pode assumir várias definições. Benoit Mandelbrot, considerado o pai dos fractais, define-o como sendo um conjunto cuja dimensão Hausdorf-Besicovitch, um método de calcular a dimensão fractal, não é um valor inteiro [PEI86].

O Geometria Fracionária adquiriu com o passar dos anos um caráter interdisciplinar e uniu os matemáticos, as ciências naturais e as ciências da computação, tornando-se, juntamente com seus conceitos, uma ferramenta central na física, química, biologia, geologia, meteorologia e as ciências dos materiais. O caráter interdisciplinar dos fractais não é somente nas ciências, mas também na arte. É largamente utilizado por projetistas gráficos e produtores de filmes na criação de paisagens, cidades, jardins e planetas virtuais e, até na composição de músicas. As imagens fractais podem ser complexas, mas as regras que as governam são simples. Graças a computação gráfica que os fractais foram aceitos como extensão à geometria clássica, a final, uma imagem fractal justifica, sem dúvidas, o seu estudo, considerando que montanhas não são cones, nuvens não são esferas e árvores não são cilindros. Hoje os fractais são utilizados pela computação gráfica na modelagem de fenômenos e objetos naturais.

Pode-se pensar que fractais e caos são sinônimos, mas não são. A teoria do caos é ilustrada através de fractais e existe há séculos. Ambos são ferramentas matemáticas diferentes utilizadas para modelar fenômenos naturais ou artificiais e objetos. Exemplos de palavras chave que descrevem o caos são: imprevisibilidade e sensibilidade à condições iniciais dos processos. Exemplos de palavras chaves que descrevem os fractais são: auto-similaridade e invariação de escala. Muitos fractais não são, de maneira nenhuma, caóticos.

A Tabela 1 mostra um resumo das principais diferenças entre fractais e as formas tradicionais de Euclides. Primeiro, fractal é uma das maiores descobertas do século XX, sem dúvida. Mesmo sendo rejeitado na virada do século, foram reconhecidos como úteis pelos cientistas naturais há pouco mais de 20 anos. Segundo, as formas Euclidianas são baseadas em tamanho ou escalas. Não é possível medir um fractal. São formas auto-similares e independentes de escala. Terceiro, a geometria Euclidiana é utilizada para representar objetos feitos pelo homem, ao passo que, os fractais são utilizados para representar a natureza. Finalmente, formas Euclidianas são usualmente representadas por fórmulas algébricas ($r^2 = x^2 + y^2$, que define um círculo de raio r), fractais, geralmente, são resultado de um procedimento ou uma construção algorítmica recursiva, própria para computadores [PEI88].

Tabela 1 - Uma comparação da geometria fractal e a Euclidiana [PEI88].

GEOMETRIA...	
...é a linguagem matemática usada para descrever, relacionar e manipular formas.	
EUCLIDIANA	***FRACTAL***
• Tradicional (> 2000 anos); • Baseados em escala ou tamanho; • Serve para representar os objetos que o homem cria; • É descrita através de fórmulas.	• Pós-moderno (~20 anos); • Não possui tamanho ou escala específica; • É apropriada para representar formas naturais; • Utiliza algoritmos recursivos.

Os fractais podem apresentar uma infinidade de formas diferentes, não existindo uma aparência consensual, contudo existem características muito freqüentes nesta geometria, como a auto-similaridade, a dimensão fracionária e iteração, que a base para a formação das imagens.

2.3.1 AUTO-SIMILARIDADE

Uma das propriedades básicas dos conjuntos fractais é a noção de auto-similaridade, que é a capacidade de, através de um sub-conjunto, poder representar o conjunto inteiro [CRI91]. Objetos fractais são infinitamente sub-divisíveis e independente de escala, cada sub-divisão produz objetos com características estatísticas ou exatamente iguais ao objeto inteiro.

O conjunto de Mandelbrot (Figura 3), por exemplo, possui auto-similaridade estatística. Parte do desenho ampliada, apresenta formas parecidas com o desenho todo. As figuras 4, 5 e 6 representam três ampliações em cascata de uma parte do conjunto de Mandelbrot (Figura 3) indicada por um retângulo em cada figura. Verifica-se que estatisticamente a forma inteira está representada nas ampliações subseqüentes. Já a curva de von Koch, quando sofre ampliações de uma parte, apresenta exatamente o desenho completo, infinitamente. Este é um exemplo de auto-similaridade exata ou repetitiva, representada na Figura 7.

Propriedades como auto-similaridade estão presentes tanto em formas quanto em fenômenos naturais. Segundo [PRU90], em muitos processos de crescimento dos sistemas

vivos, especificamente os das plantas, é notável a repetição regular das aparências multicelular das estruturas. No caso de uma folha composta, por exemplo, alguns lóbulos ou folhetos que fazem parte de uma folha em um estado mais avançado, possuem a mesma forma que a folha possuía em um estágio anterior. A auto-similaridade nas plantas é um resultado do processo de desenvolvimento.

Figura 3 - Conjunto de Mandelbrot.

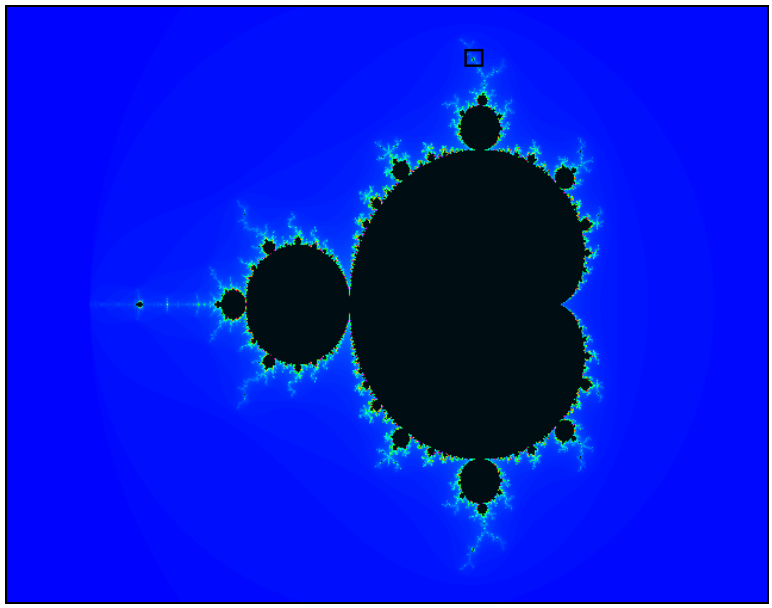


Figura 4 - Primeira ampliação da área indicada com um retângulo na **Figura 3**.

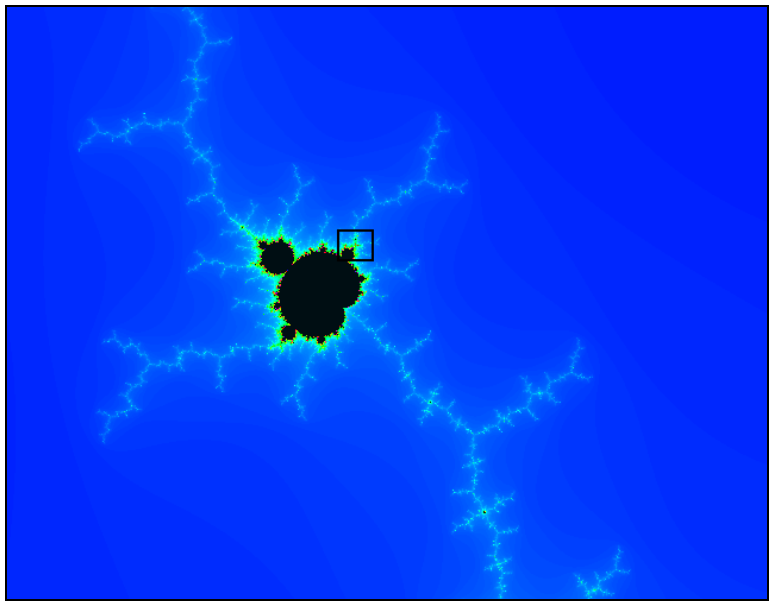


Figura 5 - Segunda ampliação da área indicada por um retângulo na **Figura 4.**

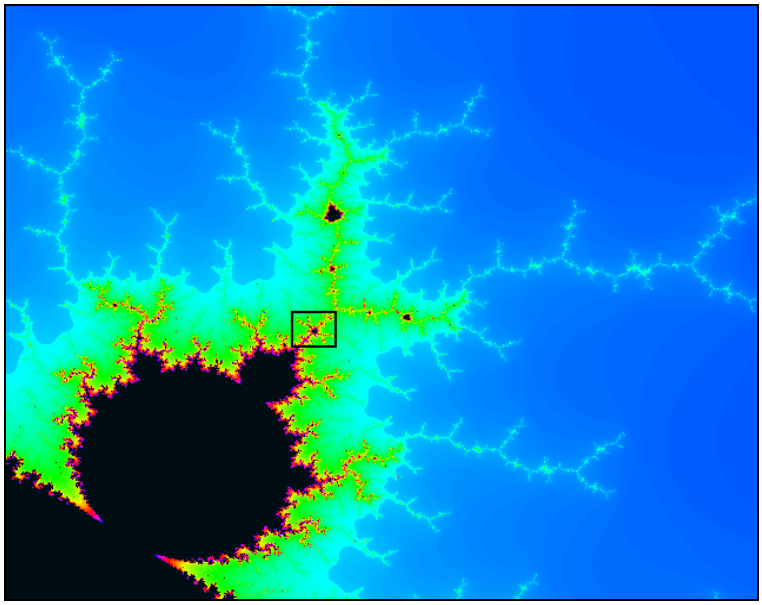
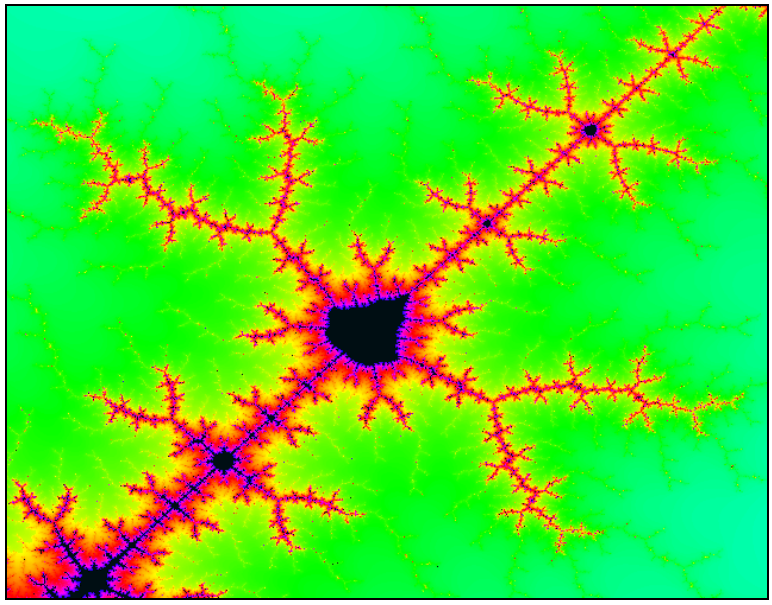
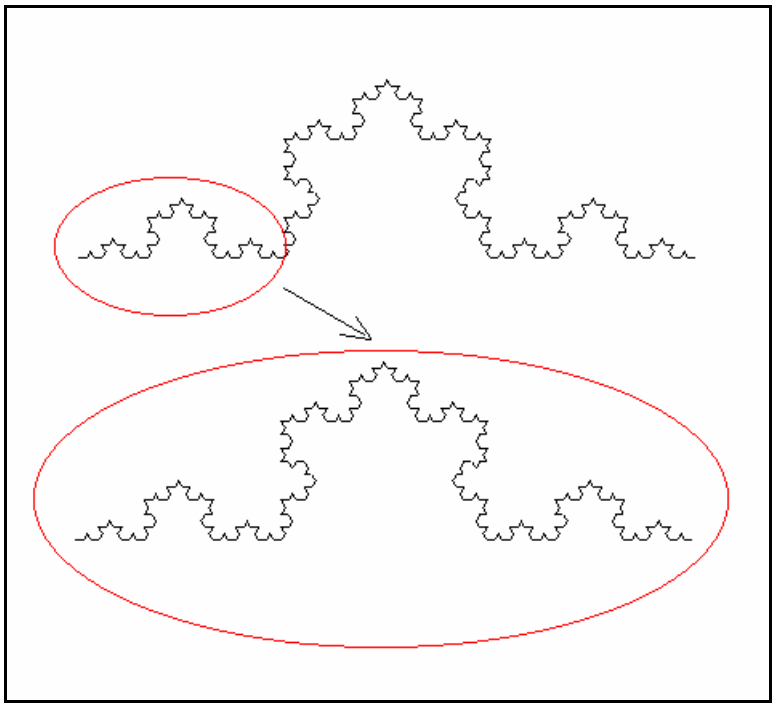


Figura 6 - Terceira ampliação da área indicada por um retângulo na **Figura 5.**



Pode-se fazer uma analogia entre auto-similaridade e as samambaias ou uma couve-flor. A menor parte retirada de uma couve-flor sempre lembrará uma couve-flor inteira. A mesma coisa acontece com uma folha de samambaia, onde pode-se concluir que esta é formada por partes repetitivas de formas reduzidas do todo.

Figura 7 - Curva de von Koch representando alto-similaridade exata.



2.3.2 DIMENSÃO FRACTAL

A propriedade de auto-similaridade ou *scalling*, como foi apresentado anteriormente é um dos conceitos centrais da geometria fractal. Está fortemente relacionado com a noção de dimensão fractal [PEI88]. Um tipo comum de dimensão fractal é a dimensão de Hausdorff-Besicovich, mas há várias maneiras diferentes de calculá-la. Embora exista outras maneiras de explicar o que é dimensão, aqui será utilizada a noção de auto-similaridade.

Hausdorff, matemático alemão, foi quem criou a noção de dimensão fractal, provando que os fractais existem entre as dimensões, ou seja, fractais podem ser encontrados entre um ponto e uma reta; entre uma reta e um plano ou entre uma plano e um objeto no espaço. A dimensão fractal é um meio de medir a complexidade dos fractais. Quanto maior for a dimensão, maior é a complexidade do fractal [PEI88].

Um objeto auto-similar com dimensão D pode ser dividido em N cópias menores auto-similares, cada uma delas reduzida a uma escala representada por [CRI91]:

$$r = \frac{1}{\sqrt[D]{N}} \text{ e } N = \frac{1}{r^D} \text{ ou } N = r^D$$

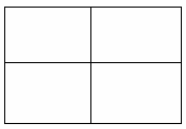
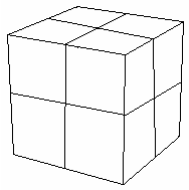
Onde *r* é a proporção de redução; *N* é o número de réplicas auto-similares e *D* é a dimensão fractal, que difere da dimensão euclidiana por não precisar ser, necessariamente, um número inteiro [PEI88].

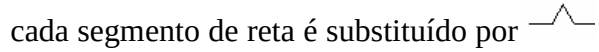
Reciprocamente, dado um objeto auto-similiar de *N* partes reduzidas à uma proporção *r* do todo, sua dimensão fractal ou semilaridade é dada por:

$$D = \frac{\log(N)}{\log(1/r)}$$

De acordo com Tabela 2, objetos como uma reta, um quadrado e um cubo também possuem propriedade de auto-similaridade. Ao calcular-se a dimensão fractal de qualquer um desses objetos, obtém-se sempre um número inteiro. O mesmo não é verdadeiro para os fractais. Com objetivo de mostrar o cálculo da dimensão fractal, serão utilizadas a Curva de von Koch, a Curva Quadrática de von Koch e o Triângulo de Sierpinski.

Tabela 2 - Interpretação padrão da dimensão inteira das figuras em termos de auto-similaridade exata [PEI88].

 Reta	Objeto unidimensional, pode ser subdividido em pedaços auto-similares à uma proporção de $r = \frac{1}{N}$, onde, $N.r^1 = 1$.
 Plano	Objeto bidimensional, pode ser subdividido em pedaços auto-similares à uma proporção de $r = \frac{1}{\sqrt{N}}$, onde, $N.r^2 = 1$
 Cubo	Objeto tridimensional, pode ser subdividido em pedaços auto-similares à uma proporção de $r = \frac{1}{\sqrt[3]{N}}$, onde, $N.r^3 = 1$

A curva de von Koch, e sua construção representada nas Figuras 8 até 12, proposta por volta de 1904, por exemplo, possui dimensão fractal aproximada de 1,26. Em um estágio inicial, a imagem tem formato de triângulo. Através de um processo iterativo, de acordo com um processo particular, cada segmento de reta é substituído por  . Isso significa que cada reta está sendo dividida em 4 partes auto-similares, então $N = 4$ com uma proporção $r = 1/3$. Aplicando a fórmula para o cálculo da dimensão fractal da Curva de von Koch, tem-se o seguinte:

$$D = \frac{\log(N)}{\log(1/r)}$$

$$D = \frac{\log(4)}{\log(3)}$$

$$D = 1.261859..$$

Ao contrário das formas euclidianas, as formas fractais mostradas aqui, apresentam detalhes em todas as escalas. Qualquer processo existente que amplie uma porção destas imagens, sempre mostrará, exatamente, os mesmos detalhes infinitamente [PEI88].

Figura 8 - Iteração zero da construção da curva de von Koch (Floco de Neve).

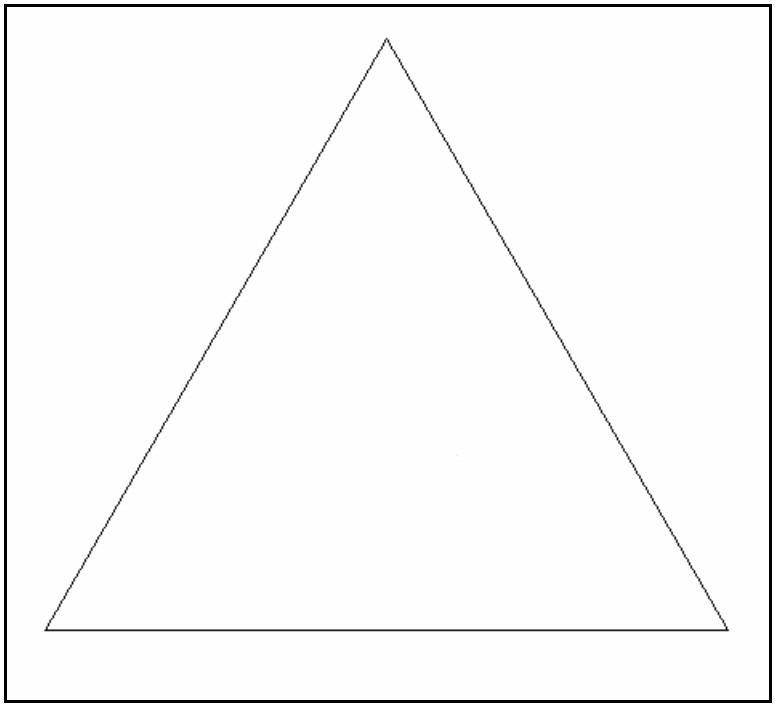


Figura 9 - Primeira iteração da construção da curva de von Koch.

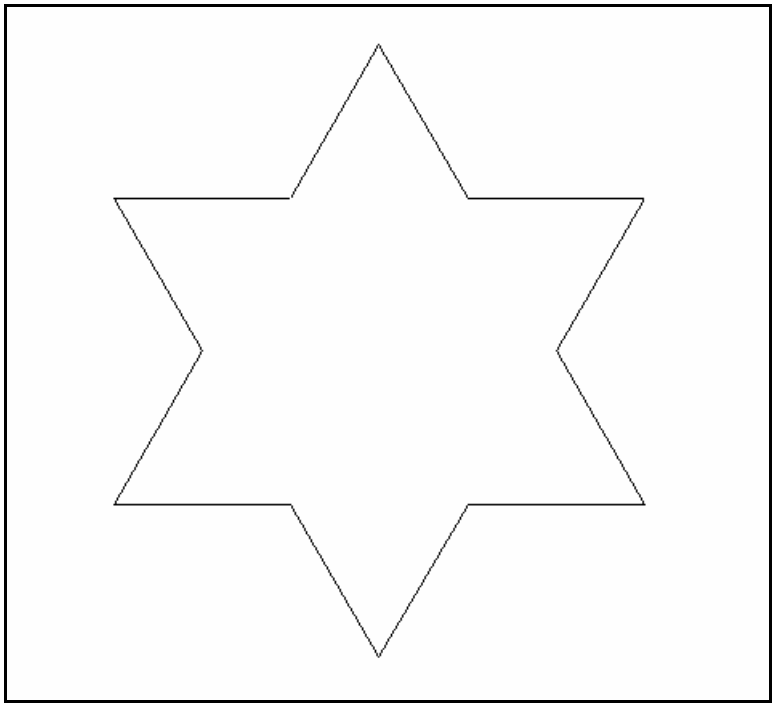


Figura 10 - Segunda iteração da construção da curva de von Koch.

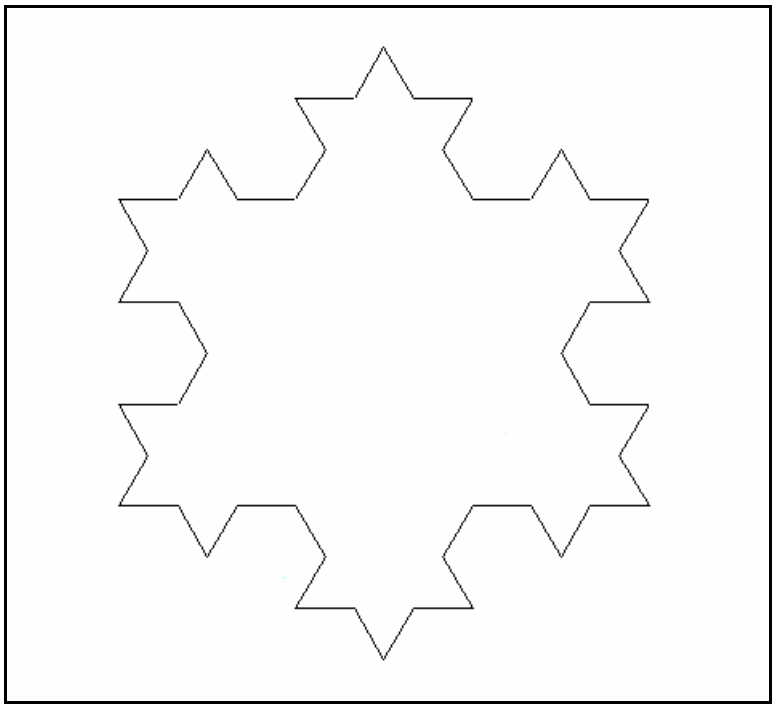


Figura 11 - Terceira iteração da construção da curva de von Koch.

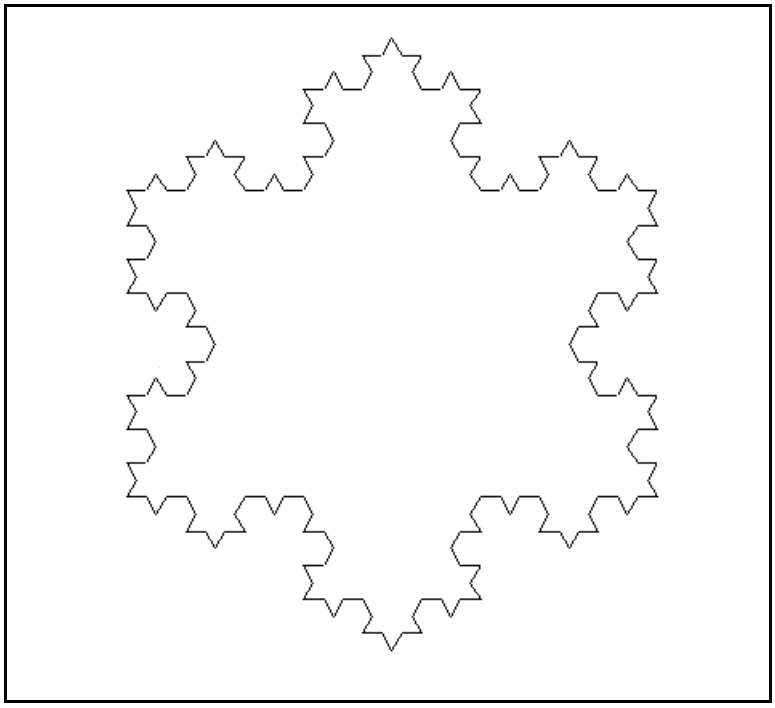
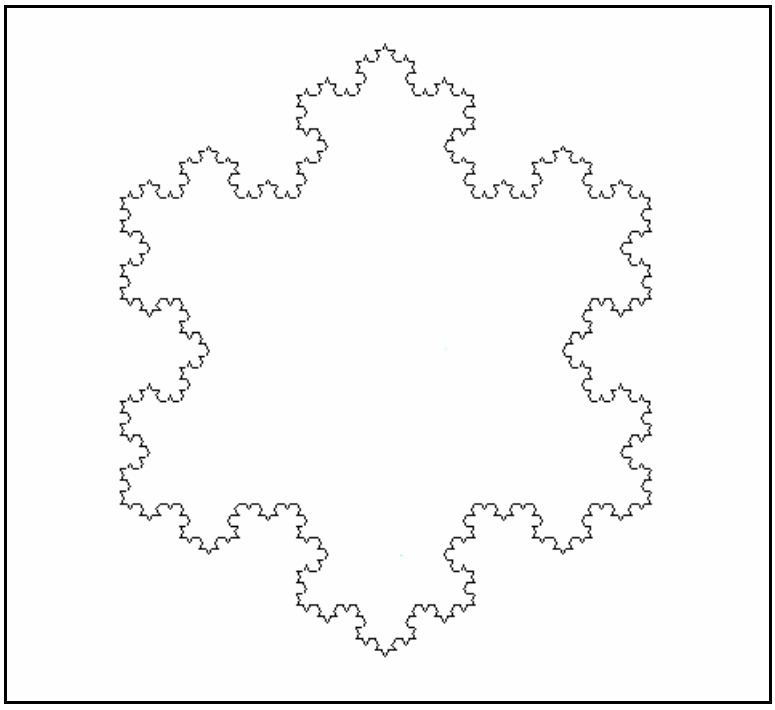


Figura 12 - Quarta iteração da construção da curva de von Koch.



As Figura 13 até 16 representam a construção de uma extensão da curva de von Koch com dimensão fractal alterada, conhecida como Curva de Quadrática de von Koch. Neste caso cada segmento de reta é substituído por  em cada iteração. A reta é quebrada em 8

partes auto-similares, então $N = 8$ e com proporção $r = 1/4$. Então essa imagem teria a seguinte dimensão fractal calculada:

$$D = \frac{\log(N)}{\log(1/r)}$$

$$D = \frac{\log(8)}{\log(4)}$$

$$D = 1.5$$

Outro detalhe importante dessas estruturas fractais é a alta-similaridade. Cada porção da imagem quando ampliada demonstra exatamente as mesmas caracterísicas da imagem inteira. Essas curvas fractais também são invariantes sob escalas diferentes [PEI88]. Crescem infinitamente em um plano finito.

Figura 13 - Iteração zero da construção da Curva Quadrática de von Koch.

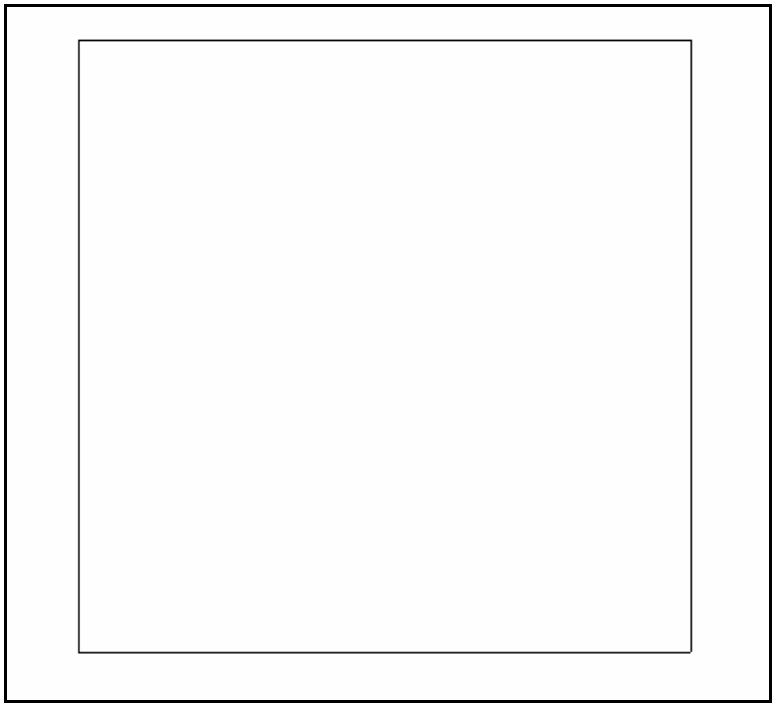


Figura 14 - Segunda iteração da construção da Curva Quadrática de von Koch.

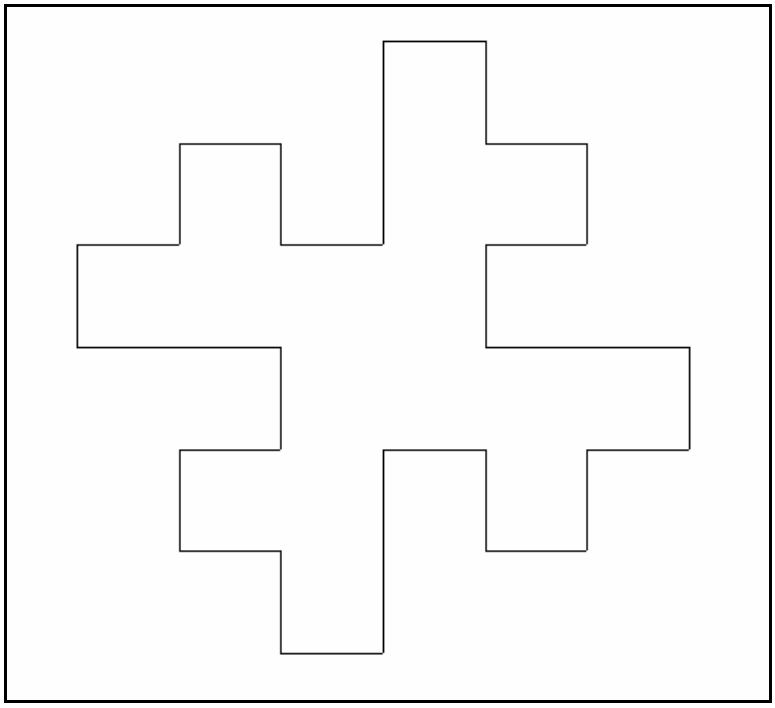


Figura 15 - Terceira iteração da construção da Curva Quadrática de von Koch.

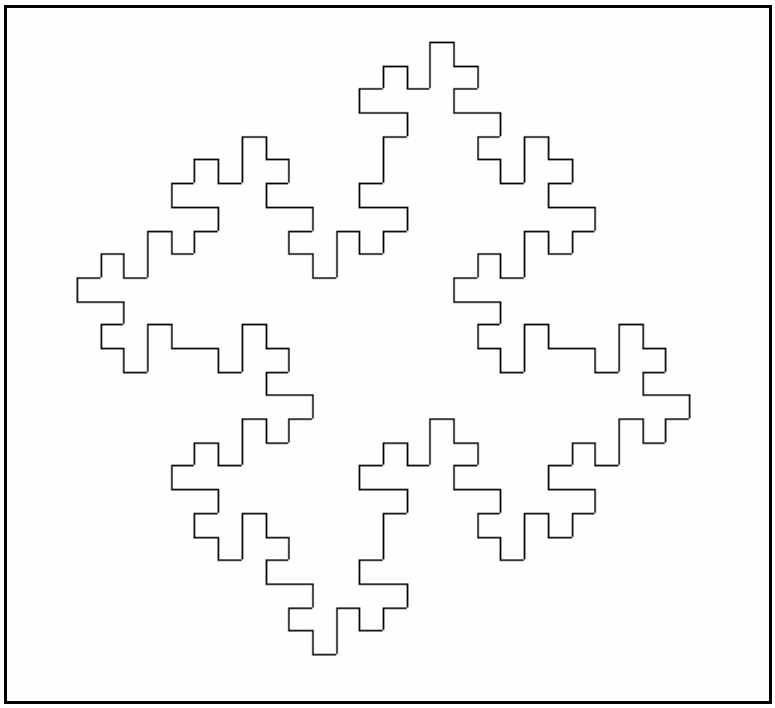
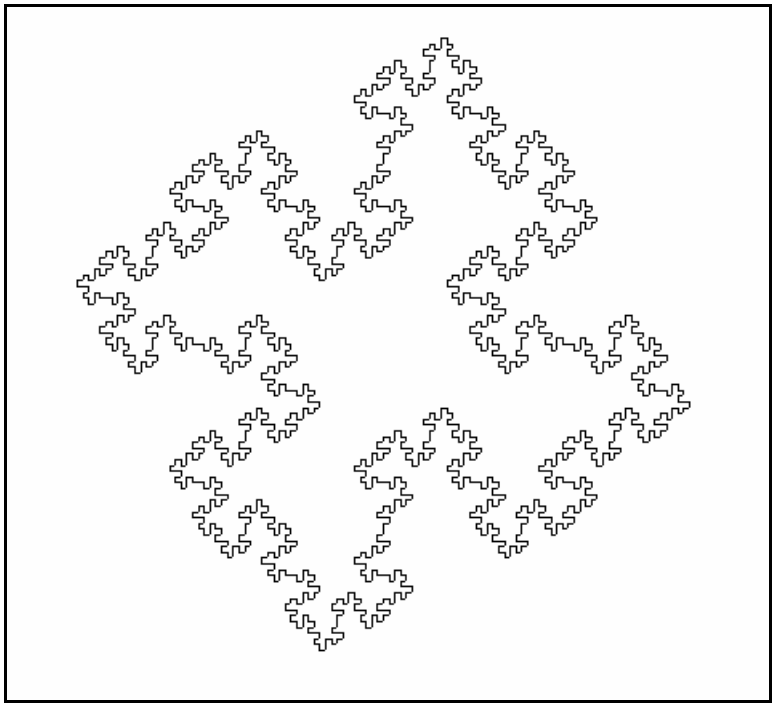


Figura 16 - Quarta iteração da construção da Curva Quadrática de von Koch



A construção do Triângulo de Sierpinski está representado pelas Figura 17 até 24. Observe que cada iteração divide os lados do triângulo ao meio, formando 3 triângulos auto-similares (o triângulo central não faz parte, é um "buraco"). Então, $N = 3$ e proporção $r = 1/2$. Calculando a dimensão fractal do Triângulo de Sierpinski teríamos:

$$D = \frac{\log(N)}{\log(1/r)}$$

$$D = \frac{\log(3)}{\log(2)}$$

$$D = 1.5849...$$

Figura 17 - Iteração zero da construção do Triângulo de Sierpinski.

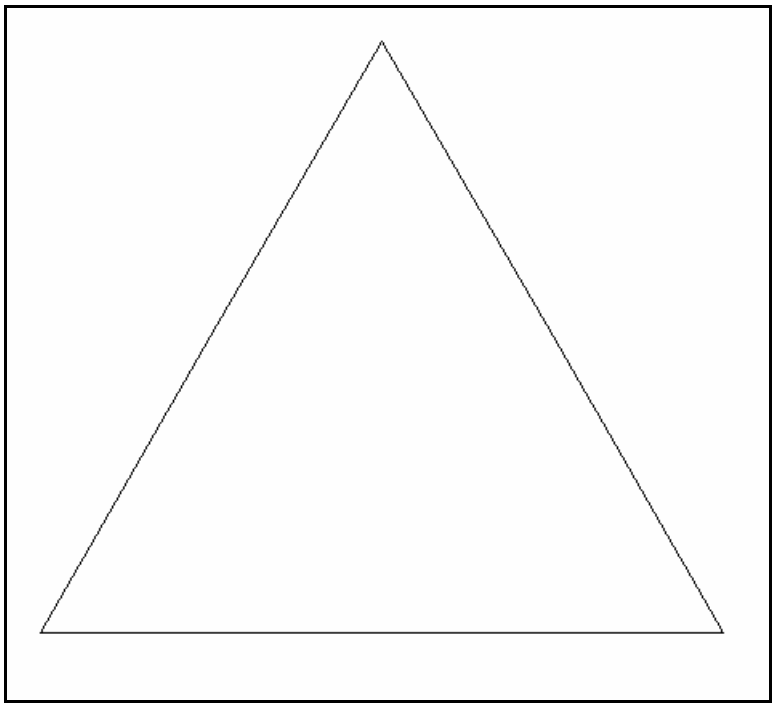


Figura 18 - Primeira iteração da construção do Triângulo de Sierpinski.

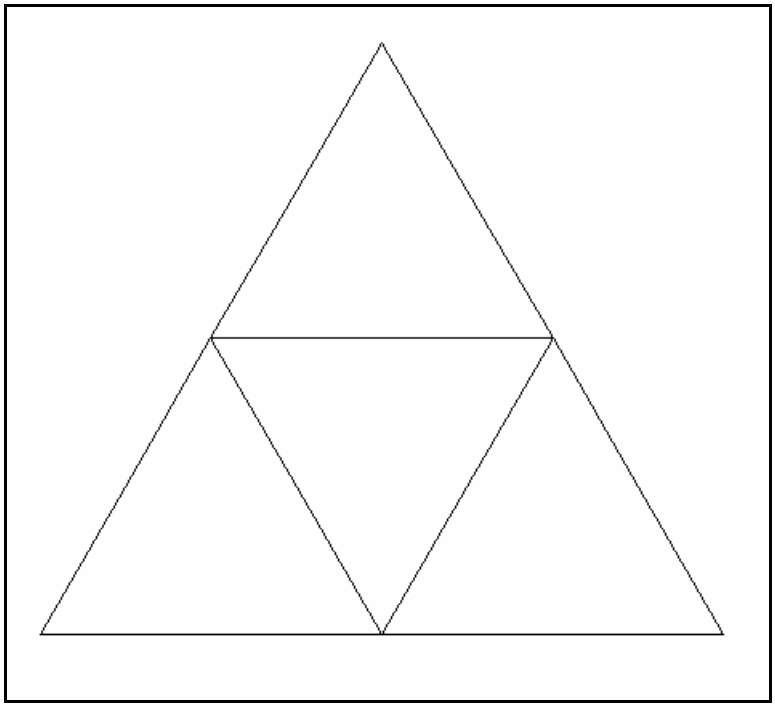


Figura 19 - Segunda iteração da construção do Triângulo de Sierpinski.

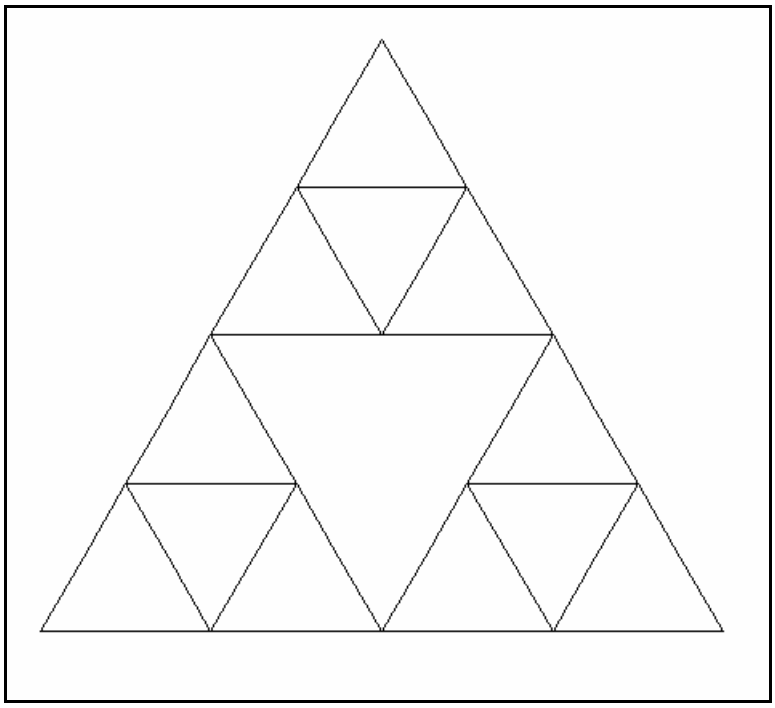


Figura 20 - Terceira iteração da construção do Triângulo de Sierpinski.

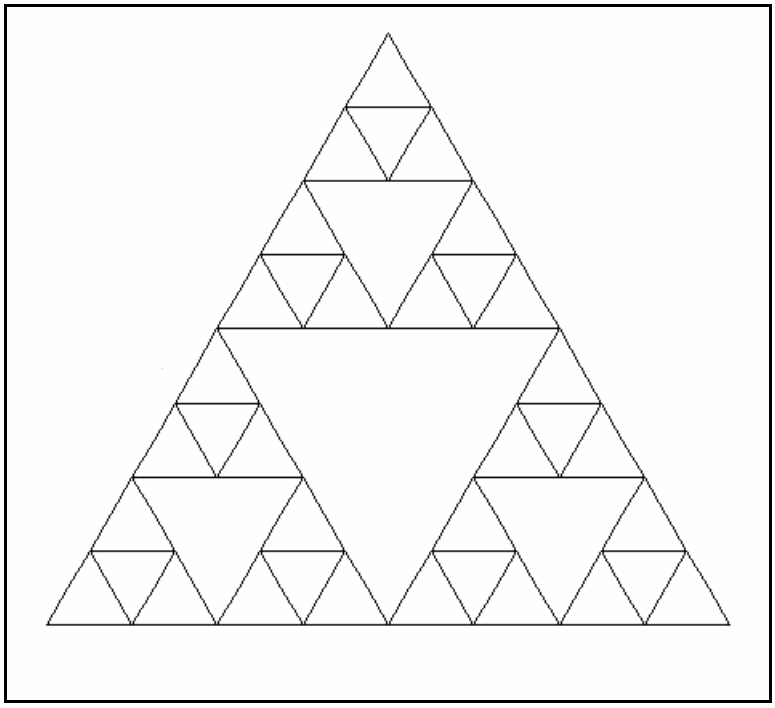


Figura 21 - Quarta iteração da construção do Triângulo de Sierpinski.

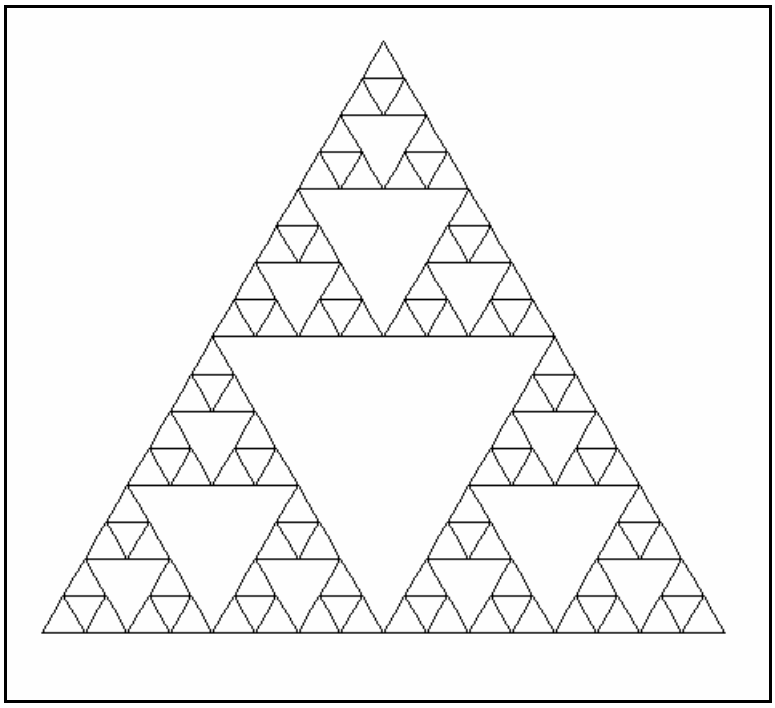


Figura 22 - Quinta iteração da construção do Triângulo de Sierpinski.

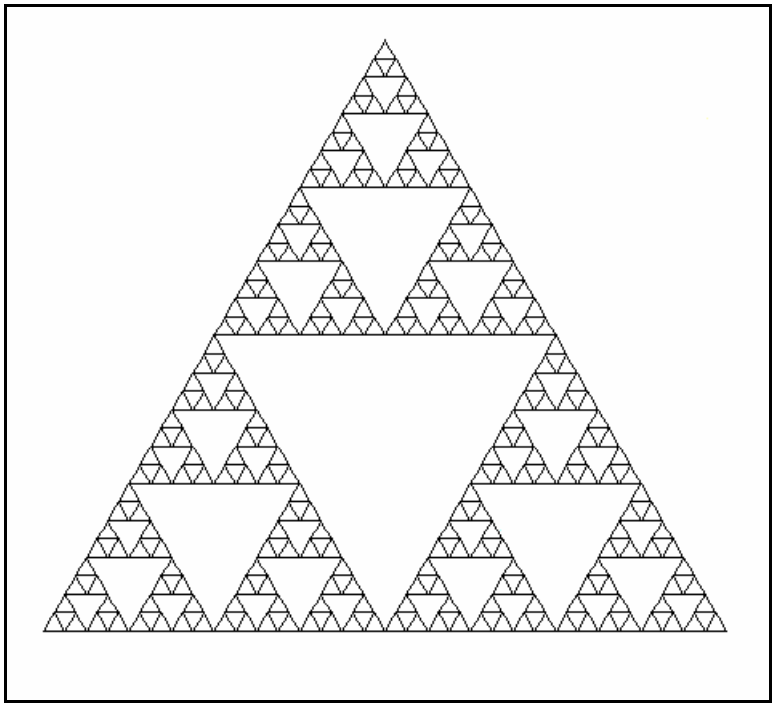


Figura 23 - Sexta iteração da construção do Triângulo de Sierpinski.

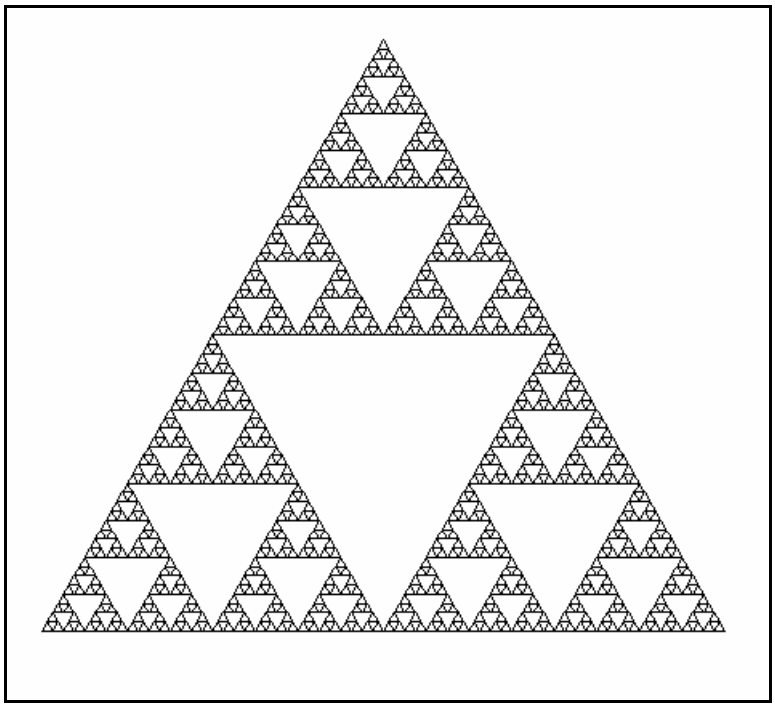
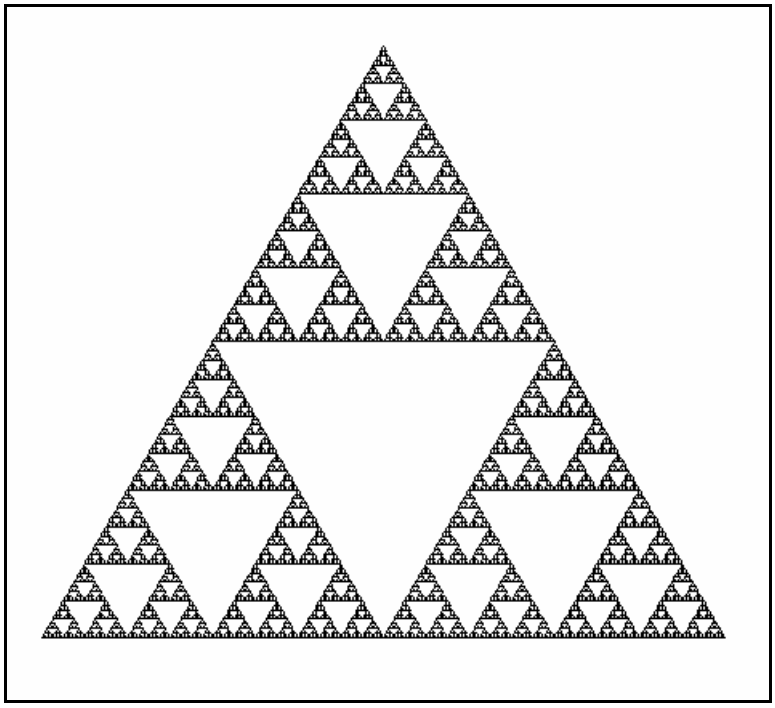


Figura 24 - Sétima iteração da construção do Triângulo de Sierpinski.



Este método de medir a dimensão fractal é válida somente para fractais que apresentam auto-similaridade exata. Através da dimensão fractal, os matemáticos são capazes

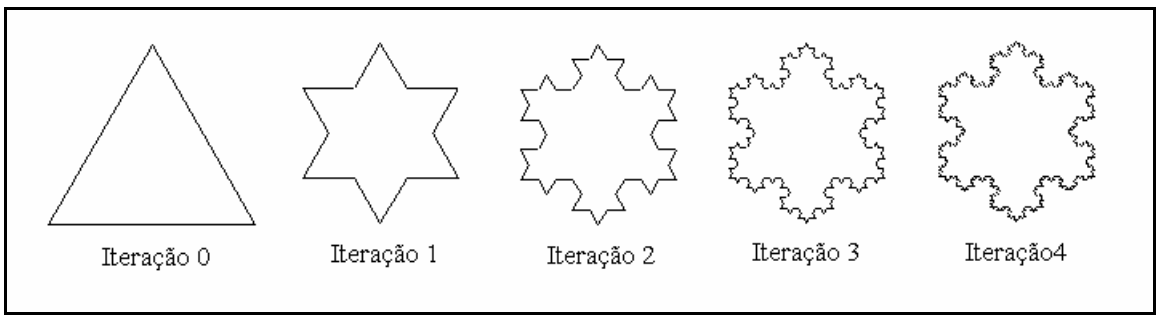
de medir formas que antes eram imensuráveis tais como montanhas, nuvens, árvores e flores. Existem, ainda, métodos conhecidos como *Mass, Box, Compass* [PEI88] entre outros. A dimensão fractal indica o nível de detalhe ou complexidade da rugosidade dos fractais e quanto espaço um fractal ocupa entre as dimensões euclidianas.

2.3.3 ITERAÇÃO

O principal ingrediente na formulação matemática de exemplos vindos da ecologia foi a iteração [PEI88]. No jargão dos matemáticos, iteração significa repetição de um processo inúmeras vezes. É um processo muito simples, que pode ser explicado utilizando o exemplo de uma calculadora científica contendo as funções x^2, $\sqrt{x}$, $sin\ x$, $\exp x$ e assim por diante. Cada uma destas teclas representam uma função matemática. Ao entrar com um valor particular na calculadora e pressionar a tecla $\sqrt{x}$ várias vezes, está ocorrendo uma iteração da função raiz quadrada. Em cada iteração da raiz quadrada utiliza-se o valor da iteração mediatamente anterior.

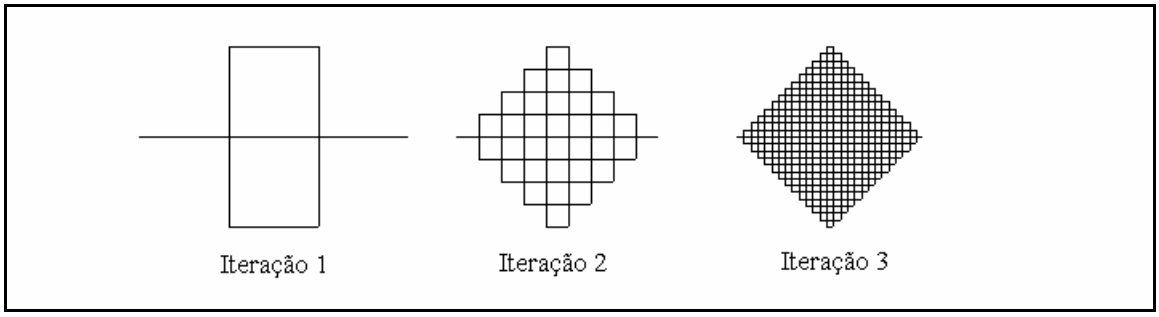
A Figura 25 demonstra em 4 iterações a formação do floco de neve de von Kock. A imagem é formada partindo de um triângulo normal, onde cada lado do triângulo (seguimento de reta) tem sua parte central substituída por dois seguimentos de $1/3$ de comprimento do seguimento original. O processo sofre iterações um número de vezes e forma o que é chamado de Floco de Neve de von Kock, uma extensão da Curva de von Kock.

Figura 25 – 5 iterações na construção da Curva de von Koch (Floco de Neve).



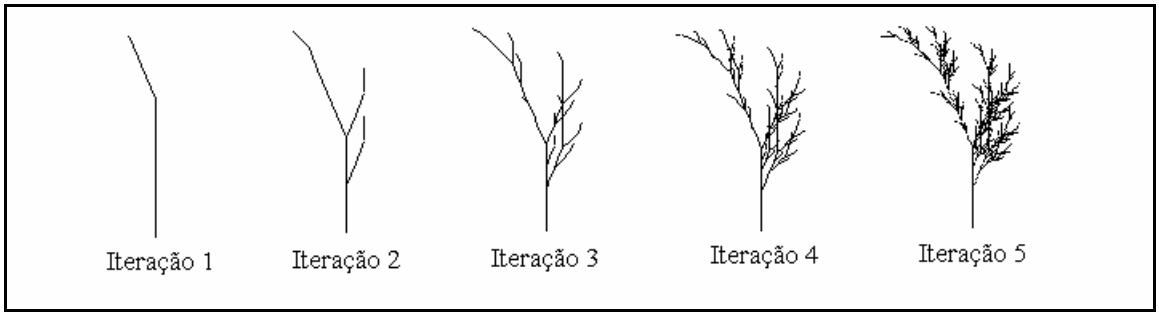
Outra variação da Curva de von Koch está representada na Figura 26 e suas iterações. Nesta construção cada seguimento de reta é substituído por outro seguimento de reta que representa $1/3$ do seguimento anterior.

Figura 26 – 3 iterações da construção da Curva de Hilbert, uma extensão da Curva de von Koch.



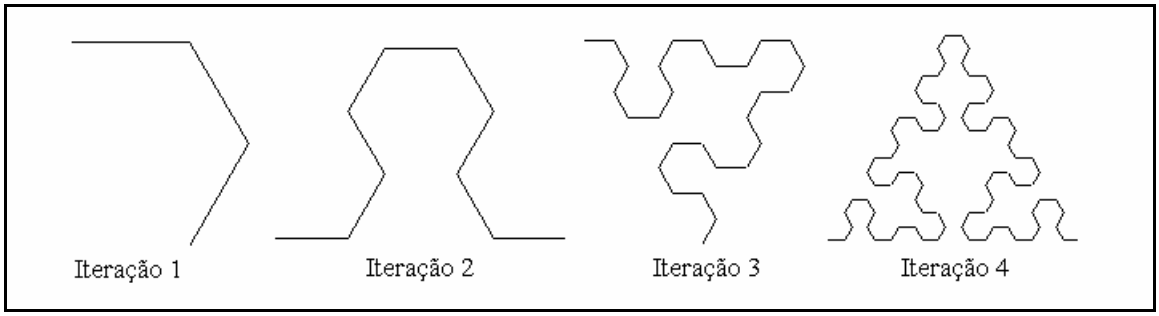
A Figura 27 mostra uma seqüência de iterações que nos lembra ramificação de alguma planta.

Figura 27 – 5 iterações de uma estrutura fractal que lembra ramificação de uma planta.



A Figura 28 mostra 4 iterações que resultam o Triângulo de Sierpinski modificado.

Figura 28 – 4 iterações da construção de uma extensão do Triângulo de Sierpinski.



Iterações de regras muito simples podem gerar formas complexas com propriedades desconhecidas. Ao contrário das formas Euclidianas, essas formas fractais possuem detalhes em todas as escalas, independentemente do fator de ampliação [PEI88].

Geralmente, os objetos fractais são frutos de processos iterativos, mas nem todos os processos iterativos resultam fractais [LAN99]. Todas as imagens fractais mostradas até aqui resultaram de um processo iterativo de acordo com uma regra particular. Uma das grandes barreiras no processo de iteração é o estabelecimento das regras necessárias para produzir um dado objeto com determinadas propriedades [PEI88].

Deve ficar salientado aqui que, há uma grande diferença entre iteração e interação. Interação é uma ação que se exerce mutuamente entre duas ou mais coisas, ou duas ou mais pessoas, etc. Iteração é a repetição contínua de um processo.

2.4 PESSOAS ENVOLVIDAS NO ESTUDO DOS FRACTAIS

A palavra fractal foi criada por Benoit Mandelbrot em 1970. Mandelbrot marca a origem da "*Geometria Fractal*" em 1975 mas indica que objetos hoje considerados fractais já existiam há muito tempo. Tais objetos foram criados por importantes matemáticos da história e abandonados justamente por fugirem dos conceitos da época. Alguns nomes são citados abaixo.

Albrecht Dürer (1471-1528) era mais um artista plástico e matemático. Foi quem criou os objetos fractais baseados em pentágonos regulares.

Dutchman Maurits Escher (1902-1972) criou texturas para objetos planos com propriedades fractais.

Georg Cantor (1845-1918). Como no final do século XIX estava sendo desenvolvido novas teorias que envolviam conjuntos, muitos matemáticos aproveitaram para criar conjuntos com propriedades estranhas. Muitos desses conjuntos criados, hoje são conhecidos como fractais. Cantor foi quem mais destacou-se.

Giuseppe Peano (1858-1932) escandalizou o senso comum dos matemáticos quando mostrou curvas contínuas de preenchimento de espaço.

David Hilbert (1862-1943) desenvolveu construções similares as de Peano.

Helge von Koch (1870-1924) propôs a famosa curva de von Koch, é uma figura clássica de objeto fractal.

Waclaw Sierpinski (1882-1969) contribuiu para o estudo da teoria dos conjuntos e teoria dos números. Seu nome está em muitos objetos fractais criados em espaço bidimensional e tridimensional. Objetos como Sierpinski Arrowhead (ou Triangle or Gasket) e Sierpinski Carpet são baseados em 2D. Sierpinski Tetrahedron e Sponge são baseados em 3D [CRI91].

Fractais gerados através das teorias de Gaston Julia (1893-1978) e Pierre Fatou (1878-1929), a partir de 1918, são baseados em plano complexo, onde as Figuras 1 e 2 são exemplos [CRI91].

Benoit Mandelbrot é um matemático que resolve problemas geometricamente e não algebricamente. Sua paixão pela geometria fez com que torna-se a geometria fractal aceita pelas diversas ciências. Através de todas as teorias e propostas já mencionadas acima, e outras, Mandelbrot deu continuidade aos estudos, validou e empregou todos os conceitos. Seus métodos de cálculo e geração de imagens foram empregados desde cenas de filmes em Hollywood até na economia. É conhecido como o pai dos fractais. A geometria fractal foi publicamente conhecida quando foi lançado o seu livro "*The Fractal Geometry of Nature*" , o qual engloba toda história e pessoas envolvidas neste estudo e resultados de suas próprias pesquisas.

Neste trabalho não são apresentadas as teorias matemáticas através das quais foram geradas as imagens que são mostradas. É apresentado uma técnica, reescrita paralela, que pode reproduzir fielmente as formas utilizando Sistema-L ao invés de um complexo algoritmo matemático.

2.5 RESUMO

Todo processo que evolui com passar do tempo é chamado de Sistema Dinâmico. Devido ao fato dos Sistemas Dinâmicos apresentarem comportamento imprevisível, os matemáticos criaram a noção de Caos, que estuda quais variáveis de um determinado sistema apresentam sensibilidade às condições iniciais. Com isso determina-se se um Sistema Dinâmico é caótico se houver, pelo menos uma variável de comportamento incerto. Não é o número de variáveis envolvidas em sistema que determina se é caótico ou não.

Em 1975 as formas geométricas geradas por sistemas dinâmicos foram nomeadas de *fractal* por Benoit Mandelbrot, com isso matemáticos, cientistas naturais e cientistas da computação uniram-se em caráter interdisciplinar com o objetivo de tornar a ciência mais unívoca com a utilização de métodos matemáticos inéditos. Foi devido à natureza apresentar formas e processos que eram praticamente impossíveis de serem representados através de processos clássicos, que os fractais foram criados, libertando a matemática dos limites impostos por Euclides e Pitágoras.

Pode-se dizer que fractais são formas que possuem auto-similaridade, possui dimensão fracionária, são irregulares, não são representados por fórmulas e geralmente são resultado de um processo iterativo. A própria natureza foi quem inspirou a criação de novos métodos matemáticos para que fosse possível a sua representação aproximada do que realmente é e como se comporta.

Pessoas como Dürer, Escher, Cantor, Peano, Hilbert, von Koch, Sierpinski, Julia, Fatou foram responsáveis, sem saber, pela criação da geometria fracionária. Mandelbrot foi quem validou e nomeou tudo o que foi criado e abandonado, pois ninguém mais tinha, até então, encontrado uma aplicação. Com isso, Mandelbrot, uma matemático com mania de geometria, tornou-se o pai dos fractais com sua obra “*The Fractal Geometry of Nature*”.

3 SISTEMAS DINÂMICOS SIMBÓLICOS, REESCRITA, SISTEMAS-L E INTERPRETAÇÃO GRÁFICA

Este capítulo apresenta um pouco mais sobre Sistemas Dinâmicos Simbólicos, que são os que se baseiam em caracteres para representar um determinado processo. Apresenta também a origem dos Sistemas-L e os elementos que o formam e formalismo para sua representação. Dois tipos de Sistemas-L serão apresentados, que são: Sistemas-LD0 e Sistemas-L Estocásticos e a característica de se gerar agrupamentos e ramificações através da interpretação gráfica dos caracteres produzidos na reescrita.

3.6 SISTEMAS DINÂMICOS SIMBÓLICOS

A dinâmica de qualquer processo refere-se a como o processo muda e adquire propriedades com o decorrer do tempo. Um sistema dinâmico é uma configuração física e regras que ditam como essa configuração muda ou evolui de um momento para outro. Um dos principais objetivos da teoria matemática dos sistemas dinâmicos é determinar ou caracterizar o comportamento à longo tempo do sistema [WRI96]. Entende-se aqui por sistema, qualquer processo, natural ou artificial, onde sistemas naturais estão presentes na natureza e evoluem sem que haja intervenção humana. Sistemas artificiais são representados, principalmente por fórmulas matemáticas.

Sistemas dinâmicos simbólicos são métodos da matemática que utilizam cadeia de símbolos formais para representar uma configuração física de uma determinada dinâmica. A evolução de um determinado sistema baseado em símbolos formais, utilizado por Aristid Lindenmayer, é governado através de um processo conhecido como reescrtia [WRI96].

3.7 REESCRITA E SISTEMAS-L

A reescrita é o conceito central dos Sistemas de Lindenmayer [PRU90]. De modo geral, a reescrita é uma técnica para criação de objetos complexos através de substituições recursivas de partes simples de um objeto inicial utilizando o que se chama de regra de reescrita ou regra de produção [CUO97]. A reescrita está em muitos ramos da ciências da computação como: inteligência artificial, álgebra simbólica e projetos de linguagem de programação de alto nível [ROZ92].

Rozemberg diz que: “do ponto de vista da biologia, os Sistemas-L fornecem uma teoria útil através da qual o comportamento natural do desenvolvimento das células pode ser discutido, calculado e comparado...do ponto de vista matemático, os Sistemas-L abrem uma nova dimensão na teoria das linguagens formais...teóricos das linguagens formais e dos autômatos encontraram nos Sistemas-L uma alternativa proveitosa e interessante para a gramática Chomsky” [ROZ92].

A primeira definição dos sistemas de reescrita foi dada no início deste século, mas o maior interesse foi semeado por Chomsky no início dos anos 50 com o seu trabalho sobre gramática formal. Chomsky utilizou o conceito de reescrita para descrever as características sintáticas de linguagens naturais. Poucos anos mais tarde Backus e Naur introduziram uma noção baseada em reescrita com o objetivo de melhorar uma definição formal da linguagem de programação ALGOL-60. Logo, a equivalência entre os dois tipos de gramáticas propostas, Backus-Naur e Chomsky, foram reconhecidas e iniciou-se um período de aplicação às ciências da computação. Os elementos envolvidos nos estudos eram cadeias de caracteres formais, métodos para geração, reconhecimento e transformação destes caracteres.

Em 1986, um botânico alemão chamado Aristid Lindenmayer, criou um novo tipo de reescrita, nomeado de Sistema-L, resumo de Sistema de Lindenmayer, também chamado de Sistema de Reescrita Paralela. A grande diferença entre o método proposto por Chomsky e Lindenmayer está no método de aplicação das produções [OCH98]. Na gramática de Chomsky, que não é utilizada para modelagem gráfica, as produções são aplicadas seqüencialmente, ao passo que nos Sistemas-L as produções são aplicadas paralelamente, substituindo simultaneamente todas as letras de uma da palavra. Esta diferença reflete a

motivação para a aplicação dos Sistemas-L na modelagem e estudo do desenvolvimento de organismos biológicos [PRU90].

Sistema-L é um tipo particular de Sistema Dinâmico Simbólico que compreende os seguintes componentes [WRI96].

Alfabeto: o alfabeto é um conjunto *V* finito de símbolos formais, geralmente caracteres como A, F, X, Y, etc. O conjunto *V* consiste de todas as letras que ocorrem no axioma e nas produções.

Axioma: também chamado de iniciador, é uma cadeia *W* de caracteres formais de *V*. O conjunto de cadeias de caracteres, também chamado de palavra, é denotado por *V**. Dado que V={a,b,c}, alguns exemplos de palavra poderiam ser: *aabca*, *caab*, *bbbc*, etc. O comprimento |*W*| de uma palavra *W* é o número de símbolos da palavra. Também é possível o mapeamento de produções de *a* para uma palavra vazia, denotada por ϕ, ou para o próprio *a*. Se um símbolo $a \in V$ não possui uma produção, assumi-se, por definição, que seja mapeado em si mesmo. Neste caso *a* é uma constante do Sistema-L.

Produções: uma produção ou regra de reescrita é um mapeamento de um símbolo $a \in V$ em uma palavra $W \in V^*$. Aqui as produções terão o seguinte formalismo:

$$p_n : a \rightarrow W, \text{ onde:}$$

- *p* significa produção;
- *n* é o número da regra de produção;
- *a* é chamado de antecessor;
- *W* é chamado de sucessor.

Quando uma produção é aplicada à uma palavra, especificamente a um axioma, o antecessor é comparado com todos os símbolos desta palavra. O sucessor substitui o símbolo da palavra que for igual ao seu antecessor. Neste processo de reescrita, percebe-se que o axioma cresce em tamanho, quantidade de caracteres, e todos os padrões do início são preservados de algum modo. O número de vezes (quantas vezes é iterado) que um axioma é reescrito é chamado de nível recursivo ou ordem. Simples axiomas e regras produzem, através do processo de substituição, objetos cada vez mais complexos. No caso de modelagem de

plantas, a reescrita pode alcançar um grau de complexidade maior que o próprio processo natural que o governa [LIN96].

Com a seguinte definição Sistema-L e nível de iteração igual a 3, pode-se observar o processo de substituição dos caracteres e como o axioma aumenta de tamanho de uma iteração para outra.

Axioma	**F++F++F**
Regras	p_1: **F→F-F++F-F**

Nível recursivo 0: o nível recursivo 0 sempre será o próprio axioma e é representado por:

F++F++F

Nível recursivo 1: percorre-se o axioma em busca de um símbolo que seja igual ao predecessor (caracter que está a esquerda do símbolo →) da regra p_1. Percebe-se que há três caracteres, **F**, no axioma que deve ser substituído, em paralelo, pelo sucessor da regra p_1. Os caracteres do axioma que não possuem um antecessor correspondente à uma regra do conjunto de regras são chamados de constantes e permanecem como estão: **+→+**, **- →-**, etc. Então cada **F** será substituído por **F++F++F**, cada **+** será substituído por **+** e cada **-** será substituído por **-** , obtendo o seguinte resultado após o a primeira iteração:

F-F++F-F++F-F++F-F++F-F++F-F

Nível recursivo 2: percorre o resultado da 1ª iteração e faz novas substituições, apresentando o seguinte resultado:

F-F++F-F-F-F++F-F++F-F++F-F-F-F++F-F++F-F++F-F-F-F++F-
F++F-F++F-F-F-F++F-F++F-F++F-F-F-F++F-F++F-F++F-F-F-F++F-F

Nível recursivo 3: percorre o resultado da 2ª iteração, fazendo novas substituições, apresentado o seguinte conjunto de símbolos:

F-F++F-F-F-F++F-F++F-F++F-F-F-F++F-F-F-F++F-F-F-F++F-
F++F-F++F-F-F-F++F-F++F-F++F-F-F-F++F-F++F-F++F-F-F-F++F-F-F-
F++F-F-F-F++F-F++F-F++F-F-F-F++F-F++F-F++F-F-F-F++F-F++F-
F++F-F-F-F++F-F-F-F++F-F-F-F++F-F++F-F++F-F-F-F++F-F++F-F++F-
F-F-F++F-F++F-F++F-F-F-F++F-F-F-F++F-F-F-F++F-F++F-F++F-F-F-
F++F-F++F-F++F-F-F-F++F-F++F-F++F-F-F-F++F-F-F-F++F-F-F-F++F-
F++F-F++F-F-F-F++F-F++F-F++F-F-F-F++F-F++F-F++F-F-F-F++F-F-F-
F++F-F-F-F++F-F++F-F++F-F-F-F++F-F

A substituição ocorre até o nível recursivo que for informado, apresentado em cada iteração, uma configuração mais complexa, mantendo sempre as características iniciais. A definição mostrada acima como exemplo é a definição para a construção da Curva de von Kock, conhecida como Floco de Neve (Figura 12).

Uma das principais características e umas das vantagens dos Sistemas-L é a capacidade de representar objetos complexos partindo de objetos extremamente simples (*amplificação de dados*). Isso pode ser percebido através do exemplo acima [PRU90].

Para validar o novo método de reescrita paralela proposto por Aristid Lindenmayer, necessitava-se de um modo de interpretar o resultado do processo. Esse conjunto de caracteres resultante deveriam ser interpretados, um a um, graficamente e através desta interpretação, gerar uma imagem.

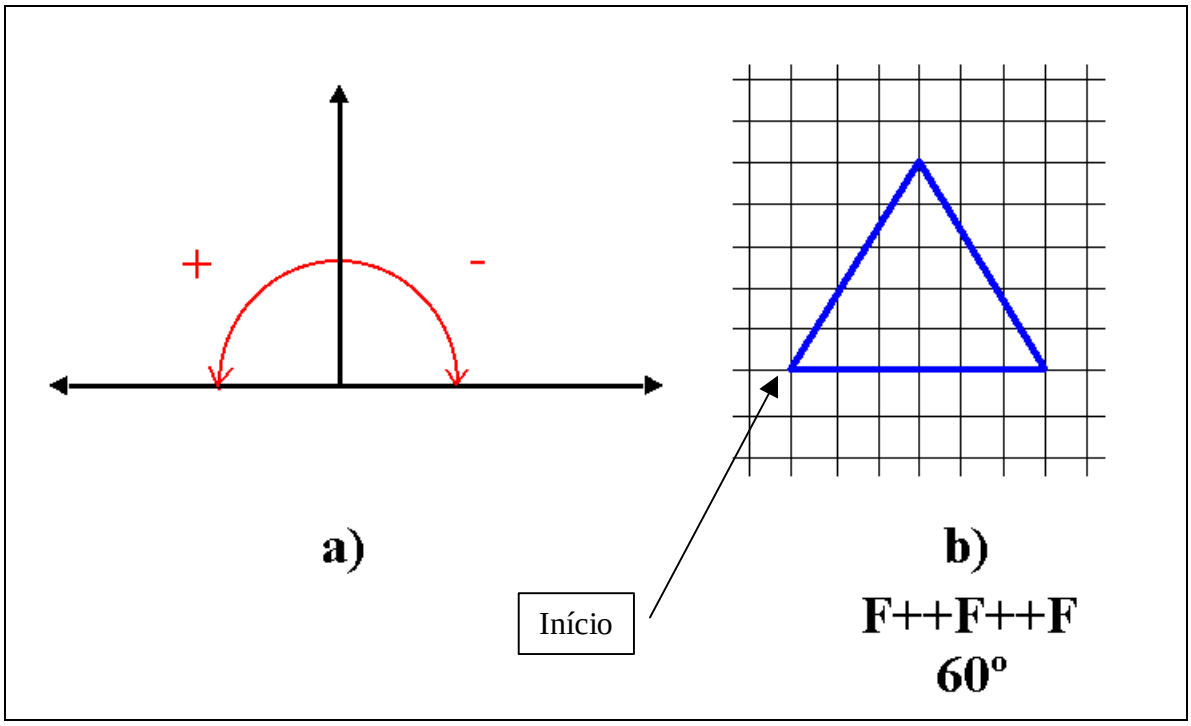
3.8 INTERPRETAÇÃO GRÁFICA DOS CARACTERES

Várias propostas de interpretação gráfica para os Sistemas-L foram apresentadas [PRU90]. Prusinkiewicz propôs uma interpretação gráfica baseada na tartaruga LOGO e apresentou grandes melhoras na geração de imagens fractais e modelagem de plantas. A interpretação tartaruga foi utilizada na modelagem de plantas herbáceas, descrição de padrões *Kolam* (uma arte do Sul da Índia), síntese de partitura musical e geração de curvas de preenchimento.

LOGO é uma linguagem de programação que foi criada com o objetivo de ensinar matemática e geometria a iniciantes, inclusive crianças. Foi desenvolvido por Daniel Bobrow e Wallace Feurzeig em Bolt, Beranek e Newman Inc. e Seymour Papert em Massachusetts Institute of Technology nos anos 60.

A idéia básica da interpretação gráfica da tartaruga é dada na Figura 29. Um estado inicial da tartaruga é definido pelo terceto (x, y, α), onde as coordenadas cartesianas (x, y) representam a posição da tartaruga e o ângulo α é interpretado como a direção da tartaruga.

Figura 29 – Interpretação gráfica da tartaruga para a cadeia de caracteres F++F++F. Iteração zero para a formação da Curva de von Koch (Floco de Neve).



Dado um comprimento d e um incremento angular δ, a tartaruga pode responder a comandos representados na Tabela 3.

Tabela 3 - Relação dos Caracteres que são interpretados pela tartaruga gráfica [PRU90].

Caracter	Significado
F	Move um passo para frente com distância *d*. O estado da tartaruga muda para (*x'*, *y'*, α), onde : $x' = x + d\cos\alpha$ e $y' = y + d\sin\alpha$. Um Seguimento de linha é desenhado entre os pontos (*x*, *y*) e (*x'*, *y'*).
f	Move um passo para frente com distância *d* sem desenhar um seguimento de reta.
+	Vira a esquerda, sentido anti-horário, um ângulo α. O próximo estado da tartaruga será ($x, y, \alpha + \delta$).
-	Vira à direita, sentido horário, um ângulo α. O próximo estado da tartaruga será ($x, y, \alpha - \delta$).
[	Empilha o estado atual da tartaruga. A informação empilhada contém a posição e a orientação da tartaruga.
]	Desempilha e torna atual a última informação empilhada.
|	Vira 180º utilizando a orientação corrente horária (+) ou anti-horária (-).
0..9	Cada valor de 0 até 9 é uma cor, sendo que os seis primeiros valores são tons de verde.

Todos os símbolos da tabela possui significado para a interpretação gráfica. Qualquer outro caracter é ignorado. Não existe um consenso sobre quais símbolos devem ser utilizados na interpretação gráfica desde que se tenha resultados. Aqui a letra *f* (efe minúsculo) move a tartaruga sem desenhar a reta. Outros autores utilizam a letra *G*, por exemplo.

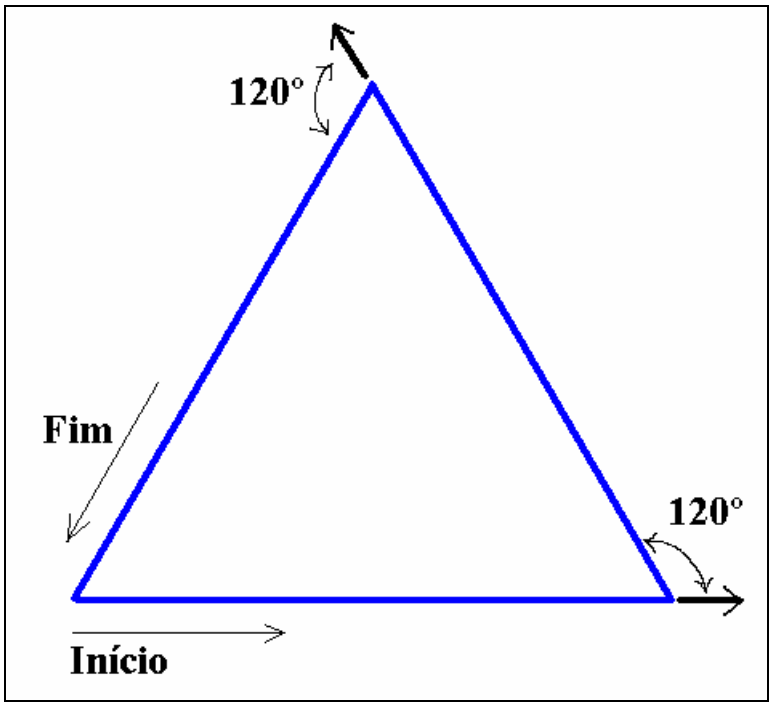
Tabela 4 – Extensão para a tartaruga gráfica em um ambiente 2D.

Caracter	Significado
@	Multiplica o comprimento atual da reta pela quantidade que segue esse caracter.
!	Inverte o significado dos caracteres +/-.
*****	Aumenta 10% o próximo passo.
/	Decrementa 10% o próximo passo.
'	Muda o comprimento do próximo passo randomicamente.
‘	Incrementa um índice de cor.
`	Decrementa um índice de cor.

Dado uma cadeia de caracteres *V*, o estado inicial da tartaruga (x_0, y_o, α_0) e parâmetros fixados *d* e δ, a interpretação gráfica de *V* é a figura (conjunto de linhas) desenhada pela tartaruga de acordo com a cadeia de caracteres *V* [PRU90].

Ao interpretar o cadeia de caracteres **F++F++F** a tartaruga gráfica encontra um **F** e desenha um seguimento de reta de comprimento *d.* Como o incremento de ângulo δ é 60º, gira à esquerda 60º, pois encontrou um caracter **+**. Gira mais 60º, pois tem dois **+** seguidos, somando 120º. Depois encontra **F** e desenha um seguimento de reta com comprimento *d.* Vira 120º novamente e desenha mais um seguimento de reta e finaliza a interpretação gráfica da cadeia de caracteres **F++F++F** e iteração zero ou o próprio axioma da definição da Curva de von Koch (Floco de Neve). Este processo está representado na Figura 30.

Figura 30 – Direções da tartaruga na interpretação gráfica da cadeia de caracteres F++F++F (axioma da definição do Floco de Neve de von Koch).



A tartaruga que interpreta a cadeia de caracteres em um Sistema-L deve aprender um vocabulário de comandos extenso, além de poder ser aplicada à interpretação de caracteres em três dimensões utilizando-se matrizes de transformações [BOW88], [HIL90], [PRU90].

Estruturas de dados mais complexas serão utilizadas e deve-se preocupar-se muito com performance da interpretação, pois necessita de vários cálculos [PEI88].

O problema maior da interpretação em 3D é visualização da imagem em 2D. Não é trivial encontrar ambientes que trabalham diretamente em 3D. É necessário que se utilize projeções ortogonais [BOW88] para possibilitar a visualização 3D em ambiente 2D. Para que seja possível ver a imagem em vários ângulos, é necessário que a imagem seja exportada para um formato que um ambiente 3D possa ler. O formato DXF (*Drawing Exchange Format*), por exemplo, é um formato suportado pelo Auto-CAD. Além disso, se for decidido aplicar técnicas de render, é necessário que trabalhe com faces. Imagens formadas por linhas, apenas apresentam formato aramado e não é possível renderizá-la. Técnicas de render são processos que deixam a imagem mais real com utilização de luz, sombras e outros efeitos.

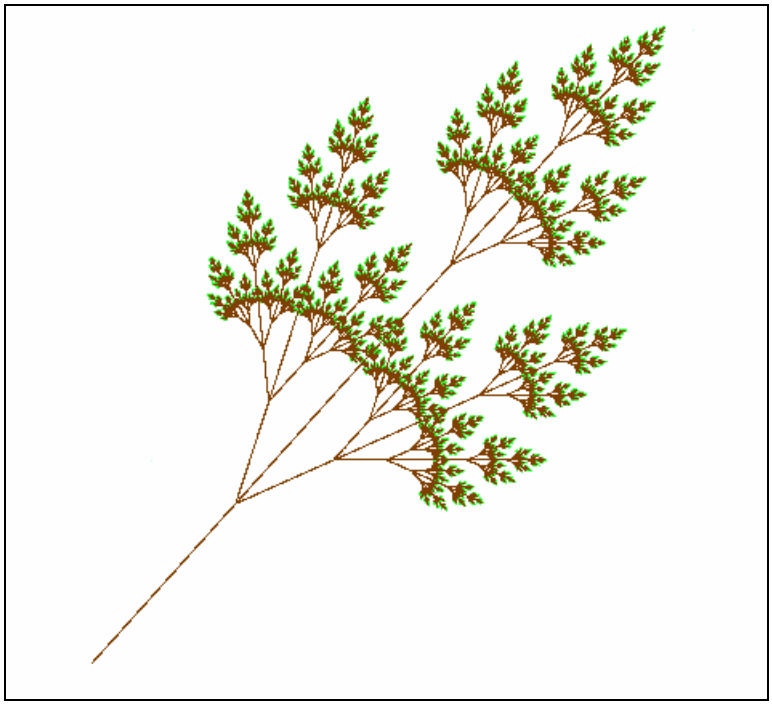
3.8.1 SISTEMAS-LD0

A classe mais simples de Sistemas-L são os Sistemas-LD0, onde a letra *D* significa Determinístico e o número Zero significa que não é sensitivo ao contexto ou é Contexto-Zero [PEI88].

Tais sistemas são determinísticos, devido ao fato de não possuir nenhum tipo de processamento especial na reescrita, ou seja, não têm nenhum procedimento de atribuição randômico ou estatístico.

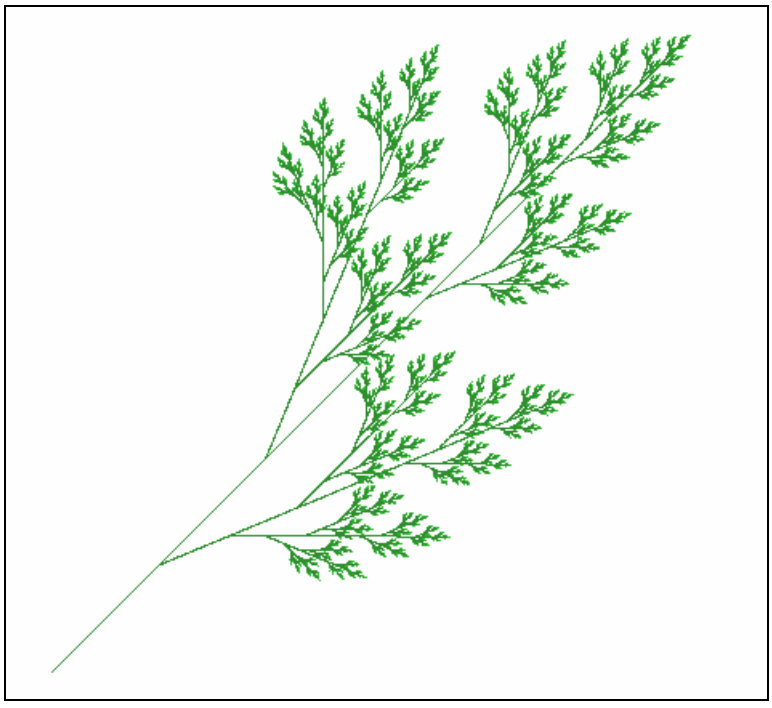
As Figura 31 e 32 são exemplos de estruturas fractais geradas a partir de reescrita paralela determinística. As definições [NEI99] Sistema-L seguem as figuras. Ambas as imagens são muito parecidas com padrões encontrados na natureza.

Figura 31 – Formação de uma estrutura determinística em 10 iterações.



Direções	**15**
Axioma	**++A**
Regras	p_1 : **A→3F[+A][-A]5FA** p_2 : **F→6FF**

Figura 32 – Formação determinística em 9 iterações.



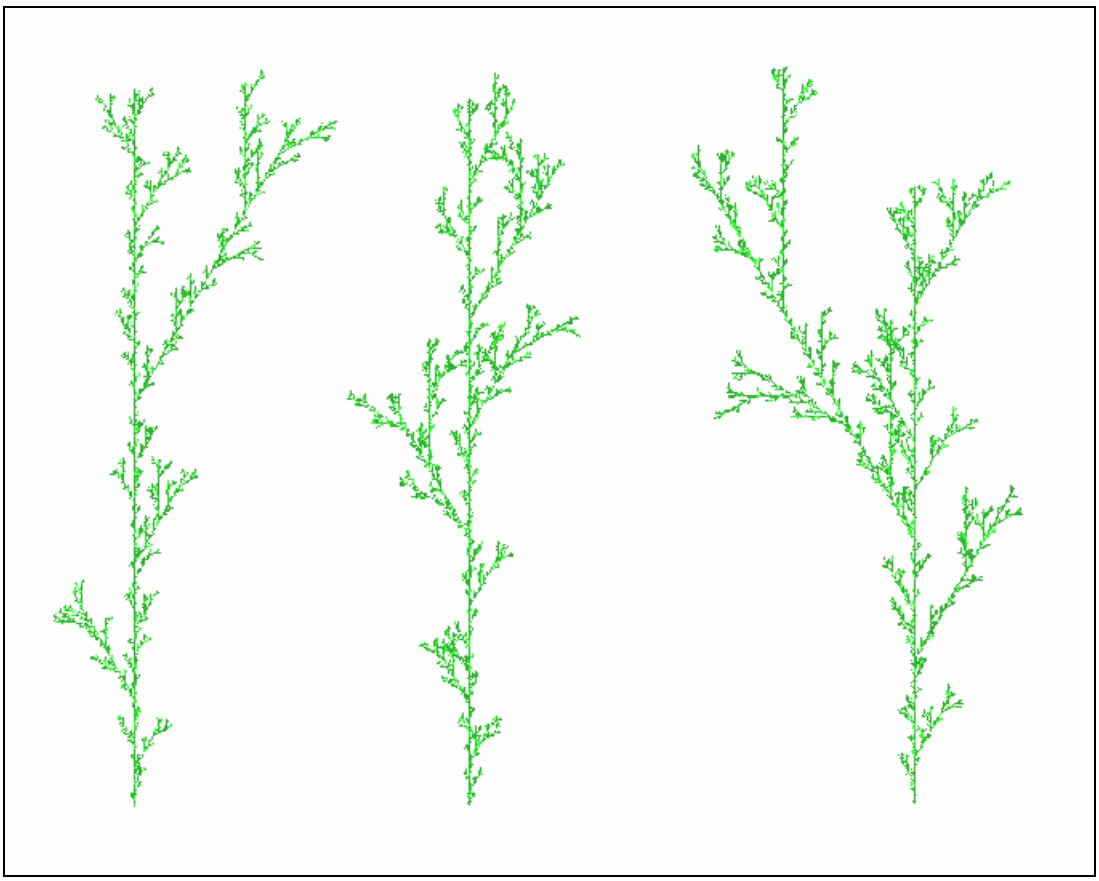
Direções	**32**
Axioma	**++++C**
Regras	p_1 : **C→3NF[--C]NF[++C]N-FF+C** p_2 : **N→1FNNF** p_3 : **P→C**

Sistemas-LD0 são utilizados na teoria da linguagem formal, nas ciências da computação e no estudo do processo de divisão e diferenciação celular, na biologia [ROZ92].

3.8.2 SISTEMAS-L ESTOCÁSTICOS

Todas os modelos gerados através de Sistema-LD0 são idênticos [PRU90]. Com o objetivo de produzir imagens diferenciáveis nas características individuais, mantendo as características gerais, os Sistemas-L Estocásticos abordam cálculos randômicos e estatística na reescrita. Figura 33 é um exemplo.

Figura 33 – Modelos estocásticos em 7 iterações.



A definição Sistema-L [PRU90] para a geração dos modelos da Figura 33 está definido logo à seguir. Os números que estão entre parênteses dizem a probabilidade, em percentual de ocorrer a regra. É importante observar que a soma dos percentuais devem ser igual a 100.

Direções	**12**
Axioma	**+++F**
Regras	p_1 : **F→(33)2F[+F]F[-F]F** p_2 : **F→(53)3F[+F]F** p_3 : **F→(14)F[-F]F**

O principal efeito de um Sistema-L estocástico é que diferentes imagens podem ser geradas com a mesma definição. A principal característica é que o axioma possui símbolos que correspondem a vários antecessores no conjunto de regras, ao passo que um Sistema-LD0 possui apenas uma regra cujo antecessor corresponde apenas a um símbolo do axioma.

Outros tipos de Sistemas-L são representados por:

a) Sistemas-L paramétricos: são utilizados para capturar detalhes específicos da arquitetura da planta;

b) Sistemas-L com fragmentação: são Sistemas-L que modelam efeitos tais como a propagação ou folhas caindo;

c) Sistemas-L sensitivo ao contexto: são Sistemas-L que consideram aspectos fisiológicos de plantas;

d) Sistemas-L sensitivo ao ambiente: são Sistemas-L que interagem com um ambiente criado. O ambiente fornece sais e minerais, água, luz e tipo de solo os quais são relevantes para a formação do modelo;

e) Sistemas-L diferenciais ou Sistemas-L temporais: são Sistemas-L utilizados para representar seqüências de crescimento de plantas ou células. Com isso é possível criar animações, podendo-se visualizar todos os estágios de crescimento de acordo com o tempo.

Os diversos tipos de Sistemas-L ainda podem se fundir para implementação de ambientes ainda mais complexos.

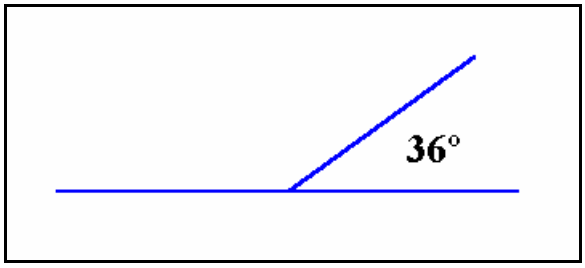
3.9 AGRUPAMENTOS E RAMIFICAÇÃO

Muitas estruturas biológicas são “agrupadas”, “fragmentadas” ou “celular” na aparência e crescimento [WRI96]. Para permitir que agrupamentos ocorram durante a interpretação gráfica, os símbolos “[“ (empilha os próximos movimentos) e “]” (desempilha os movimentos) devem ser usados. Isso permite que a interpretação gráfica da tartaruga realize uma ramificação por vez e volte à posição onde começou a ramificação. O exemplo abaixo mostra como funciona um agrupamento ou ramificação agrupada.

Direções	**10**
Axioma	**F**
Regra	p_1 : **F→F[+F]F**

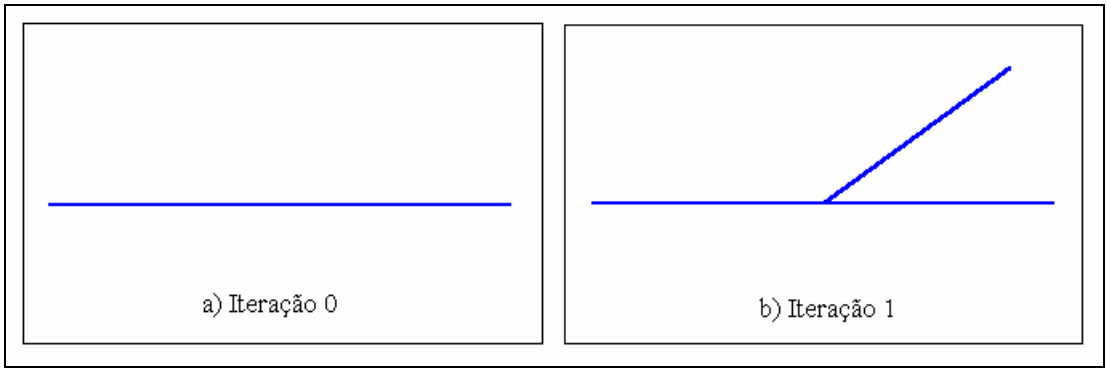
Podes-se entender através da definição acima, que cada seguimento de reta será substituído pela configuração mostrada na Figura 34.

Figura 34 – Regra de formação de um simples agrupamento.



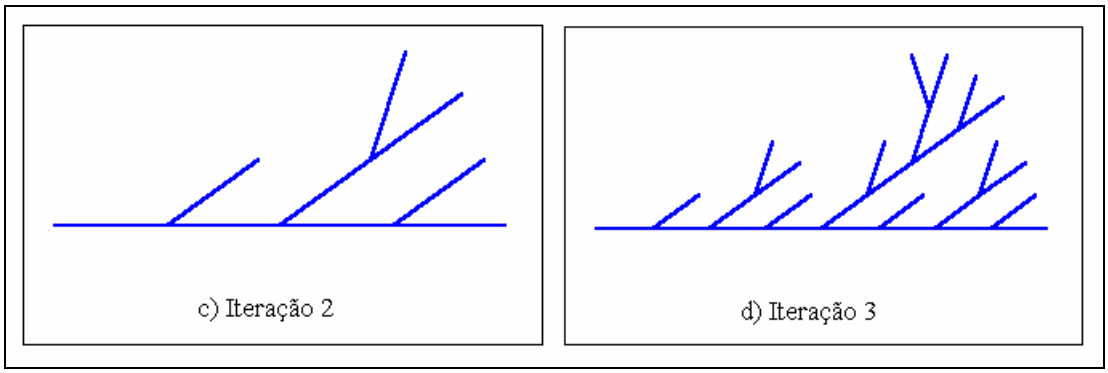
A Figura 35 mostra as iterações 0 e 1 da definição Sistema-L dada acima. Observa-se o a iteração 0 (zero) é a interpretação do próprio axioma da definição. Como o axioma é apenas um letra efe, então é desenhada uma reta através da interpretação gráfica. Na iteração seguinte (iteração 1), a reta já é substituída pela regra.

Figura 35 – Exemplo de ramificação. Iteração 0 e 1.



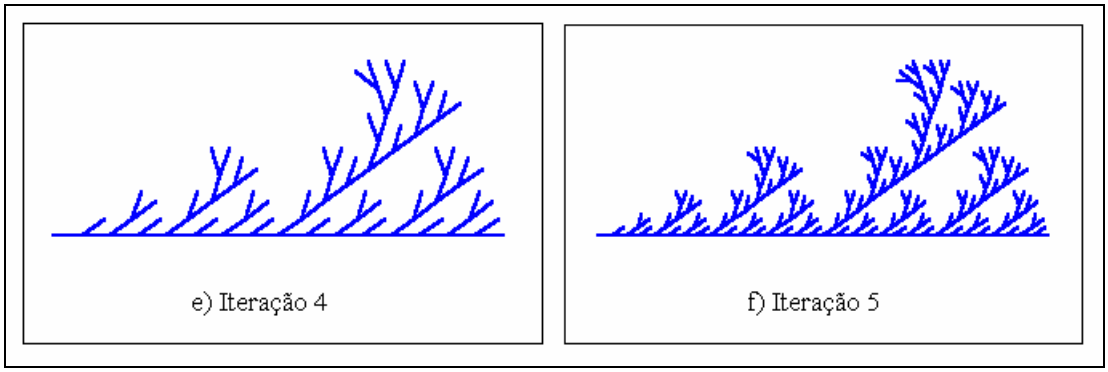
A Figura 36 mostra as iterações 2 e 3. Percebe-se claramente que cada reta está sendo substituída pela configuração da Figura 34.

Figura 36 – Exemplo de ramificação. Iteração 2 e 3.



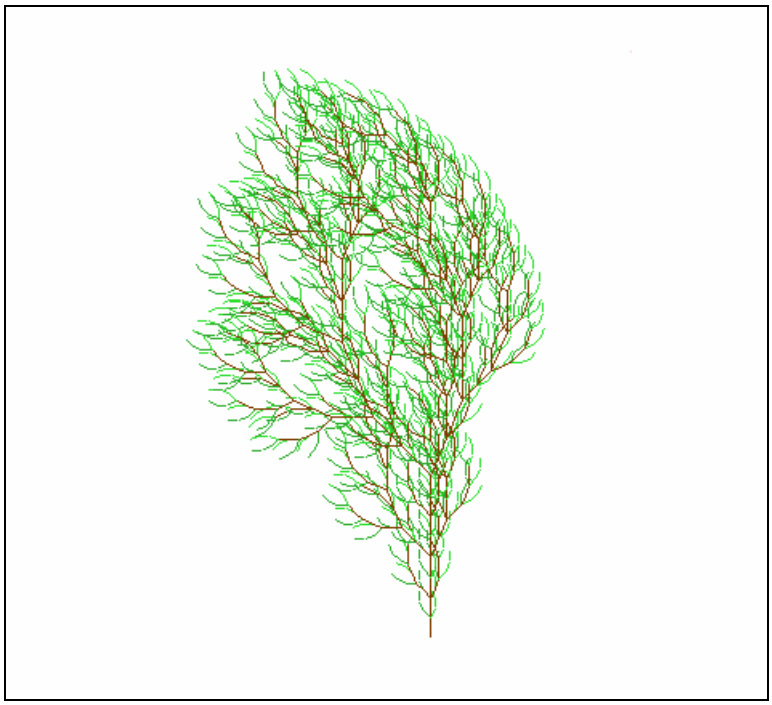
Em níveis de iteração maiores, a Figura 37 mostra a iteração 4 e 5 da definição Sistema-L dada acima. Quanto mais agrupamentos há na definição, maior será a complexidade da imagem à medida que o nível de iteração aumenta.

Figura 37 – Exemplo de ramificação. Iteração 4 e 5.



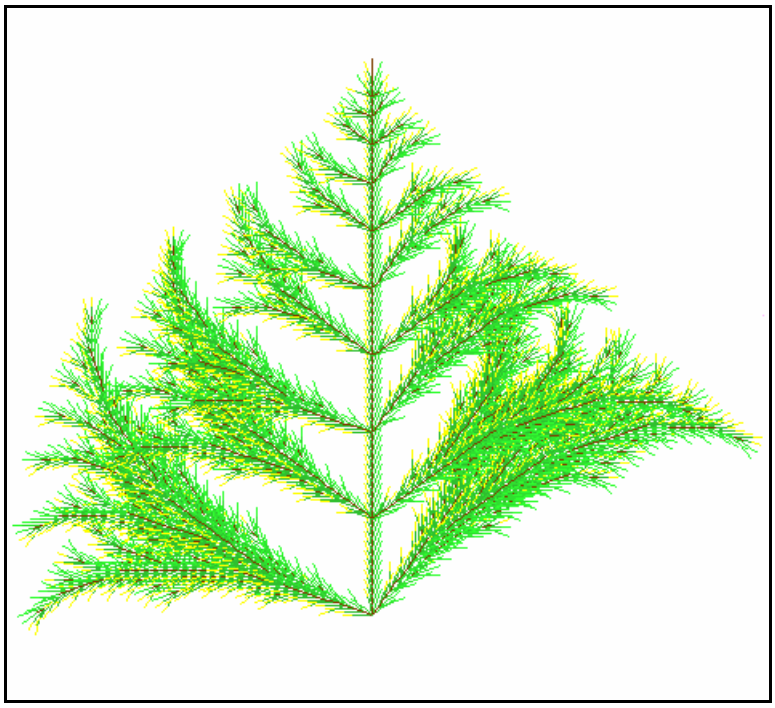
Exemplos de Sistemas-L agrupados estão nas figuras 38 e 39. As definições [PEI88] Sistema-L seguem as imagens. O que caracteriza um Sistema-L agrupado é a presença dos símbolos "[" e "]" nas regras.

Figura 38 – Formação agrupada de um arbusto em 4 iterações.



Direções	**16**
Axioma	**++++F**
Regras	p_1 : **F→6FF+[+2F-F-F]-[-4F+F+F]**

Figura 39 – Formação agrupada de um arbusto em 11 iterações.



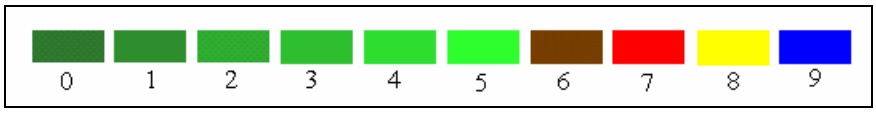
Direções	**20**
Axioma	**+++++SLFF**
Regras	p_1 : **T→6TL** p_2 : **S→[+++G][---G]TS** p_3 : **L→[4-FF5F][+3FF8F]6F** p_4 : **H→-G[+H]L** p_5 : **G→+H[-G]L**

Observa-se nas figuras que o nível de iteração não é alto, mas as imagens já possuem um grau de complexidade. A imagem não terá uma boa visualização se houver exageros nos agrupamentos e no nível de iteração.

Através da definição Sistema-L da Figura 39 é possível imaginar que criar uma definição que mapeie uma determinada dinâmica não é trivial. Em uma aplicação prática é necessário que se conheça, através de estudos, as métricas do processo em questão e que regras o governam, então o processo poderá ser representado através de símbolos formais.

Os números representam cores e podem variar de 0 a 9. Os primeiros 6 índices são tons de verde para deixar a imagem com um tom mais realístico. Neste trabalho podem ser coloridos os vários seguimentos de reta que formam a imagem ou colorir cada regra do conjunto de regras. Isso permite analisar quando as regras ocorrem. Dependendo do que será analisando através das cores, qualquer abordagem de coloração é válida. A Figura 40 mostra as cores disponíveis e seus índices.

Figura 40 – Paleta de cores.



3.10 RESUMO

Sistemas Dinâmicos Simbólicos são sistemas que se utilizam de caracteres para representar uma configuração física de um determinado processo, natural ou artificial.

Sistemas-L é uma gramática formal criada pelo botânico Aristid Lindenmayer para modelagem de crescimento de plantas e estruturas celulares. Tem como base a reescrita proposta por Chomsky, que nunca foi utilizada para modelagem de padrões naturais, mas diferenciada no método de substituição das regras, onde o método de Chomsky e seqüencial e o método proposto por Lindenmayer é paralelo.

Para que fosse possível a validação dos métodos de reescrita, foi proposta uma interpretação gráfica, baseada na tartaruga LOGO. O método de interpretação dos caracteres, nomeado de Tartaruga Gráfica, interpreta o resultado da reescrita e, através de seguimentos de reta, gera a imagem.

Além de existirem outros tipos de Sistema-L mais complexos, dois tipos foram apresentados: o Sistemas-LD0 e Sistemas-L estocásticos. O primeiro é determinístico e outro utiliza uma determinada estatística durante a substituição das regras. Ambos podem representar crescimento agrupado ou ramificações com a utilização dos caracteres "[" e "]".

4 DESCRIÇÃO DA IMPLEMENTAÇÃO E COMO USAR O PROTÓTIPO

Primeiramente, neste capítulo, é descrito a função do protótipo, depois serão explicados detalhes da implementação, ferramentas e linguagem utilizadas. A linguagem de modelagem utilizada para especificar o protótipo, bem como a arquitetura e estruturas de dados embutidas também serão abordadas. Para que seja obtido resultados satisfatórios do protótipo por um usuário leigo, são apresentados detalhadamente suas telas e opções do menu.

4.1 O PROTÓTIPO E SUAS CAPACIDADES

O protótipo realiza reescrita paralela partindo de uma definição formal contendo um axioma, um conjunto de regras de produção e um incremento angular. O processo de reescrita expande o axioma, gerando uma cadeia de caracteres que aumenta exponencialmente. O axioma expandido é interpretado gerando a imagem, que por sua vez é composta por um conjuntos de retas.

O protótipo pode gravar as imagens em formato .BMP no tamanho em que foram geradas, imprimir em qualquer formato de papel, gravar as definições e abrir uma já existente. Pode-se escolher, através do menu, o nível de iteração desejado, visualizar o nível imediatamente anterior ou o posterior. Para que seja possível visualizar o axioma expandido, com o objetivo de acompanhar sua evolução e validação, também é dada uma opção de menu que salva a cadeia de caracteres resultante do processo de reescrita em um arquivo texto. Também é possível acompanhar quantos caracteres, quantos bytes e o nível de iteração que estão sendo utilizados no processo, evitando, com isso, que o sistema pare de funcionar por falta de recursos, uma vez que o axioma informado aumenta exponencialmente.

4.2 DESCRIÇÃO DA IMPLEMENTAÇÃO

O protótipo foi implementado em um PC de configuração padrão com sistema operacional Windows 98. As ferramentas utilizadas, a linguagem de programação e modelagem são apresentados a seguir.

4.2.1 FERRAMENTAS E LINGUAGEM

Para o desenvolvimento do protótipo foi utilizado o ambiente de programação *Delphi 3.0* que utiliza como linguagem de programação o *Object Pascal*. Delphi é um ambiente de programação que pertence à uma família de ferramentas denominadas RAD (*Rapid Application Development* – Desenvolvimento Rápido de Aplicações) que é orientado a eventos mas que possui suporte a programação orientada a objetos [CAN96]. Além de muitas outras características não relevantes aqui, o ambiente Delphi foi escolhido por possuir um bom apoio ao desenvolvimento de interfaces e desenvolvimento gráfico e, também uma maior afinidade por parte do autor.

A ferramenta utilizada para a modelagem do protótipo foi uma versão beta do *Rational Rose 4.0* da Rational Corporation. O Rational Rose é uma ferramenta para modelagem de sistemas orientados a objetos que utilizam a UML (*Unified Modeling Language* – Linguagem de Modelagem Unificada). De acordo com [FUR98], a UML é uma linguagem de modelagem, não uma metodologia. A UML reúne o que há de melhor entre as técnicas de modelagem OMT (*Object Modeling Technique*), Booch e OOSE (*Object-Oriented Software Engineering*) com o objetivo de tornar única as técnicas de produção de modelos que especificam o domínio do problema de um sistema, que é a Orientação a Objetos.

A UML é a linguagem padrão para especificar, visualizar, documentar e construir artefatos de um sistema e pode ser utilizada com todos os processos ao longo do ciclo de desenvolvimento e através de diferentes tecnologias de implementação [FUR98].

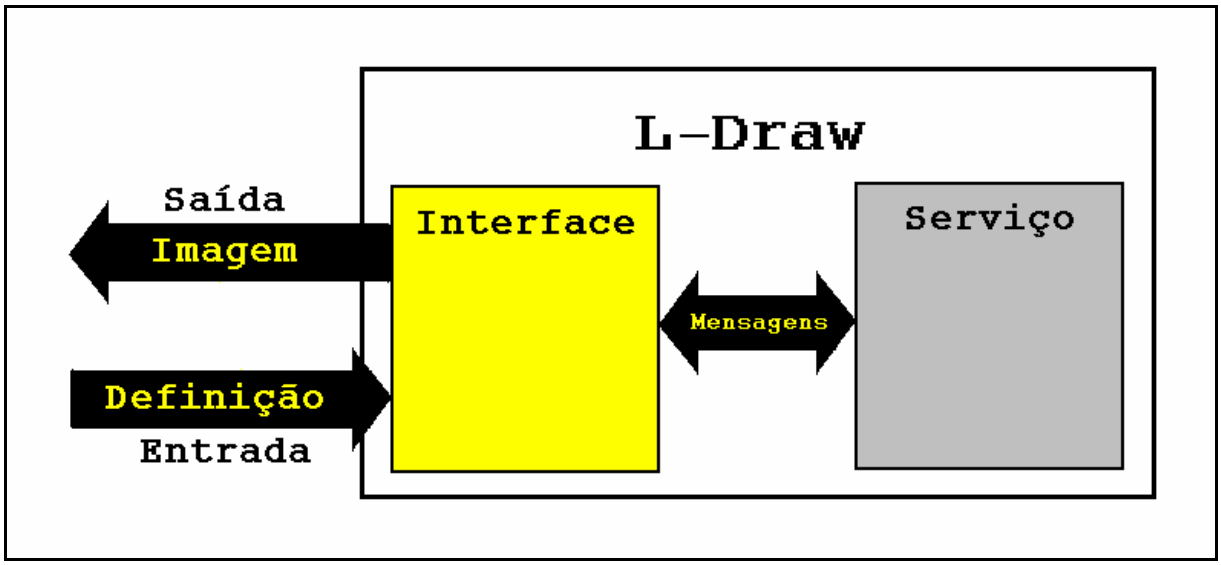
Para descrever os vários aspectos de modelagem pela UML é utilizado a notação definida pelos seus vários tipos de diagramas. Um diagrama é uma apresentação gráfica de uma coleção de elementos de modelos, freqüentemente mostrado como um grafo conectado

de arcos (relacionamentos) e vértices (outros elementos do modelo) [FUR98]. Neste processo de modelagem são utilizados ícones, símbolos 2-D, caminhos e texto.

4.2.2 ARQUITETURA DO PROTÓTIPO

Seguindo uma das tendências no desenvolvimento de Software [HAM99], o protótipo, aqui nomeado de *L-Draw*, foi dividido em duas camadas, representadas na Figura 41.

Figura 41 – Arquitetura do protótipo.



O L-Draw possui como entrada uma definição Sistema-L, constituída por um axioma, um conjunto de regras e um incremento angular e a saída é uma imagem formada por retas. A camada de interface entrega a definição à camada de serviço, que por sua vez, realiza todo o processamento de reescrita e devolve à camada interface o axioma expandido. A camada de interface percorre o axioma expandido, caracter a caracter, envia cada caracter para a camada de serviço que passa informações necessárias para que a imagem seja desenhada pela camada de interface.

A principal razão por adotar-se essa arquitetura é simplesmente a reutilização da parte computacional que compõe o L-Draw, ou seja, todos os procedimentos e funções utilizados para o processamento são independentes de linguagem. Não importando qual linguagem, a

lógica algorítmica será a mesma. O que varia de uma linguagem para outra é a maneira pela qual a interface interage com esse processamento.

Neste caso, para uma implementação em outra linguagem, basta criar uma camada de interface adequada utilizando a linguagem escolhida para que manipule entrada/saída e, que através de uma DLL (*Dynamic Link Library* – Biblioteca de Ligação Dinâmica). Neste caso a camada de serviço deveria se transformar em uma DLL para a linguagem na qual foi desenvolvida a interface.

Cada camada foi modelada e implementada como uma classe, totalizado duas classes, as quais se comunicam através de mensagens. Detalhes e funcionamento destas classes, bem como a modelagem em UML estão definidas logo a seguir.

4.2.3 MODELAGEM

A UML possui vários diagramas, cada um possui uma particularidade aplicada aos diversos casos de modelagem. Aqui, os diagramas julgados necessários são o diagrama de caso de uso, diagrama de atividades, diagrama de classe e diagrama de seqüência.

Diagrama de caso de uso: os casos de uso descrevem a funcionalidade do sistema percebida por atores externos [FUR98], neste caso é o próprio usuário. O L-Draw apresenta apenas um caso de uso o qual está representado abaixo na Figura 42.

Figura 42 – Diagrama de caso de uso do L-Draw.

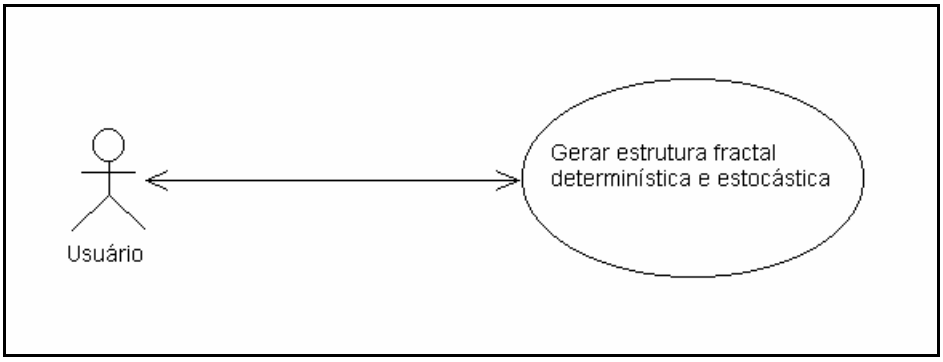


Diagrama de atividades: um dos objetivos do diagrama de atividades é representar os fluxos dirigidos por processamento interno, descrevendo as atividades desempenhadas em uma operação [FUR98]. A Figura 43 mostra uma visão macro do protótipo, mostrando as atividades executadas. Dada uma definição Sistema-L, a primeira atividade do sistema e expandir o axioma utilizando a reescrita determinística ou estocástica. Logo em seguida é calculado o tamanho da imagem e, finalmente, é feita a interpretação gráfica.

Figura 43 – Diagrama de atividades do L-Draw.

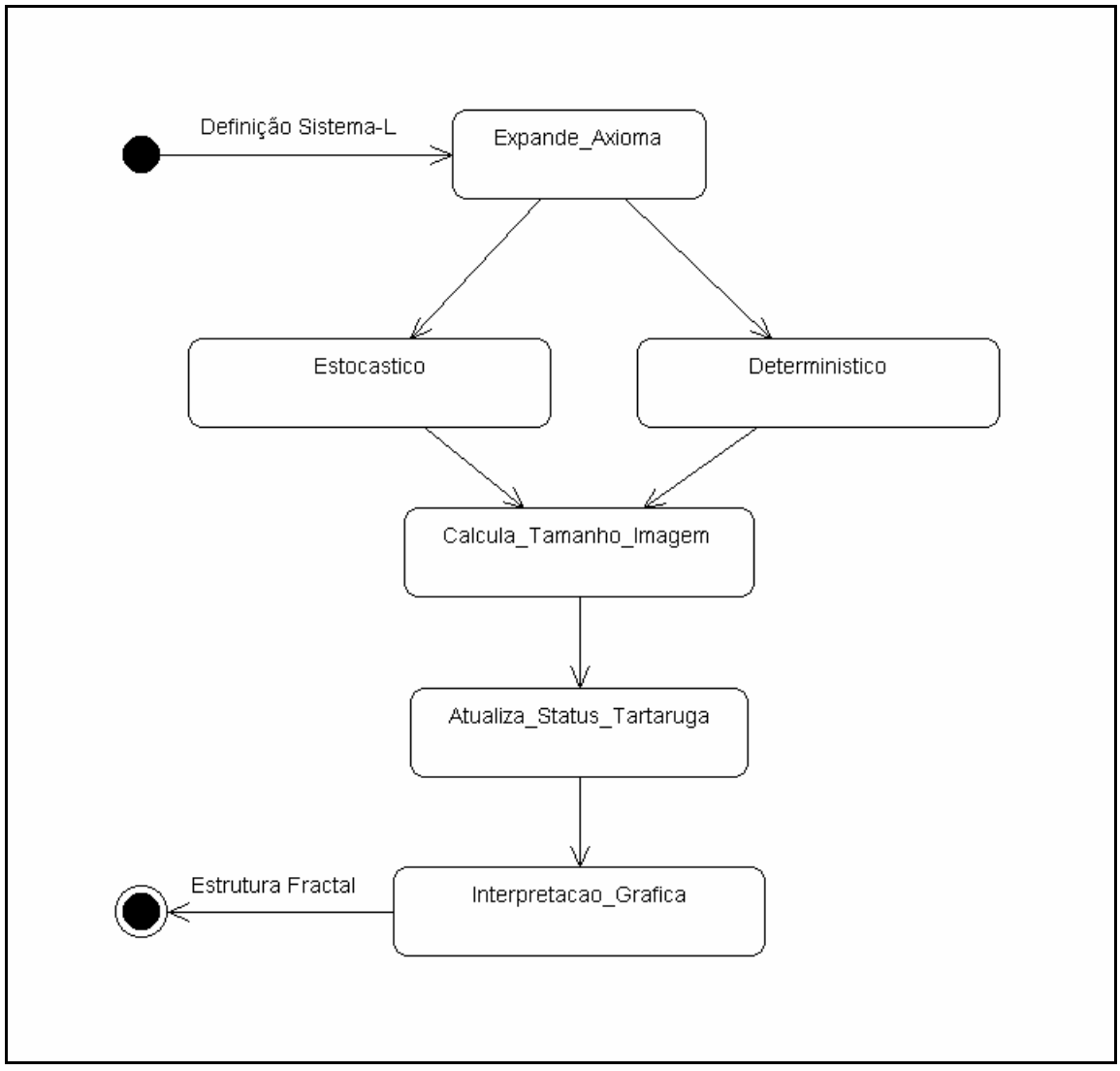


Diagrama de classe: denota a estrutura estática do sistema e as classes que representam coisas que são manipuladas por este sistema. É um gráfico bidimensional de elementos de modelagem [FUR98]. O diagrama de classe do L-Draw é formado por duas classes, onde cada classe representa uma camada. Está representado logo abaixo na Figura 44.

Os métodos presentes na classe de interface estão ligados a procedimentos como gravar e ler informações do disco, imprimir a imagem ou uma definição, limpar a tela. A classe de serviço implementa o que é relevante para o processamento como um todo, independentemente da interface. A classe de serviço fornece dados já processados que são utilizados pela interface para que seja desenhada a estrutura na tela.

Basicamente, a classe de interface lê uma definição Sistema-L de um arquivo com formato 2SL (esse formato é explicado mais adiante), passando para a classe de serviço que realiza todo o processamento de reescrita e validação das regras. Como os caracteres do axioma são alocados dinamicamente em memória, a classe de serviço repassa o ponteiro de início do axioma. A classe de interface percorre a lista encadeada de caracteres (axioma expandido) enviando um a um para a classe de serviço que atualiza o *status* do caminhamento da tartaruga devolvendo as coordenadas necessárias para a classe de interface que realiza o comando de desenho.

Diagrama de seqüência: apresenta a iteração de seqüência de tempo dos objetos que participam na iteração. O aspecto mais importante desse diagrama é mostrar a seqüência de mensagens enviadas entre os objetos [FUR98]. O L-Draw possui o diagrama de seqüência mostrado na Figura 45.

Figura 44 – Diagrama de classe do L-Draw.

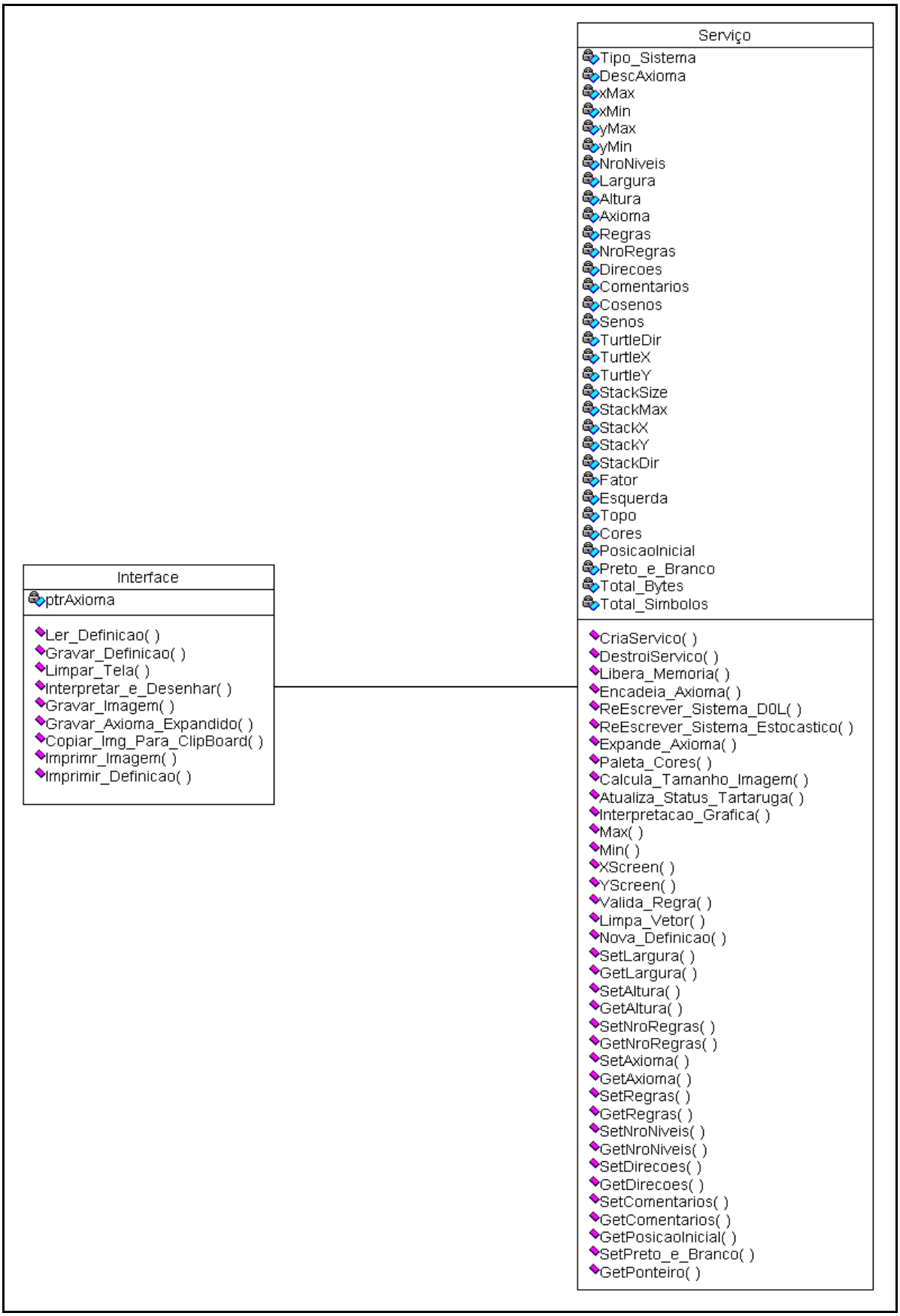
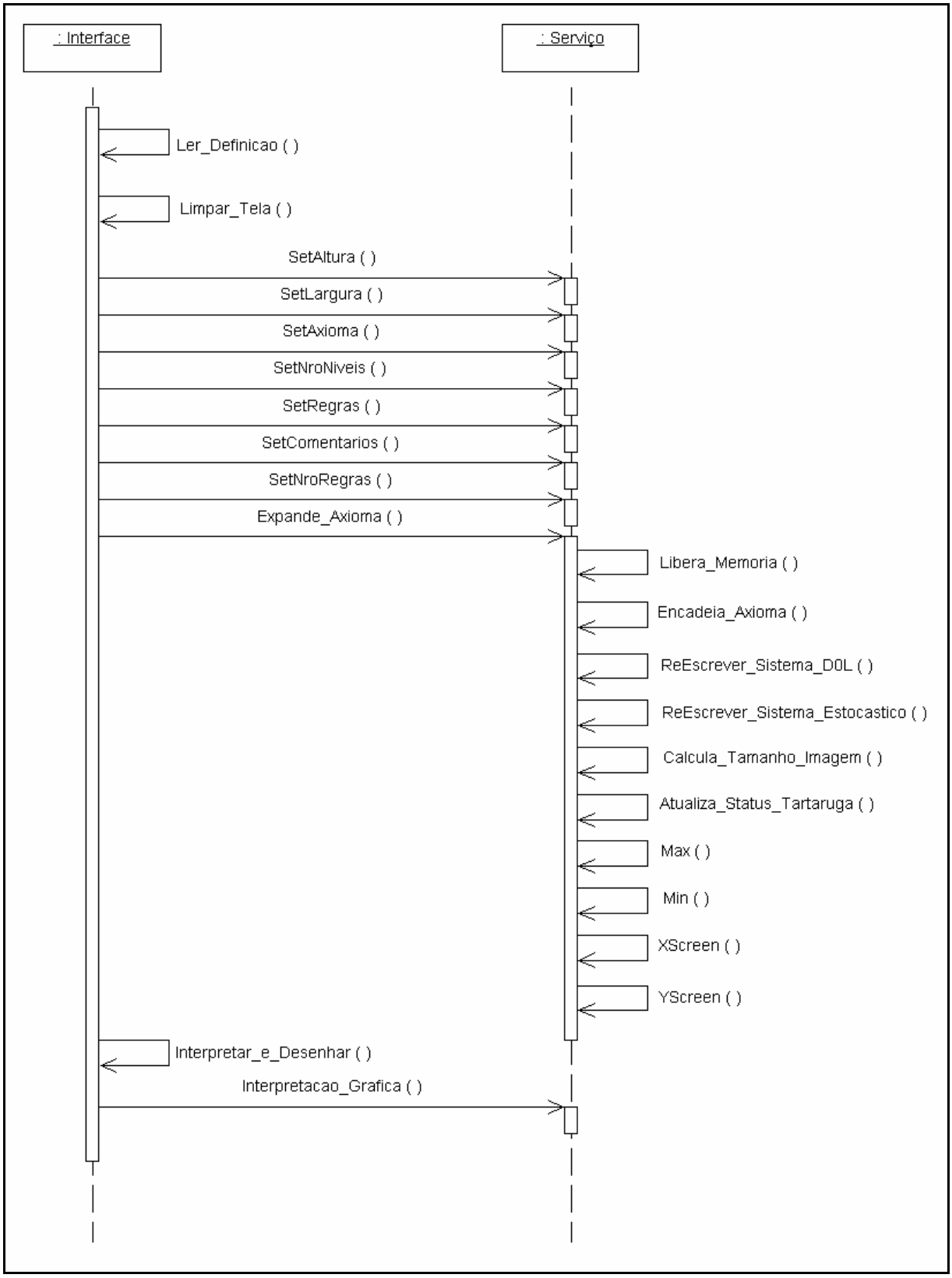


Figura 45 – Diagrama de seqüência do L-Draw.



4.2.4 ESTRUTURA DE DADOS UTILIZADA

O processo de substituição das regras é feito em memória através de uma lista simples encadeada dinamicamente. A decisão por utilizar lista neste processo foi simplesmente o ganho de performance no manuseio de cadeia de caracteres, que através de comandos oferecidos pela linguagem de programação ficavam cada vez mais lento à medida que o axioma se expandia. Isso foi percebido através de uma tentativa em utilizá-los.

A estrutura do nó e do descritor da lista, bem como outras estruturas estão representados abaixo.

O tipo que armazena o ponteiro para uma estrutura TAxioma.

```
TPtrAxioma   =      ^TAxioma;
```

Cada nó da lista é formado por um símbolo e um ponteiro para o próximo símbolo.

```
TAxioma      =      record
                        Comando     : char;
                        Proximo     : TPtrAxioma;
                    end;
```

A lista possui um descritor como definido abaixo, contendo o primeiro e o último endereço do símbolo.

```
TDescAxioma      =      record
                            Primeiro     : TPtrAxioma;
                            Ultimo       : TPtrAxioma;
                        end;
```

O tipo TString100 é um tipo utilizado sempre que se deseja uma cadeia de caracteres com até 100 caracteres.

```
TString100   =      String[100];
```

O tipo TRegras é um vetor de até 20 regras, onde cada regra pode conter até 100 caracteres.

```
TRegras = array [1..20] of TString100;
```

As definições Sistema-L podem ser gravadas e lidas de um arquivo binário com extensão 2SL (2 = duas dimensões e SL = Sistema de Lindenmayer). Cada arquivo pode possuir apenas uma definição que corresponde o *layout* seguinte.

```
TRecArquivo =     record
                      Comentarios  : TString100;
                      Axioma       : TString100;
                      Regras       : TRegras;
                      Direcoes     : byte;
                      NroRegras    : integer;
                  end;
```

O arquivo pode conter um comentário sobre a definição, um axioma, um conjunto de regras, o número de direções e total de regras. Exceto os comentários, o axioma, as regras e o número de direções são necessários para a geração das imagens.

As imagens geradas podem ser gravadas em arquivo de formato BMP (*Bit Map* - Mapa de Bits) ou copiadas para a área de transferência do Windows 98 e utilizadas em outros ambientes. Também podem ser impressas independentemente de dispositivos.

É importante lembrar os tipos não definidos pelo usuário e a sintaxe para a declaração dos tipos pertencem a linguagem de programação utilizada. Para saber sobre Delphi verificar em [CAN96].

O item seguinte mostrará como foi implementado a reescrita determinística e estocástica e como é feita a interpretação gráfica utilizando a sintaxe da linguagem de programação *Object Pascal*.

4.3 FUNCIONAMENTO E IMPLEMENTAÇÃO

A classe de interface é responsável pelo método de leitura da definição Sistema-L, ou seja, o axioma, os comentários, o conjunto de regras e número de direções são repassados à classe de serviço através da classe de interface. Também são repassados a altura e largura do plano onde será desenhada a imagem, o número de níveis de reescrita e a quantidade de regras que contém a definição. Com estas informações, a interface executa o método *Expande_Axioma()* que pertence à classe de serviço, representado no quadro 1.

Quadro 1 – Método Expande_Axioma().

```
function TServico.Expande_Axioma:TPtrAxioma;
var
  I       : integer;
  Comando : Char;
  nRegras : byte;
begin
  if DescAxioma.Primeiro <> nil then
    Libera_Memoria;
  Encadeia_Axioma;
  if Direcoes = 0 then
    raise Erro.Create('Não há número de direções informado!');
  if NroRegras = 0 then
    raise Erro.Create('Não há regras informada!');
  I:=1;
  nRegras:=1;
  while (I <= nroRegras) and (nRegras = 1 ) do
  begin
    Comando:=Regras[I][1];
    if Comando = Regras[I+1][1] then
      inc(nRegras)
    else
      inc(I);
  end;
  if nRegras = 1 then
  begin
    ReEscrever_Sistema_D0L;
    Tipo_Sistema:='Sistema-L Determinístico';
  end
  else
  begin
    ReEscrever_Sistema_Estocastico;
    Tipo_Sistema:='Sistema-L Estocástico';
  end;
  Result:=DescAxioma.Primeiro;
end;
```

O método *Expande_Axioma()* libera a memória ocupada, se houver, encadeia o axioma, através do método *Expande_Axioma()*, mostrado no quadro 2, verifica se foi informado o número de direções e o número de regras. Também identifica o tipo de Sistema-L, se é determinístico ou estocástico, armazenado-o no atributo *Tipo_Sistema* da classe de serviço.

Para saber se é estocástico, é verificado se há mais de um antecessor que seja igual a um símbolo contido no axioma, então é chamado o método *Reescrever_Sistema_Estocastico()*, representado no quadro 3. Se, para cada símbolo do axioma, houver apenas um antecessor correspondente no conjunto de regras, é determinístico, então o método *Reescrever_Sistema_D0L()* é chamado, representado no quadro 4.

Quadro 2 – Método Encadeia_Axioma().

```
procedure TServico.Encadeia_Axioma;
var
  I         : longint;
  Atual     : TPtrAxioma;
  AuxAxioma : TString100;
begin
  if Axioma = '' then
    raise Erro.Create('Axioma não informado!')
  else
  begin
    if Preto_e_Branco then
    begin
      AuxAxioma:=Axioma;
      for I:=1 to length(Axioma) do
      begin
        if Axioma[I] in ['0'..'9'] then
          Delete(Axioma,I,1);
      end;
    end;
    DescAxioma.Primeiro:=nil;
    DescAxioma.Ultimo:=nil;
    for I:=1 to length(Axioma) do
    begin
      GetMem(Atual,5);
      Atual^.Proximo:=nil;
      Atual^.Comando:=Axioma[I];
      if DescAxioma.Primeiro = nil then
        DescAxioma.Primeiro:=Atual
      else
        DescAxioma.Ultimo^.Proximo:=Atual;
      DescAxioma.Ultimo:=Atual;
    end;
    if Preto_e_Branco then
      Axioma:=AuxAxioma;
  end;
end;
```

O método *Encadeia_Axioma()* verifica se foi especificado um axioma e retira os caracteres que representam cores do axioma, se for informado que a imagem deva ser em preto e branco. O encadeamento é feito, e o axioma original volta a ter os caracteres de cor novamente se o atributo *Preto_e_Branco* for verdadeiro. O mesmo acontece com o conjunto de regras.

Quadro 3 – Método Reescrever_Sistema_Estocastico().

```
procedure TServico.ReEscrever_Sistema_Estocastico;
var
  AuxRegras  : TRegras;
  AuxDesc    : TDescAxioma;
  Atual,Aux  : TPtrAxioma;
  Nivel,I,k  : Longint;
  Utilizadas : array [1..20] of Longint;
  Achou      : Boolean;
  Regra      : TString100;
  Fs         : Longint;
  cChance    : String[02];
  nChance, Total : double;
  P : Byte;
  J : Byte;
begin
  //Retira os caracteres de cor das regras
  if Preto_e_Branco then
  begin
     AuxRegras:=Regras;
    for I:=1 to NroRegras do
    begin
      P:=Pos(')',Regras[I]);
      for J:=1 to Length(Regras[I]) do
      begin
        if (Regras[I][J] in ['0'..'9']) and (J > P) then
          Delete(Regras[I],J,1);
      end;
    end;
  end;

  //Executa o processo de substituição dos caracteres
  FillChar(Utilizadas,SizeOf(Utilizadas),0);
  Fs:=0;
  for Nivel:=1 to nroNiveis do
  begin

    AuxDesc.Primeiro:=nil;
    AuxDesc.Ultimo:=nil;
```

```
    Atual:=DescAxioma.Primeiro;

    while Atual <> nil do
    begin

      if Atual^.Comando = 'F' then
        inc(Fs);

      Achou:=False;
      repeat
        K:=Random(nroRegras+1);
        if (K > 0) then
        begin
        Achou:=True;
        cChance:=Copy(Regras[K],
                 Pos('(',Regras[K])+1,Pos(')',Regras[K])-
                 Pos('(',Regras[K])-1);
        nChance:=StrToInt(cChance);
        Total:=Fs*(nChance/100);
        if Utilizadas[K] < Total then
        begin
          inc(Utilizadas[K]);
          Achou:=True;
        end;
        end;
      until Achou;

      if Atual^.Comando = Regras[k][1] then
      begin

        Regra:=Regras[K];
        Delete(Regra,1,Pos(')',Regra));

        for I:=1 to Length(Regra) do
        begin
          GetMem(Aux,5);
          if Aux <> nil then
          begin
            Aux^.Proximo:=nil;
            Aux^.Comando:=Regra[I];
            if AuxDesc.Primeiro = nil then
              AuxDesc.Primeiro:=Aux
            else
              AuxDesc.Ultimo^.Proximo:=Aux;
            AuxDesc.Ultimo:=Aux;
          end
          else
          begin
            raise Erro.Create('Não há memória suficiente!);
```

```
          end;
        end;
      end
      else
      begin
        GetMem(Aux,5);
        Aux^.Proximo:=nil;
        Aux^.Comando:=Atual^.Comando;
        if AuxDesc.Primeiro = nil then
          AuxDesc.Primeiro:=Aux
        else
          AuxDesc.Ultimo^.Proximo:=Aux;
        AuxDesc.Ultimo:=Aux;
      end;
      DescAxioma:=AuxDesc;
      Atual:=Atual^.Proximo;
    end;
  end;
  if Preto_e_Branco then
     Regras:=AuxRegras;
end;
```

O método *Rescrever_Sistema_Estocástico()*, inicialmente, retira os caracteres de cor das regras do conjunto de regras, se for necessário. Em uma estrutura de repetição que vai de 1 até o número de níveis informado, é inicializado um descritor auxiliar e toma-se o endereço do primeiro símbolo. Enquanto houver símbolos, são contados os caracteres **F**. Uma outra estrutura de repetição sorteia randomicamente números que podem ir de 1 até o número de regras informado.

Após o sorteio do número, que é um índice para o vetor que contém as regras, e verificado qual é a sua probabilidade. Toma-se o total de símbolos **F** já passados e multiplica-se pelo quociente da probabilidade por 100. Verifica-se quantas vezes a regra sorteada já foi utilizada, se não ultrapassar esse resultado, a regra é utilizada para a substituição, do contrário, é feito um novo sorteio.

Isso é feito enquanto houver símbolos encadeados. Ao término, o descritor da classe recebe o endereço do primeiro e ultimo símbolo que está no descritor auxiliar.

Quadro 4 – Método Reescrever-Sistema_D0L().

```
procedure TServico.ReEscrever_Sistema_D0L;
var
  Regra     : TString100;
  AuxDesc   : TDescAxioma;
  AuxRegras : TRegras;
  Comando   : char;
  Atual,Aux : TPtrAxioma;
  Nivel,I,k : Longint;
  J : Byte;
begin
  //Retira os caracteres de cor das regras
  if Preto_e_Branco then
  begin
     AuxRegras:=Regras;
    for I:=1 to NroRegras do
    begin
      for J:=1 to Length(Regras[I]) do
      begin
        if (Regras[I][J] in ['0'..'9']) then
          Delete(Regras[I],J,1);
      end;
    end;
  end;

  //Executa o processo de substituição dos caracteres
  for Nivel:=1 to nroNiveis do
  begin
    AuxDesc.Primeiro:=nil;
    AuxDesc.Ultimo:=nil;
    Atual:=DescAxioma.Primeiro;
    while Atual <> nil do
    begin
      Comando:=Atual^.Comando;
      K:=1;
      while (K <= nroRegras ) and (Comando <> Regras[K][1]) do
        inc(K);
      if Atual^.Comando = Regras[k][1] then
      begin
        Regra:=Regras[K];
        Comando:=Regra[1];
        Delete(Regra,1,4);
        for I:=1 to Length(Regra) do
        begin
          GetMem(Aux,5);
          if Aux <> nil then
          begin
            Aux^.Proximo:=nil;
```

```
            Aux^.Comando:=Regra[I];
            if AuxDesc.Primeiro = nil then
              AuxDesc.Primeiro:=Aux
            else
              AuxDesc.Ultimo^.Proximo:=Aux;
            AuxDesc.Ultimo:=Aux;
          end
          else
          begin
            raise Erro.Create('Não há memória suficiente!);
        end;
      end
      else
      begin
        GetMem(Aux,5);
        Aux^.Proximo:=nil;
        Aux^.Comando:=Atual^.Comando;
        if AuxDesc.Primeiro = nil then
          AuxDesc.Primeiro:=Aux
        else
          AuxDesc.Ultimo^.Proximo:=Aux;
        AuxDesc.Ultimo:=Aux;
      end;
      DescAxioma:=AuxDesc;
      Atual:=Atual^.Proximo;
    end;
  end;
  if Preto_e_Branco then
     Regras:=AuxRegras;
end;
```

O método *Reescrever_Sistema_D0L()* percorre o axioma e faz a substituição pela regra quando encontrar um símbolo que tenha um antecessor correspondente no conjunto de regras. O método *Reescrever_Sistema_Estocastico()* faz o mesmo procedimento, mas antes de fazer a substituição, verifica a probabilidade da regra encontrada.

O método *Expande_Axioma()* chamado pela interface, devolve um ponteiro para uma lista contendo o endereço do primeiro símbolo do axioma expandido. A interface percorre essa lista, nó a nó, e pede para classe de serviço que calcule as coordenadas na tela e envie a cor, se foi feito a reescrita considerando as cores da paleta de cores.

O quadro 5 mostra como é feito o caminhamento na lista e obtenção das coordenadas através do método *Interpretacao_Grafica()* da classe de serviço que é chamado pela interface.

Quadro 5 – Interpretação do Axioma Expandido.

```
.
.
.
while PtrAxioma <> nil do
  begin
    Posicao:=Servico.interpretacao_Grafica(PtrAxioma.Comando);
    case PtrAxioma.Comando of
      'F'     : Tela.LineTo(Posicao.X,Posicao.Y);
      'f',']' : Tela.MoveTo(Posicao.X,Posicao.Y);
      '0'..'9': Tela.Pen.Color:=Posicao.Cor;
    end;
    PtrAxioma:=PtrAxioma.Proximo;
  end;
```

O método *Interpretacao_Grafica()*, representado no quadro 6, leva como parâmetro o símbolo contido no endereço de memória atual, devolvendo a coordenada **(*X,Y*)** para que seja desenhada a imagem. Se o símbolo encontrado for **"F"**, então a interface deve desenhar um segmento de reta, se for **"f"** ou **"]"** deve apenas mover a posição da tartaruga, sem desenhar nada e se for um símbolo de 0 à 9, é uma das cores da paleta de cor. Os comandos gráficos mudam para interfaces diferentes. Aqui a sintaxe utilizada é do Delphi 3.0.

Os atributos *TurtleX* e *TurtleY* acumulam o coseno e o seno dos ângulos de desvio da tartaruga, respectivamente. À medida que a tartaruga encontra o símbolo +, o atributo *TurtleDir* é incrementado. Se encontrar o símbolo -, *TurtleDir* é decrementado. *TurtleDir* é um número que indica quantos graus deve virar a tartaruga gráfica quando encontrar um símbolo **F**.

O atributo *Fator* é uma constante calculada de acordo como mostra o quadro 7. Os atributos *xMin, yMin, xMax, yMax* armazenam as coordenadas superior esquerda e inferior direita, respectivamente. O valor de *x* é calculado pelo método *Xscreen()* e o valor de *y* é calculado através do método *Yscreen()*. Os métodos *Xscreen()* e *Yscreen()* são mostrados no quadro 8.

Quadro 6 – Método Interpretação_Grafica().

```
function Servico.interpretacao_Grafica(Comando:Char):TPosicao;
var
  Posicao : TPosicao;
begin
  FillChar(Posicao,SizeOf(Posicao),0);
  if not (Comando in ['0'..'9']) then
       Atualiza_Status_Tartaruga(Comando);
  case Comando of
     'F' : begin
           Posicao.X:=XScreen(TurtleX, Fator, xMin)+Esquerda;
           Posicao.Y:=YScreen(TurtleY, Fator, yMin)+Topo;
           end;
 'f',']' : begin
           Posicao.X:=XScreen(TurtleX, Fator, xMin)+Esquerda;
           Posicao.Y:=YScreen(TurtleY, Fator, yMin)+Topo;
           end;
'0'..'9' : begin
                Posicao.Cor:=Cores[StrToInt(Comando)];
           end;
     end;
  Result:=Posicao;
end;
```

Antes da imagem ser desenhada, é necessário que se chame o método *Calcula_Tamanho_Imagem()*, representado no quadro 8. Este método preenche dois vetores contendo os senos e os cosenos de cada ângulo que será utilizado pela tartaruga gráfica.

Através da diferença entre a altura máxima e a mínima e da diferença entre a largura máxima e a mínima do plano onde deve ser desenhada a imagem, e calculado uma distância fixa (*Fator*) que é utilizada no cálculo das coordenadas.

Quadro 7 – Métodos Xscreen() e Yscreen().

```
     function TServico.XScreen(X, Fator, xMin: Double):integer;
     begin
       Result:=Round(Fator*(X-xMin));
     end;

     function TServico.YScreen(Y, Fator, yMin: Double):integer;
     begin
       Result:=Round(Fator*(Y-yMin));
     end;
```

Quadro 8 – Método Calcula_Tamanho_Imagem().

```
    procedure TServico.Calcula_Tamanho_Imagem;
    var
      Atual          : TPtrAxioma;
      Comando        : Char;
      I              : Longint;
      DeltaX,DeltaY  : Double;
    begin
      for I:=0 to Direcoes-1 do
      begin
        Cosenos[I]:=cos(2*pi*I/Direcoes);
        Senos[I]  :=sin(2*pi*I/Direcoes);
      end;
      TurtleDir:=0;
      TurtleX:=0.0; TurtleY:=0.0;
      xMin:=0; yMin:=0; xMax:=0; yMax:=0;
      Atual:=DescAxioma.Primeiro;
      while Atual <> nil do
      begin
        Comando:=Atual^.Comando;
        Atualiza_Status_Tartaruga(Comando);
        if Comando in ['F','f'] then
        begin
          xMax:=Max(TurtleX,xMax);
          xMin:=Min(TurtleX,xMin);
          yMax:=Max(TurtleY,yMax);
          yMin:=Min(TurtleY,yMin);
        end;
        Atual:=Atual^.Proximo;
      end;
    TurtleDir:=0;
    TurtleX:=0.0;
    TurtleY:=0.0;
    DeltaX:=xMax-xMin;
    DeltaY:=yMax-yMin;
    if DeltaY <> 0 then
      Fator:=Min((Largura-60)/(DeltaX),(Altura-60)/(DeltaY))
    else
      Fator:=0;
    Esquerda:=Round((Largura - DeltaX*Fator)/2);
    Topo:=    Round((Altura - 25 - DeltaY*Fator)/2);
      PosicaoInicial.X:=XScreen(TurtleX,Fator,xMin)+Esquerda;
      PosicaoInicial.Y:=YScreen(TurtleY,Fator,yMin)+Topo;
end;
```

Para que a imagem fique no centro do plano, também é calculado quanto deve ser somado à *x* (*Esquerda*) e quanto deve ser somado à *y* (*Topo*). O ponto inicial da imagem também é calculado pelo método *Calcula_Tamanho_Imagem()*.

O método *Calcula_Tamanho_Imagem()* e o método *Interpretacao_Grafica()*, chamam o método *Atualiza_Status_Tartaruga()* que controla a posição da tartaruga gráfica no plano. A identificação dos símbolos é feita neste momento, como mostra o quadro 9.

Quadro 9 – Método Atualiza_Status_Tartaruga().

```
Procedure TServico.Atualiza_Status_Tartaruga(Comando : Char);
begin
     case Comando of
       'F','f' : begin
                   TurtleX:=TurtleX+Cosenos[TurtleDir];
                   TurtleY:=TurtleY+Senos[TurtleDir];
                 end;
       '+'     : begin
                   dec(TurtleDir);
                   if TurtleDir < 0 then
                     TurtleDir:=Direcoes-1;
                 end;
       '-'     : begin
                   inc(TurtleDir);
                   if TurtleDir = Direcoes then
                     TurtleDir := 0;
                 end;
       '|'     : begin
                   TurtleDir:=TurtleDir + Direcoes div 2;
                   if TurtleDir > Direcoes then
                     TurtleDir:=TurtleDir - Direcoes;
                 end;
       '['     : begin
                   StackX[StackSize]:=TurtleX;
                   StackY[StackSize]:=TurtleY;
                   StackDir[StackSize]:=TurtleDir;
                   inc(StackSize);
                 end;
       ']'     : begin
                   dec(StackSize);
                   TurtleX:=StackX[StackSize];
                   TurtleY:=StackY[StackSize];
                   TurtleDir:=StackDir[StackSize];
                 end;
       end;
end;
```

As tabelas 5 e 6 mostram uma breve descrição dos atributos e métodos da classe de serviço, respectivamente.

Tabela 5 – Documentação dos atributos da classe de serviço.

Atributos	Informação que armazena
Tipo_Sistema	O tipo de Sistema-L.
DescAxioma	O primeiro e o último ponteiro para o axioma expandido.
Xmax	Maior valor de *x*.
Xmin	Menor valor de *x*.
Ymax	Maior valor de *y*.
Ymin	Menor valor de *y*.
NroNiveis	O Número de níveis de reescrita.
Largura	Largura do plano onde será desenhada a imagem.
Altura	Altura do plano onde será desenhada a imagem.
Regras	O conjunto de regras.
NroRegras	O número de regras que a definição contém
Direcoes	O número de direções.
Comentarios	Os comentários à respeito da definição.
Cosenos	Os cosenos de cada ângulo que a tartaruga gráfica utilizará.
Senos	Os senos de cada ângulo que a tartaruga gráfica utilizará.
TurtleDir	O número de direções enquanto a tartaruga gráfica caminha no plano.
TurtleX	O acúmulo de coseno.
TurtleY	O acúmulo de seno.
StackSize	Tamanho da pilha.
StackMax	Tamanho máximo da pilha.
StackX	Posições *x* (na pilha).
StackY	Posições *y* (na pilha).
StackDir	Posições da tartaruga gráfica (na pilha).
Fator	Distância fixa.
Esquerda	Valor que deve ser somado à *x* para que a imagem fique no centro.
Topo	Valor que deve ser somado *y* para que a imagem fique no centro.
PosicaoInicial	A coordenada inicial da imagem.
Preto_e_Branco	Verdadeiro (preto e branco)/ Falso (colorido)
Total_Bytes	Total de bytes utilizados na reescrita.
Total_Simbolos	Total de símbolos (caracteres) utilizado na reescrita.

Tabela 6 – Documentação dos métodos da classe de serviço.

Método	Comportamento
CriaServico()	Inicia os atributos do objeto servico.
DestroiServico()	Libera a instância da classe servico.
Libera_Memoria()	Devolve ao sistema a memória ocupada na reescrita.
Encadeia_Axioma()	Encadeia o axioma em uma lista dinâmica.
ReEscrever_Sistema_D0L()	Executa a reescrita determinística.
ReEscrever_Sistema_Estocastico()	Executa a reescrita estocástica.
Expande_Axioma()	Função que expande o axioma e devolve o ponteiro inicial da lista.
Paleta_Cores()	Cria uma paleta com 10 cores.
Calcula_Tamanho_Imagem()	Calcula o tamanho da imagem.
Atualiza_Status_Tartaruga()	Administra a posição da tartaruga gráfica no plano.
Interpretacao_Grafica()	Função que interpreta o axioma, devolvendo a coordenada (*x,y*) e uma cor.
Max()	Função que devolve o maior entre dois valores passados como parâmetro.
Min()	Função que devolve o menor entre dois valores passados como parâmetro.
Xscreen()	Função que calcula o valor *x* da coordenada.
Yscreen()	Função que calcula o valor *y* da coordenada.
Valida_Regra()	Função que verifica se a regra foi passada corretamente.
Limpa_Vetor()	Limpa o vetor que contém as regras da definição.
Nova_Definicao()	Prepara a classe de servico para uma nova reescrita.
SetLargura()	Atribui a largura do plano ao atributo *Largura*.
GetLargura()	Função que lê o atributo *Largura*.
SetNroRegras()	Atribui o número de regras ao atributo *NroRegras*.
GetNroRegras()	Função que lê o atributo *NroRegras*.
SetAxioma()	Atribui o axioma ao atributo *Axioma*.
GetAxioma()	Função que lê o atributo *Axioma*.
SetRegras()	Atribui o conjunto de regras ao atributo *Regras*.
GetRegras()	Função que lê o atributo *Regras*.
SetNroNiveis()	Atribui o número de níveis ao atributo *NroNiveis*.
GetNroNiveis()	Função que lê o atributo *NroRegras*.
SetDirecoes()	Atribui o número de direções ao atributo *Direcoes*.
GetDirecoes()	Função que lê o atributo *Direcoes*.
SetComentarios()	Atribui os comentários ao atributo *Comentarios*.
GetComentarios()	Função que lê o atributo *Comentários*.
GetPosicaoInicial()	Função que lê a coordenada inicial do atributo *PosicaoInicial*.
SetPreto_e_Branco()	Atribui verdadeiro (preto e branco)/falso (colorido)
GetPonteiro()	Função que lê o ponteiro inicial do axioma do atributo *DescAxioma*.

A classe de interface possui apenas um atributo (global) que armazena o ponteiro para o primeiro símbolo do axioma expandido. Os métodos estão documentados na tabela7 .

Tabela 7 – Documentação dos métodos da classe de interface.

Método	Comportamento
Ler_Definicao()	Lê uma definição de um arquivo de formato 2SL.
Gravar_Definicao()	Grava uma definição Sistema-L em um arquivo de formato 2SL.
Limpar_Tela()	Limpa a tela para uma nova imagem ou iteração.
Interpretar_e_Desenhar()	Interpreta o axioma e desenha a imagem.
Gravar_Imagem()	Grava a imagem em formato BMP.
Gravar_Axioma_Expandido()	Grava o axioma expandido em arquivo de formato TXT.
Copia_Img_Para_ClipBoard()	Copia a imagem para o clipboard.
Imprimir_Imagem()	Imprime a imagem independentemente de dispositivo.
Imprimir_Definicao()	Imprime a definição Sistema-L atual.

Em resumo, depois de ser passado a definição Sistema-L, a altura e largura do plano onde vai ser desenhada a imagem e se a imagem vai ser colorida ou não, o método a ser chamado é o *Expande_Axioma()*. Este por sua vez chama os métodos *Encadeia_Axioma()*, *Reescrever_Sistema_D0L()* ou *Reescrever_Sistema_Estocastico()*. Após o axioma ser expandido, é calculado o tamanho da imagem pelo método *Calcula_Tamanho_Imagem()*. O método *Atualiza_Status_Tartaruga()* é chamado por *Calcula_Tamanho_Imagem()* e *Interpretação_Grafica()* que desenha a imagem

O diagrama de classe mostrado aqui apresenta duas pequenas imagens representadas por um cadeado e uma barra de cor lilás. Isso significa que os atributos e métodos que possuem um cadeado na frente, são privados à esta classe e só podem ser acessados pela própria classe. Outras classes só podem ter acesso através de métodos implementados. A barra lilás diz que os métodos e atributos são públicos, ou seja, são visíveis por outras classes. Essa é uma convenção adotada pela UML.

Aqui pode-se entender que, um método é uma função ou um procedimento executado por um objeto, que é uma classe estanciada. Um atributo, é uma variável do objeto.

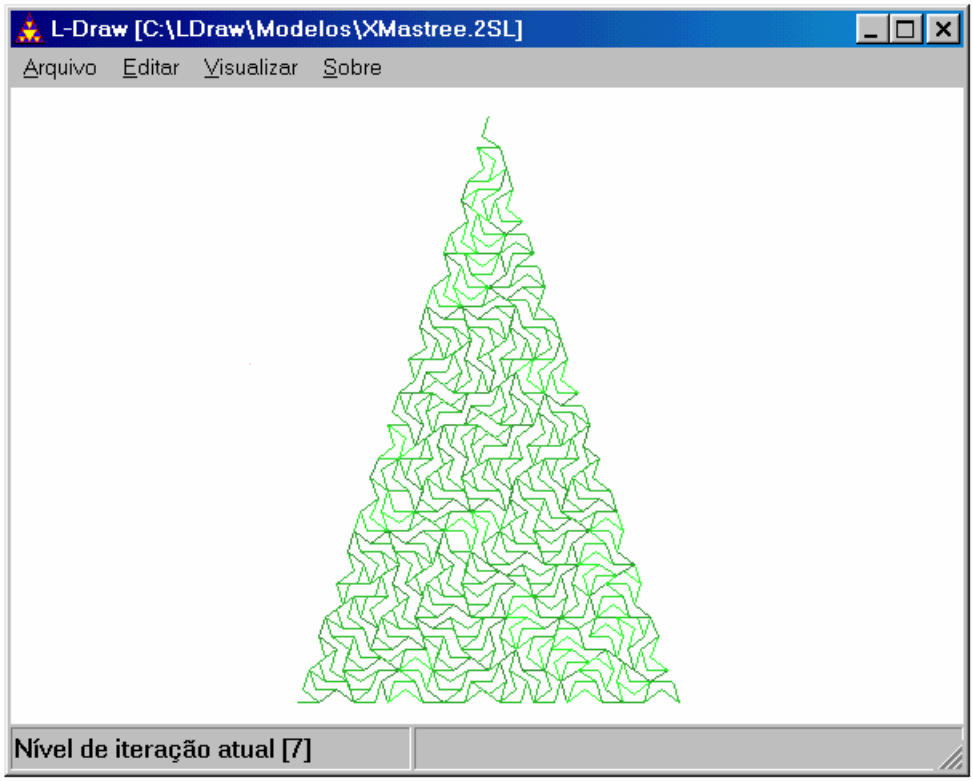
O item a seguir mostra, à nível de usuário, como usar o sistema implementado, bem como as informações básicas para o seu perfeito funcionamento. Todas as telas, suas funcionalidades e opções do menu também serão mostrados.

4.4 USANDO O SISTEMA IMPLEMENTADO

O protótipo, nomeado de L-Draw, é fácil de ser usado e foi implementado de modo a permitir que qualquer tipo de usuário possa usar no estudo da reescrita paralela ou para, simplesmente, brincar com as imagens, pois são atraentes em suas formas.

A Figura 46 apresenta a tela principal e suas opções de menu, onde na parte superior da tela é mostrado um modelo que foi aberto de um arquivo chamado XmasTree.2LS contendo uma definição Sistema-L para formação da imagem que semelhante à uma árvore de natal. Todas as telas demonstradas deste ponto em diante, considerará as configurações e definição contidas neste arquivo. O nível de iteração sendo utilizado no momento e o nome do arquivo com o caminho completo são mostrados claramente como parte da interface do L-Draw.

Figura 46 – Tela principal do L-Draw.



As opção de menu do L-Draw são as seguintes: Arquivo, Editar, Visualizar e Sobre.

Arquivo

As opções do menu Arquivo está relacionadas aos processos de gravação e impressão

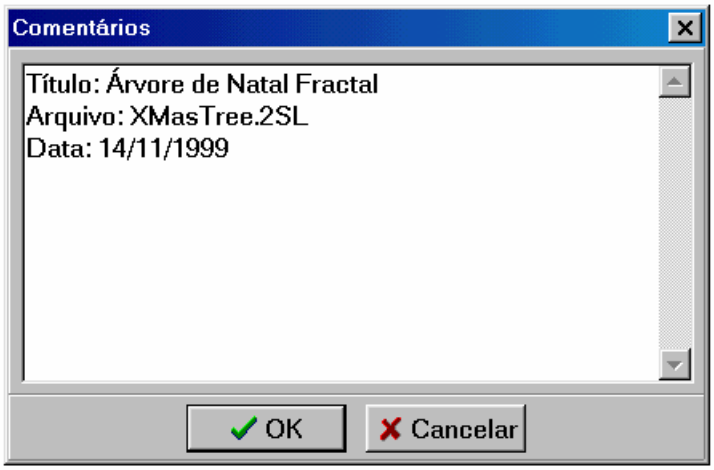
- Novo (Ctrl+N): prepara o L-Draw para que seja informado uma nova definição Sistema-L;
- Abrir (F3): abre um arquivo no formato 2SL (formato do L-Draw) contendo uma definição Sistema-L;
- Salvar: grava uma definição Sistema-L em arquivo de formato 2SL;
- Salvar Regras Como: grava uma definição Sistema-L com outro nome de arquivo de formato 2SL;
- Salvar Imagem: grava a imagem gerada em arquivo de formato BMP. O tamanho da imagem é o mesmo tamanho da tela;
- Salvar Axioma Expandido: permite que seja gravado o axioma expandido em arquivo de formato texto, possibilitando um acompanhamento da evolução da reescrita nas primeiras iterações;
- Imprimir Imagem: permite que a imagem seja impressa em qualquer formato de papel;
- Imprimir Definição: imprime uma definição Sistema-L;
- Sair: encerra o L-Draw.

Editar

As opções do menu Editar estão relacionadas a inserção, exclusão e alteração de uma definição Sistema-L e estão apresentadas à seguir.

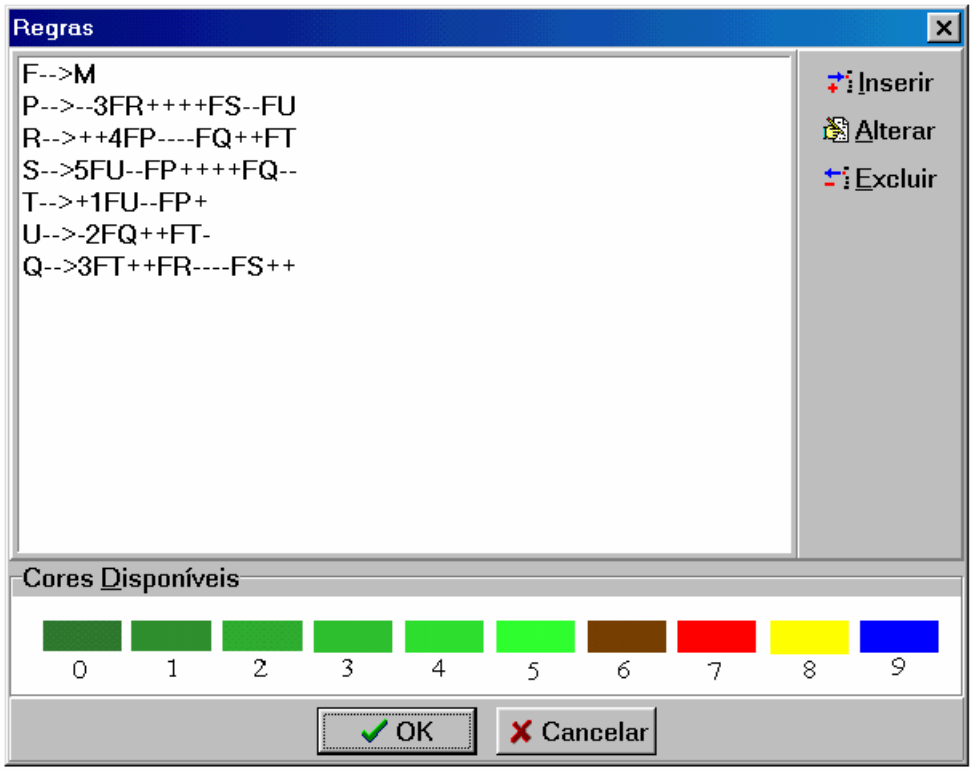
- Comentários: possibilita que sejam gravados comentários a respeito da definição. A tela está representada na Figura 47.

Figura 47 – Comentários na definição.



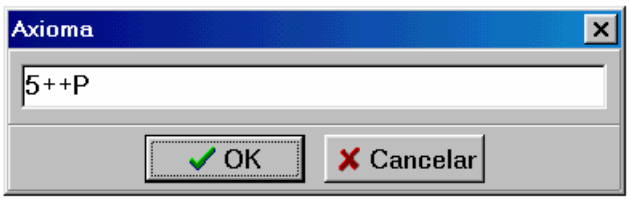
- Regras: nesta opção, representada na Figura 48, pode-se inserir, alterar ou excluir uma regra do conjunto de regras. É preciso prestar muita atenção quando informar o conjunto de regras. Uma regra sempre é formada por um antecessor e um sucessor que são separados por dois sinais de subtração e um sinal de maior. O antecessor é o caracter que está antes dos símbolos --> e o sucessor é a cadeia de caracteres que vem depois. Para modelos estocásticos deve-se informar a probabilidade entre parênteses e a soma das probabilidades deve ser igual a 100.

Figura 48 – Inserção, alteração ou exclusão de regras de produção.



- Axioma: altera ou inseri um axioma. Lembrando que um axioma deve, necessariamente, conter no mínimo uma letra e esta deve possuir um (Sistema-L Determinístico) ou vários (Sistema-L Estocástico) antecessor correspondente no conjunto de regras. Pode-se utilizar os sinais de + ou – para posicionar a imagem na tela no local desejado. A Figura 49 mostra a tela para informação do axioma.

Figura 49 – Informação do axioma.



- Direções: altera ou inseri um incremento angular. Se for definido que a tartaruga gráfica deva virar 36º cada vez que encontrar o símbolo + ou -, então deve-se dividir 360º graus por 36º resultando em 10. O incremento angular diz que a tartaruga gráfica pode fazer no máximo 10 viradas de 36º, totalizando 360º. O número de direções é informado na tela representada pela Figura 50.

Figura 50 – Informação do incremento angular.

- Copiar Imagem Para o Clipboard: copia a imagem para o clipboard para ser utilizada em outro ambiente;

- Cor do Fundo: permite trocar a cor de fundo. Isso é necessário devido às cores da imagem às vezes não se destacarem com a cor de fundo branca. Um fundo preto, por exemplo, realça melhor as imagens que têm amarelo ou verde claro.

Visualizar

As opções do menu Visualizar são as que dão a interação do usuário com o L-Draw na apresentação do resultado, que é a imagem. Estão apresentados abaixo.

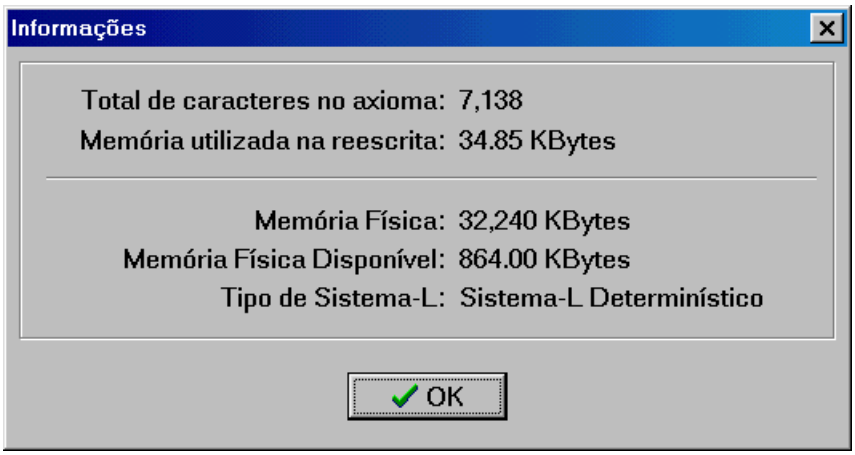
- Próximo Nível (Ctrl+P): mostra o próximo nível de iteração de reescrita. Deve-se prestar atenção antes de utilizar essa opção devido ao fato de o axioma aumentar exponencialmente o seu tamanho, consumindo todo recurso do sistema. Na dúvida é melhor ir na opção *Informações*;

- Nível Anterior: mostra o nível imediatamente anterior ao atual;

- Nível: é dada uma lista de 0 à 16 níveis de iterações para que seja escolhido diretamente, sem observar a evolução da imagem. É aconselhável que se utilize, dependendo da definição, até o nível 7 pelo mesmo motivo da opção anterior;

- Informações: mostra informações de status do L-Draw. Quando já tiver sendo utilizado nível maior que 2, dependendo da definição, é necessário consultar para evitar que todo recurso dos sistema seja consumido pelo L-Draw. A Figura 51 mostra a quantidade de caracteres que formam o axioma expandido e o total, em Kbytes, de memória necessária para armazená-lo. Mostra também, quanto de memória há no sistema e quanto ainda há disponível. A partir do momento que o total disponível for zero, deve-se ter cautela nos próximos níveis de iteração.

Figura 51 – Tela de informações do L-Draw.



- Definição: apresenta a definição Sistema-L sendo utilizada no momento. Não mostra o tipo de Sistema-L. Isso já e reconhecido pelo L-Draw. Basta que um caracter no axioma tenha vários antecessores no conjunto de regras para ser estocástico, do contrário é determinístico.

- Imagem em preto e branco: é uma opção que retira os caracteres de cores (números) tornando o processo mais rápido, com isso a imagem ficará em preto e branco.

Sobre

A opção Sobre não possui funcionalidade. Apresenta apenas uma tela com informações gerais como detalhes do curso e título do trabalho feito. Mostra também o ícone do L-Draw, versão e autor.

E obrigatório a informação do axioma, das regras e do incremento angular. Não é possível gerar imagens na falta de um desses elementos. Os números que estão juntos com as regras são números que correspondem a um índice em uma pequena paleta de cores que está representada na Figura 40.

4.5 RESUMO

O protótipo, nomeado de L-Draw, é baseada em reescrita paralela, um processo proposto por Aristid Lindenmayer que utiliza caracteres formais e regras de substituição para geração de estruturas fractais. Apresenta opções como gravar, imprimir e interação com processo através de uma interface amigável.

Foi utilizado, na implementação, o ambiente Delphi 3.0, que por sua vez utiliza a linguagem Object Pascal como linguagem de programação. A ferramenta utilizada para a modelagem foi Rational Rose 4.0.

Com o objetivo de tornar mais dinâmico a utilização da parte computacional propriamente dita, o L-Draw foi dividido em duas camadas: interface e serviço. Cada camada foi implementada como uma classe que se comunicam através de mensagens. Uma abordagem orientada a objetos foi utilizada através da linguagem de modelagem UML (*Unified Modeling Language*) .

Evitou-se utilizar comandos de manipulação de cadeia de caracteres oferecidos pela linguagem uma vez que pretendia-se um ganho de performance.

O L-Draw apresenta uma interface de fácil navegabilidade e proporciona todas as informações necessárias ao usuário sem muito esforço. A manipulação das entradas e saídas, bem como a interação com o processo é de forma fácil.

5 ANÁLISE DOS RESULTADOS E COMENTÁRIOS

São apresentados neste capítulo várias imagens geradas através do emprego de técnicas de reescrita paralela e interpretação gráfica. São apresentadas estruturas arbóreas determinísticas e estocásticas, curvas de preenchimento, curvas fractais, formas matemáticas abstratas e outras imagens.

5.1 ANÁLISE DAS IMAGENS GERADAS

Uma forte razão que justifica o estudo dos fractais é sua presença abundante nas formas naturais que nos cercam [PEI86]. Ao olhar para uma árvore, não é difícil perceber que um ramo desta árvore é uma versão menor da árvore inteira. Partindo do princípio que, se uma parte de um objeto pode representar o todo, exatamente ou estatisticamente, é porque esse objeto possui auto-similaridade, característica básica de um fractal [PEI88].

Durante muitos séculos não conseguiram descrever como uma planta cresce. O que governa o crescimento dos organismos vivos não é uma fórmula e sim uma regra de formação [PRU90].

Um fato verdadeiro é que Sistemas-L exploram a auto-similaridade, propriedade básica dos fractais. Muitos objetos fractais foram propostos por Hilbert, Peano, Sierpinski, Cantor e Koch e, que através de regras de reescrita são facilmente desenhados. Tais objetos são apresentados através de um formalismo matemático que se utiliza de caracteres para representar a formação destes objetos, ao invés de utilizar fórmulas matemáticas complexas.

O protótipo implementado possibilita a criação de várias imagens com duas dimensões tais como: plantas, fractais, curvas de preenchimento, texturas e outras variações. O objetivo não é modelar uma planta tal como aparece na natureza, e sim sua formação topológica e ramificação.

5.1.1 PLANTAS

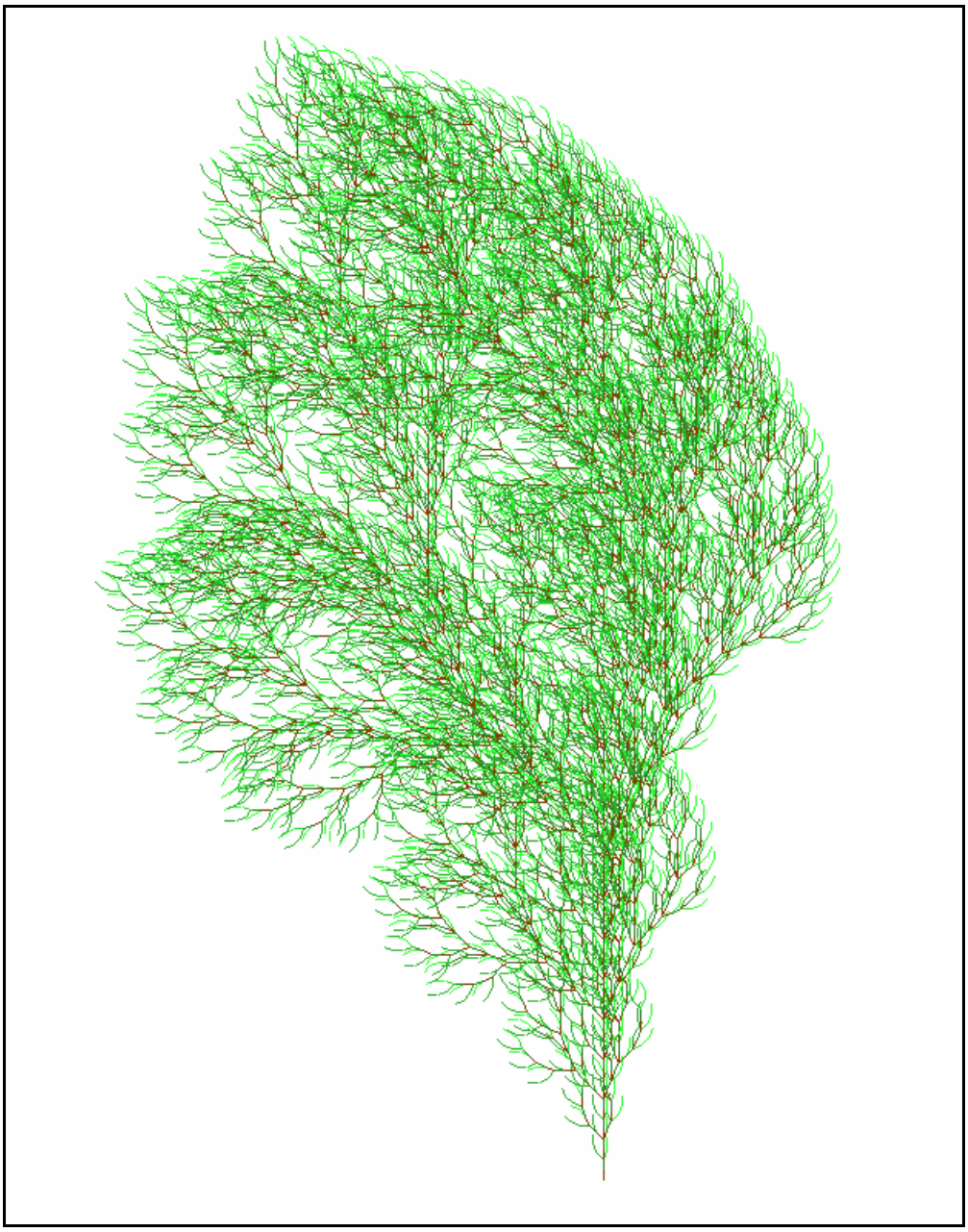
A razão pela qual foi criada a reescrita paralela é a modelagem de plantas. É mostrado neste item como um pequeno conjunto de informação pode criar imagens tão complexas que nos lembram formas naturais.

As primeiras imagens são geradas utilizando Sistema-L Determinístico. Uma característica é que só pode haver uma regra de produção para cada símbolo do axioma. Essas imagens sempre exibirão um formato artificial quando agrupadas. Um jardim formado por plantas geradas utilizando-se Sistema-LD0, possui todas as plantas iguais. No processo de reescrita não é utilizado nenhum processo de randômico de regras ou direções.

Em seguida são mostrados alguns modelos gerados através de Sistema-L Estocástico. Nesse processo de reescrita é utilizado uma estatística. Nessa definição pode haver várias regras para cada símbolo do axioma. O somatório das probabilidades de cada regra deve ser igual à 100. Como é feito um sorteio de qual regra deverá ser aplicada, cada vez que for gerado um modelo estocástico, uma imagem diferente é gerada. Essa idéia permite a criação de um jardim de plantas de mesma família, porém, com características próprias de cada uma.

A Figura 52 é uma imagem clássica dos livros sobre fractais. Foi gerada através de Sistema-D0L agrupado com 5 iterações de reescrita. Através da visualização desta imagem, se não há conhecimento prévio, não é possível perceber que foi gerada tendo como princípio um conjunto de informações, a definição [PEI88] Sistema-L que a segue, é realmente bem pequena.

Figura 52 – Modelo determinístico com agrupamentos (Modelo 01).

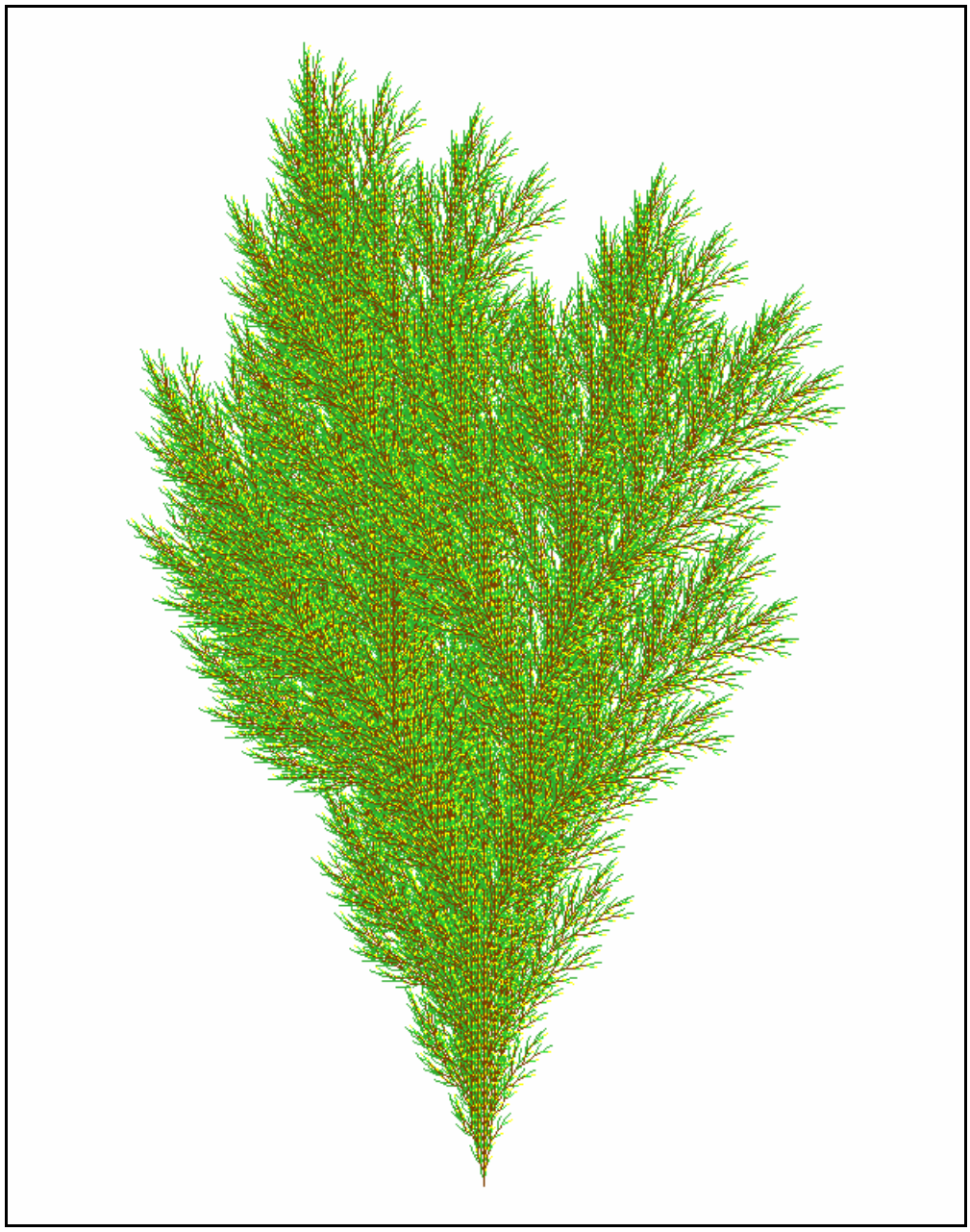


Direções	**16**
Axioma	**F**
Regra	p_1 : **F→6F0F+[+2F-F-F]-[-4F+5F+F]**

O número de direções igual à 16 diz que o ângulo de desenvolvimento da estrutura arbórea mostrada na Figura 52 é de, aproximadamente, 22º, pois 360/16 = 22.5.

Uma pequena modificação na única regra da definição Sistema-L que gerou a Figura 52, e 6 iterações de reescrita, resulta em uma imagem totalmente diferente. O resultado pode ser visto na Figura 53. A pequena diferença pode ser analisada na definição Sistema-L que a segue a imagem.

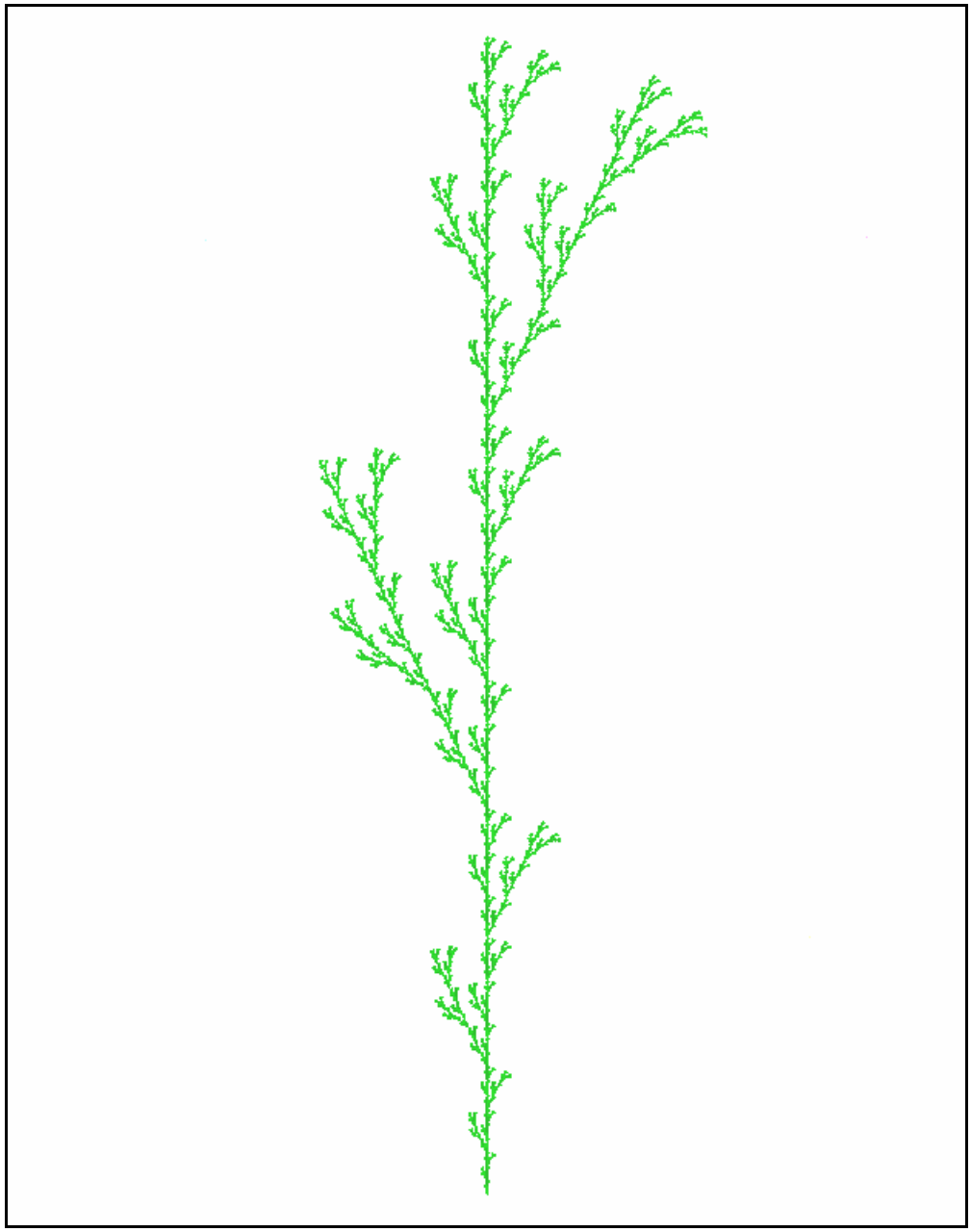
Figura 53 - Modelo determinístico com agrupamentos (Modelo 02).



Direções	**16**
Axioma	**F**
Regra	p_1 : **F→6FF+[2FF-FF]-[-4F+F-F]**

Com uma definição [PEI88] Sistema-L mais simples e 8 iterações de reescrita, a Figura 54 possui uma ramificação com, aproximadamente, 25º, mais aberta que a estrutura mostrada na Figura 53.

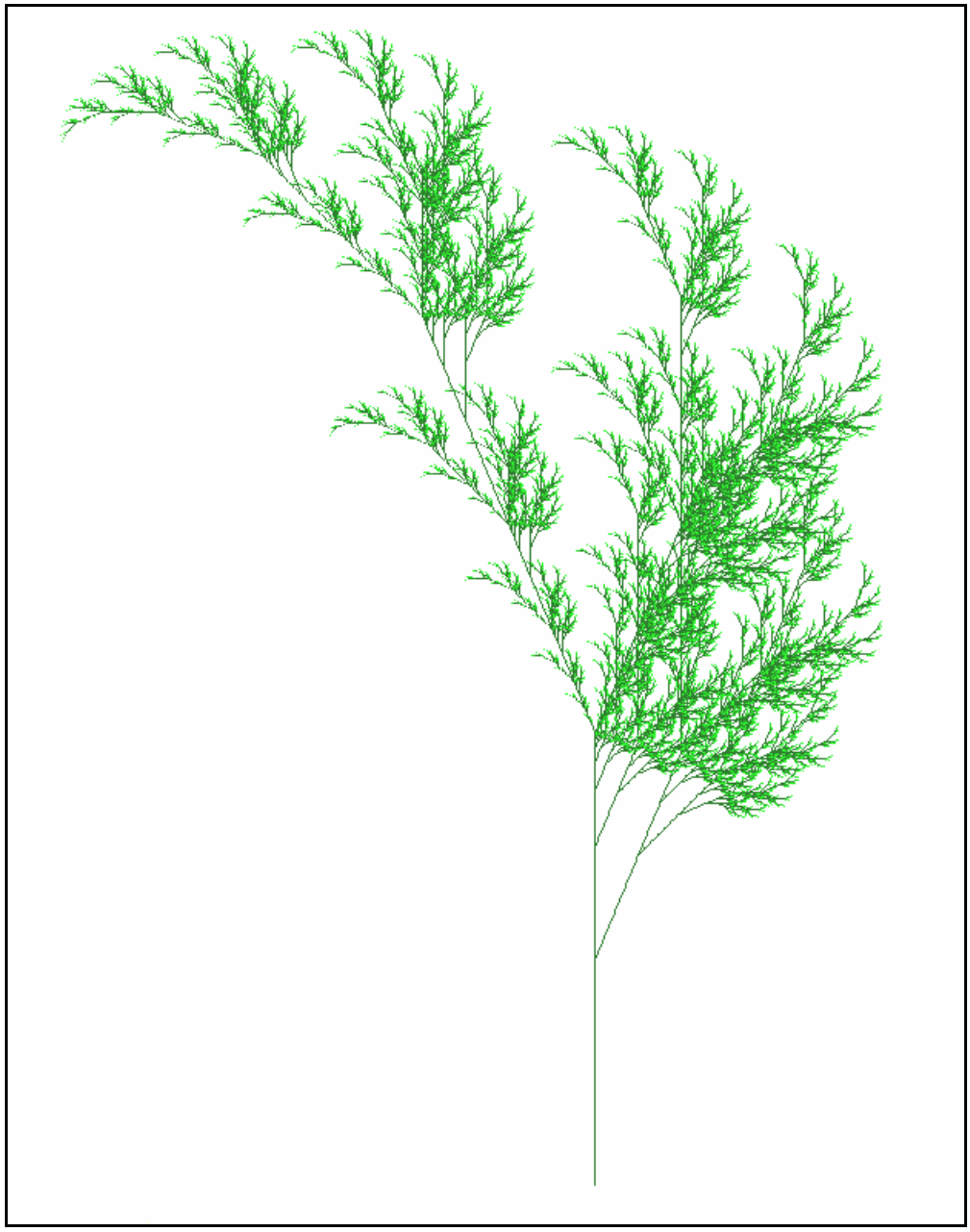
Figura 54 - Modelo determinístico com agrupamentos (Modelo 03).



Direções	**14**
Axioma	**F**
Regra	p_1 : **F→F[+F]F[-F]F**

É fácil perceber o conceito de auto-similaridade através da Figura 55. Sua definição [NIE99] Sistema-L segue a figura.

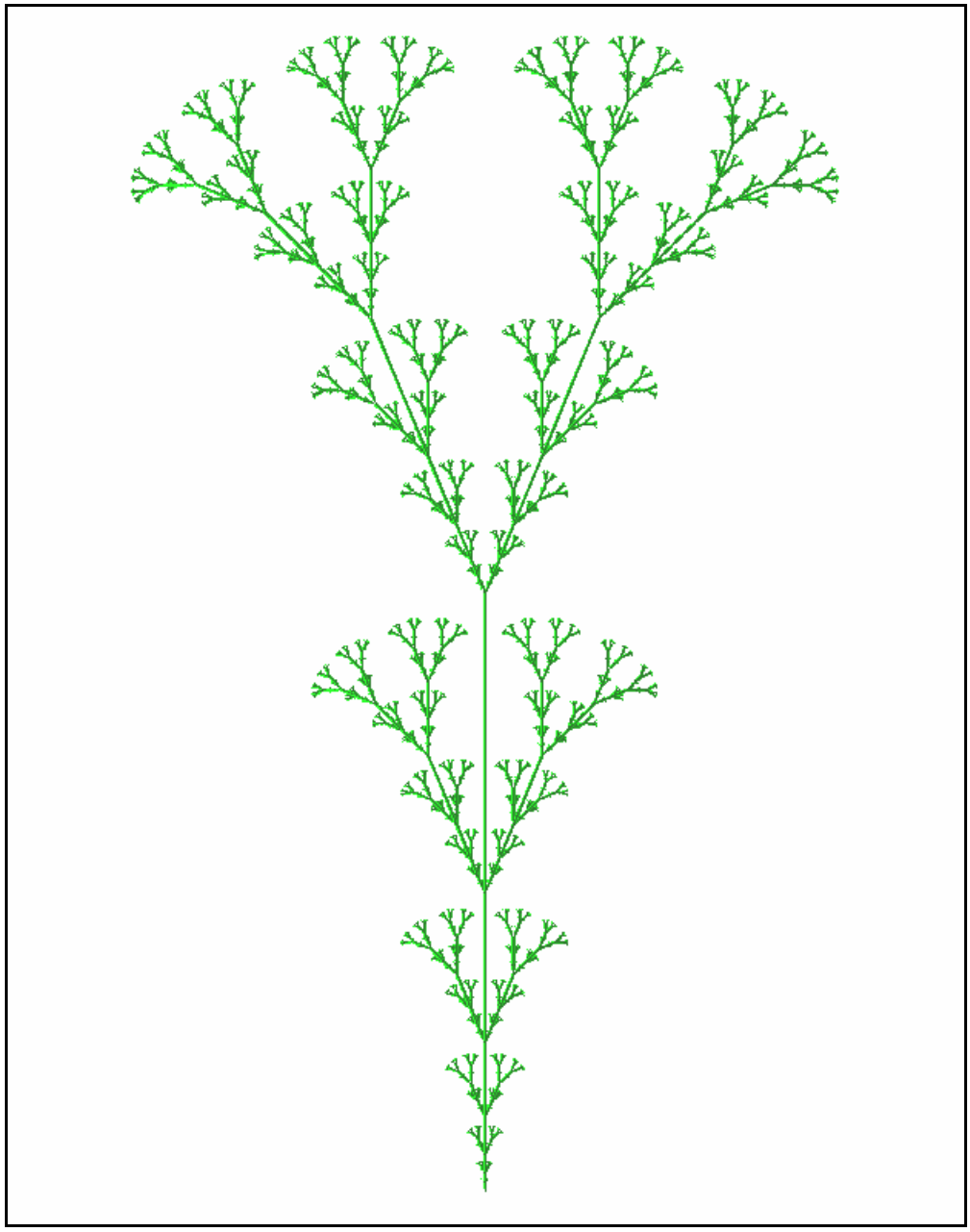
Figura 55 - Modelo determinístico com agrupamentos (Modelo 04).



Direções	**16**
Axioma	**X**
Regras	p_1 : **F→0FF** p_2 : **X→3F-[[X]+X]+4F[+5FX]-X**

Com 9 iterações de reescrita é possível gerar a imagem da Figura 56. Sua definição [PEI88] Sistema-L segue a figura.

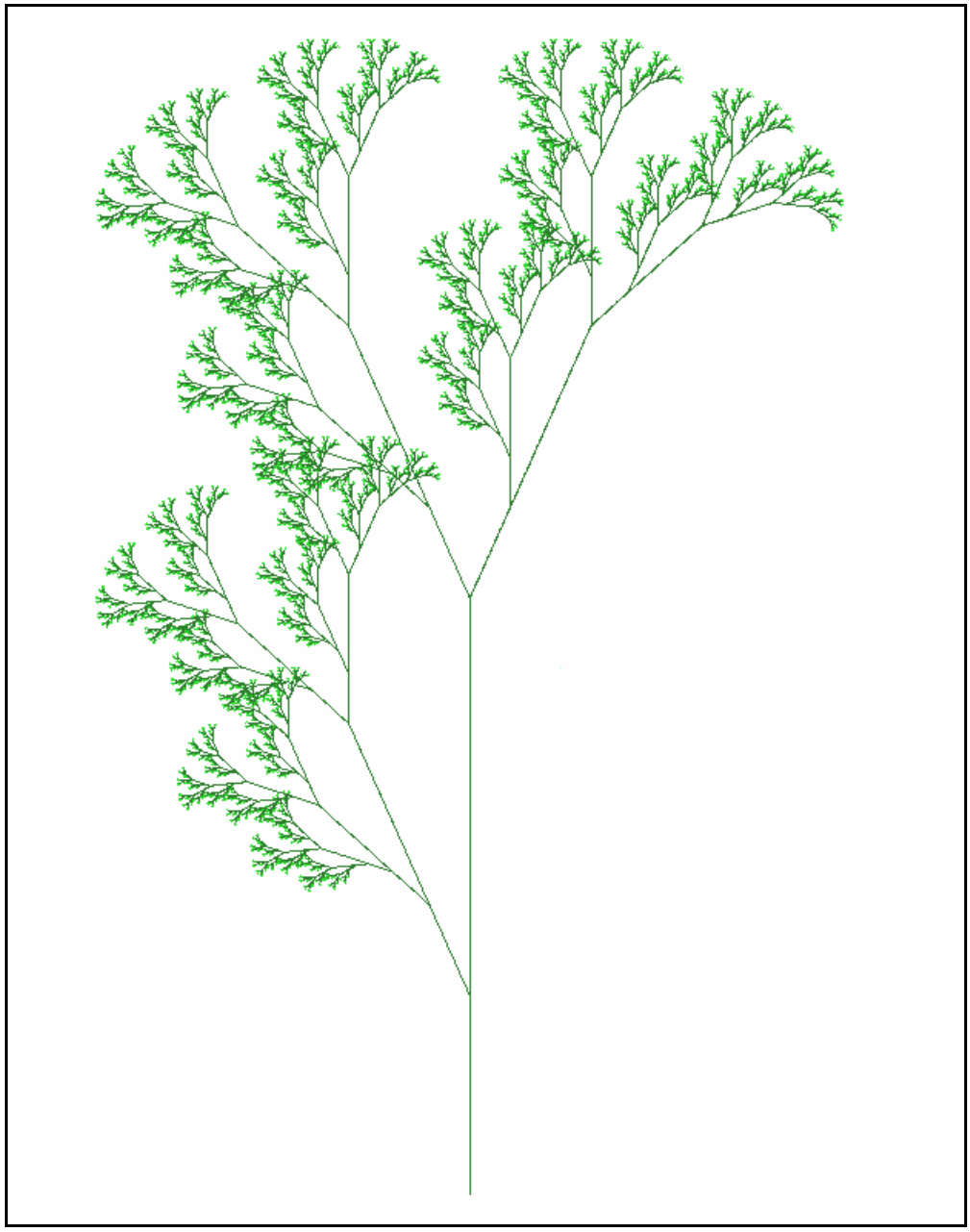
Figura 56 - Modelo determinístico com agrupamentos (Modelo 05).



Direções	**16**
Axioma	**G**
Regras	p_1 : **G→G6FX[+G][-G]5** p_2 : **X→X2[-FFF][+5FFF]6FX**

A Figura 57 foi gerada em 9 iterações e utilizando a definição Sistema-L da Figura 52 com algumas modificações.

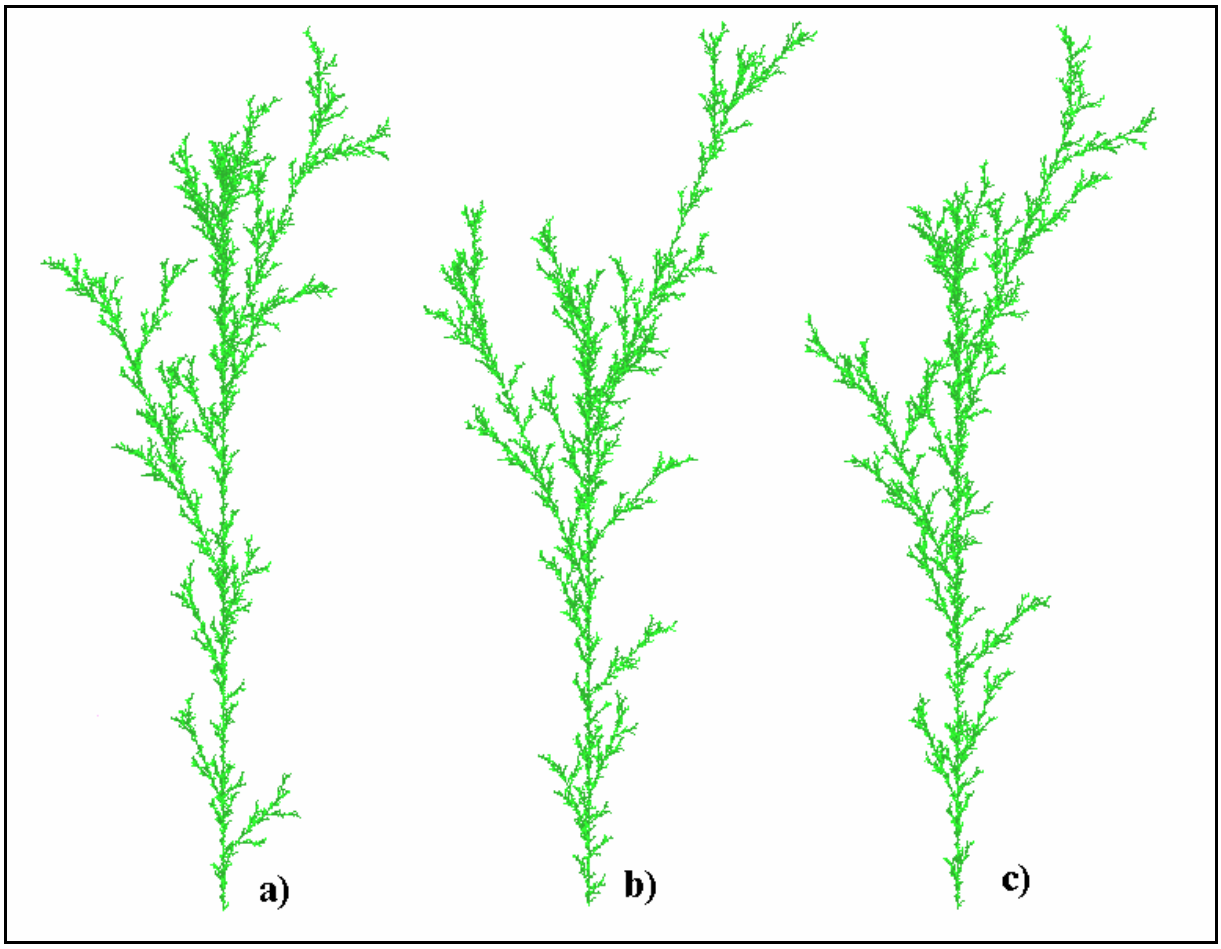
Figura 57 - Modelo determinístico com agrupamentos (Modelo 06).



Direções	**15**
Axioma	**X**
Regras	p_1 : **X→2F[+X]4FF[-X]+X** p_2 : **F→0FF**

A duas imagens seguintes mostram resultados de Sistema-L Estocástico que modela topologicamente a formação de estruturas arbóreas naturais. A Figura 58 apresenta três imagens que preservam suas características como espécie e variam detalhes particulares de cada uma.

Figura 58 – Estrutura arbórea estocástica em 5 iterações.



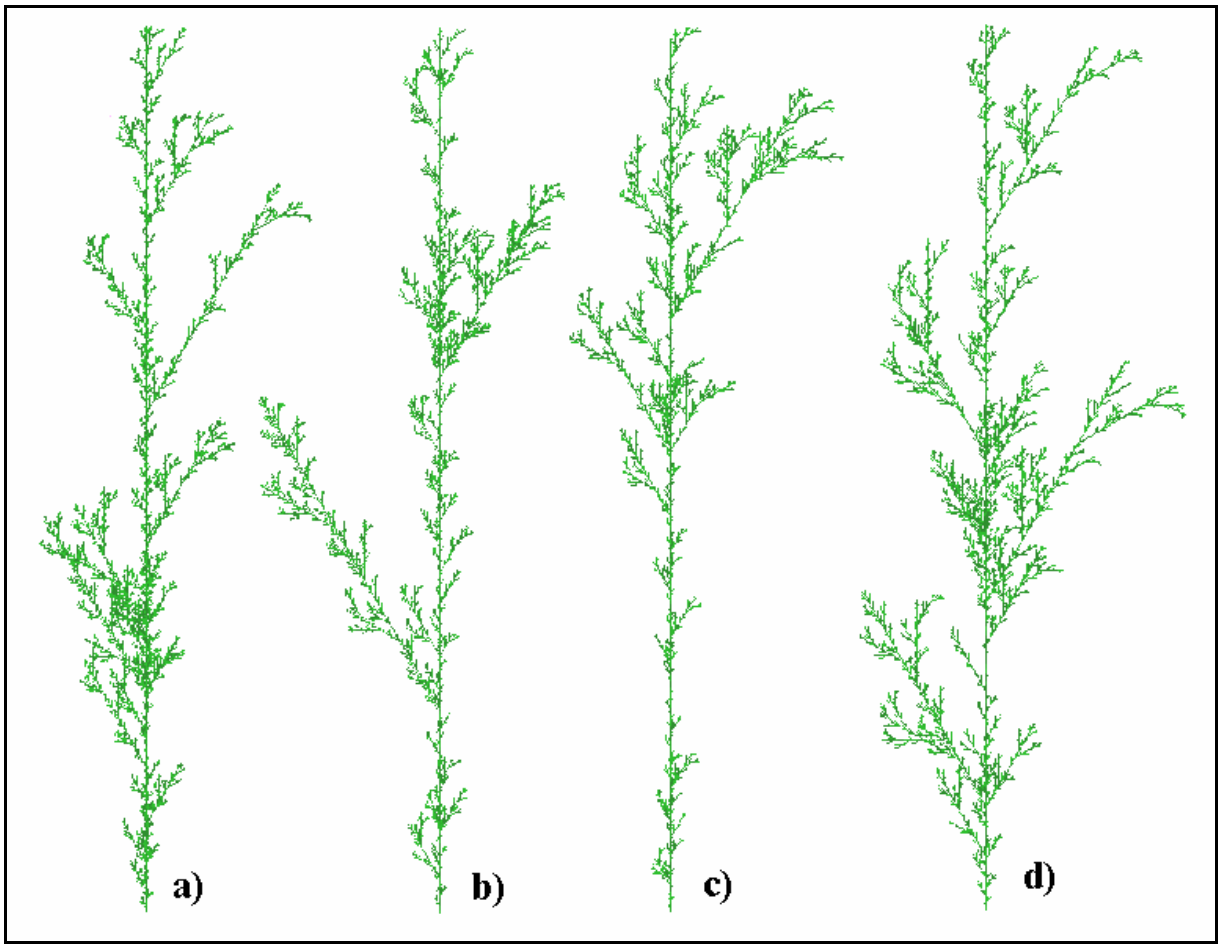
Direções	**16**
Axioma	**++++F**
Regras	p_1 : **F→(60)4F-[-FF]+5F[+FF]F** p_2 : **F→(40)2FF[+FF]3F[-FF][F]F**

A definição da para as imagens da Figura 58 apresentam duas regras de formação, sendo que a regra p_1 tem 60% de chance de ocorrer (o número entre parênteses indica a

probabilidade de ocorrer a regra). Já a regra p_2 possui 40% de chance de ocorrer. Lembrando que a soma entre as probabilidades deve ser 100.

A Figura 59 possui quatro imagens que exibem a mesma situação da Figura 58 quanto ao visual.

Figura 59 - Estrutura arbórea estocástica em 7 iterações.



Direções	**12**
Axioma	**+++F**
Regras	p_1 : **F→(40)1F[+F]F[-F]F** p_2 : **F→(30)2FF** p_3 : **F→(30)3F[+F]F[-F][F]**

A diferença está no ângulo de ramificação e na quantidade de regras. A Figura 59 apresenta uma definição com três regras, sendo que a regra p_1 tem 40% de chance de acontecer, as regras p_2 e p_3 têm, cada uma, 30% de chance de ocorrer. A combinação de tons de verde e a utilização de agrupamentos deixam a imagem mais realística. Considerando a definição Sistema-L estocástica da figuras 53 e 54, observa-se que as imagens geradas apresentam suaves diferenciações nas características individuais, mas suas características como espécie não mudam.

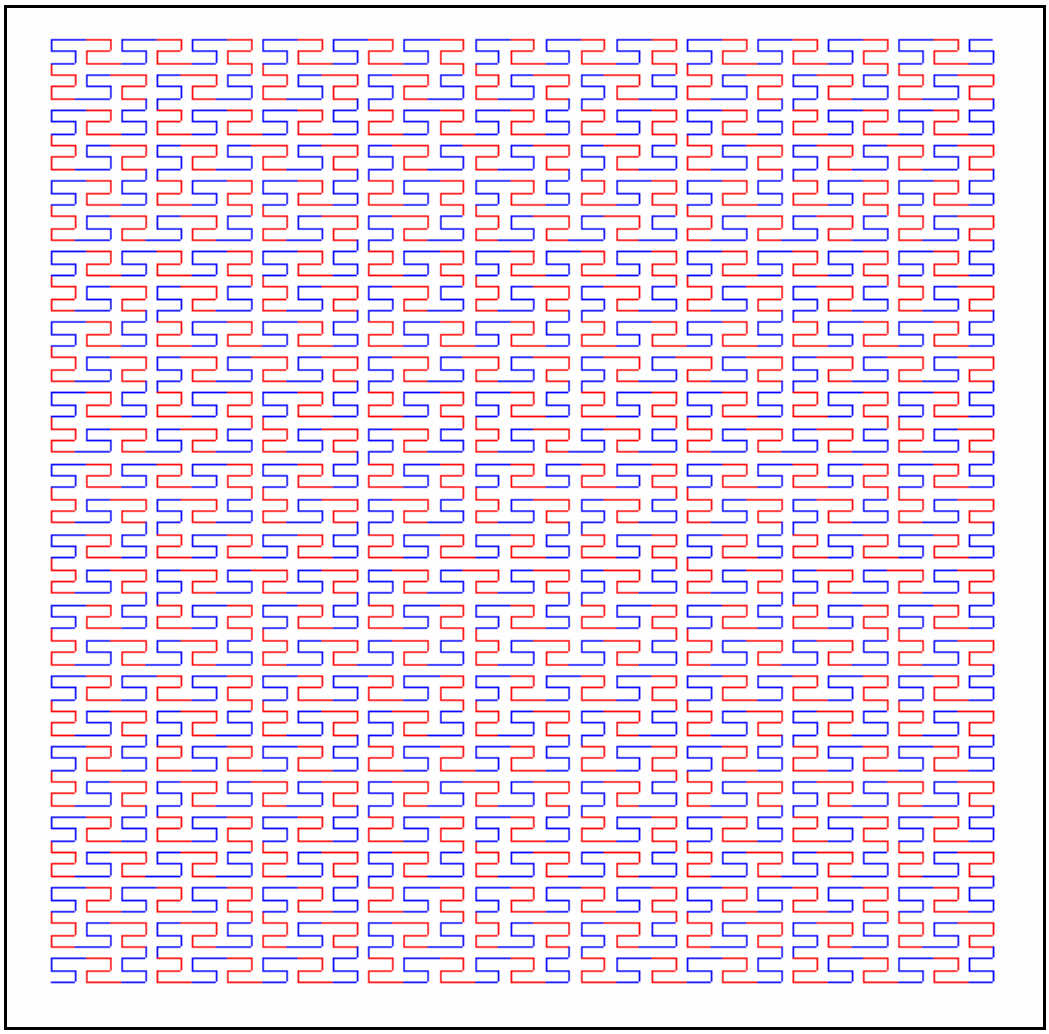
Um melhoramento destes modelos estocásticos seria randomizar o incremento de ângulo. Especificar uma faixa angular entre os quais se comportaria o desenvolvimento da estrutura. Também está limitado aqui a utilização de apenas um caracter (**F**) no axioma. A utilização de várias letras no axioma correspondente com uma ou várias regras de produção produziria imagens mais complexas e formas mais diferenciadas.

5.1.2 CURVAS DE PREENCHIMENTO

Muitas curvas de preenchimento possuem características e propriedades que não estão nos formalismos matemáticos de hoje em dia. Assim como muitos outros objetos, as curvas de preenchimento foram tratadas como “Monstros da Matemática” simplesmente por fugirem dos padrões matemáticos e não terem utilidades. Em 1970, Mandelbrot demonstrou sua utilização aplicando-as aos padrões de formação da natureza.

A Curva de preenchimento de Peano, representada na Figura 60 seguida sua definição [PEI88] Sistema-L, foi gerada em 4 iterações de reescrita. Nota-se que há duas regras, cada uma foi configurada com uma cor diferente, assim percebe-se onde aparecem as produções na imagem.

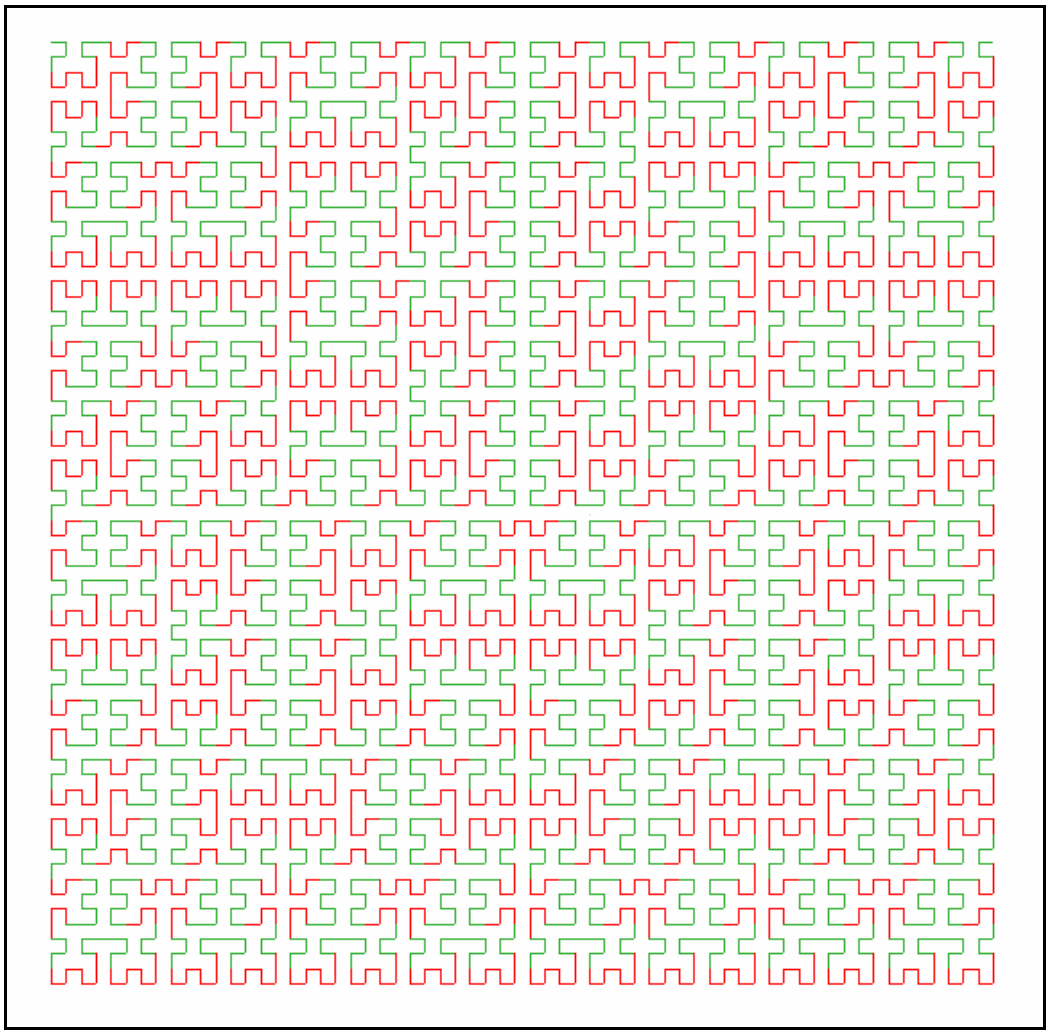
Figura 60 – Curva de preenchimento de Peano em 4 iterações de reescrita.



Direções	**4**
Axioma	**X**
Regras	p_1 : **X→9XFYFX+F+YFXFY-F-XFYFX** p_2 : **Y→7YFXFY-F-XFYFX+F+YFXFY**

A Figura 61 mostra a Curva de preenchimento de Hilbert. Sua definição [PEI88] Sistema-L segue a figura. Nota-se que também há duas regras com cores diferentes para que seja melhor o entendimento da aplicação das regras de produção.

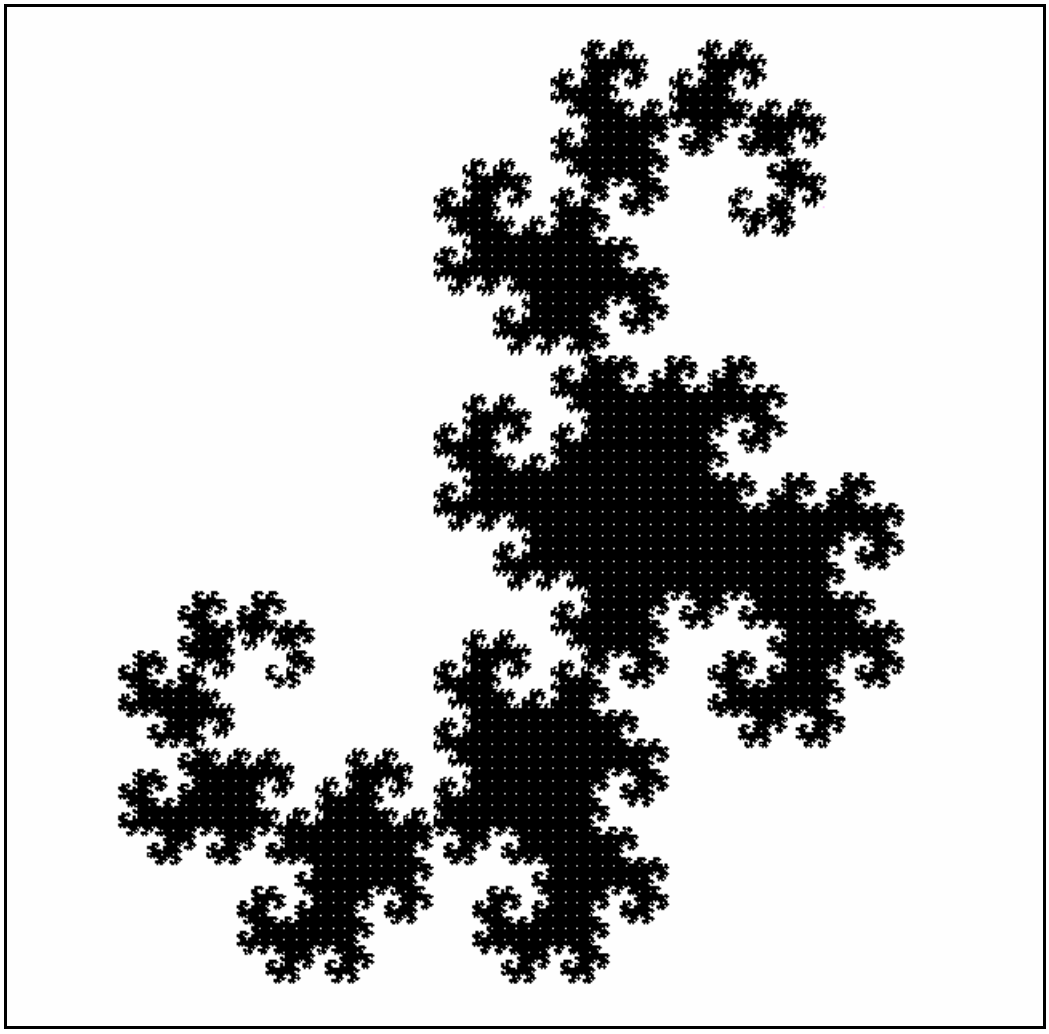
Figura 61 – Curva de preenchimento de Hilbert em 6 iterações de reescrita.



Direções	**4**
Axioma	**X**
Regras	p_1 : **X→-7YF+XFX+FY-** p_2 : **Y→+2XF-YFY-FX+**

Uma curva de preenchimento muito famosa é a Curva Dragão. Sua construção é fundamentada em uma seqüência de dobras com 90º em um papel [CRI91]. Sua imagem está representada na Figura 62. Recebeu o nome de “Curva Dragão” devido à semelhança das suas bordas com tradicionais dragões chineses. A imagem foi construída com 17 iterações de reescrita. A definição Sistema-L [PEI88] segue a imagem.

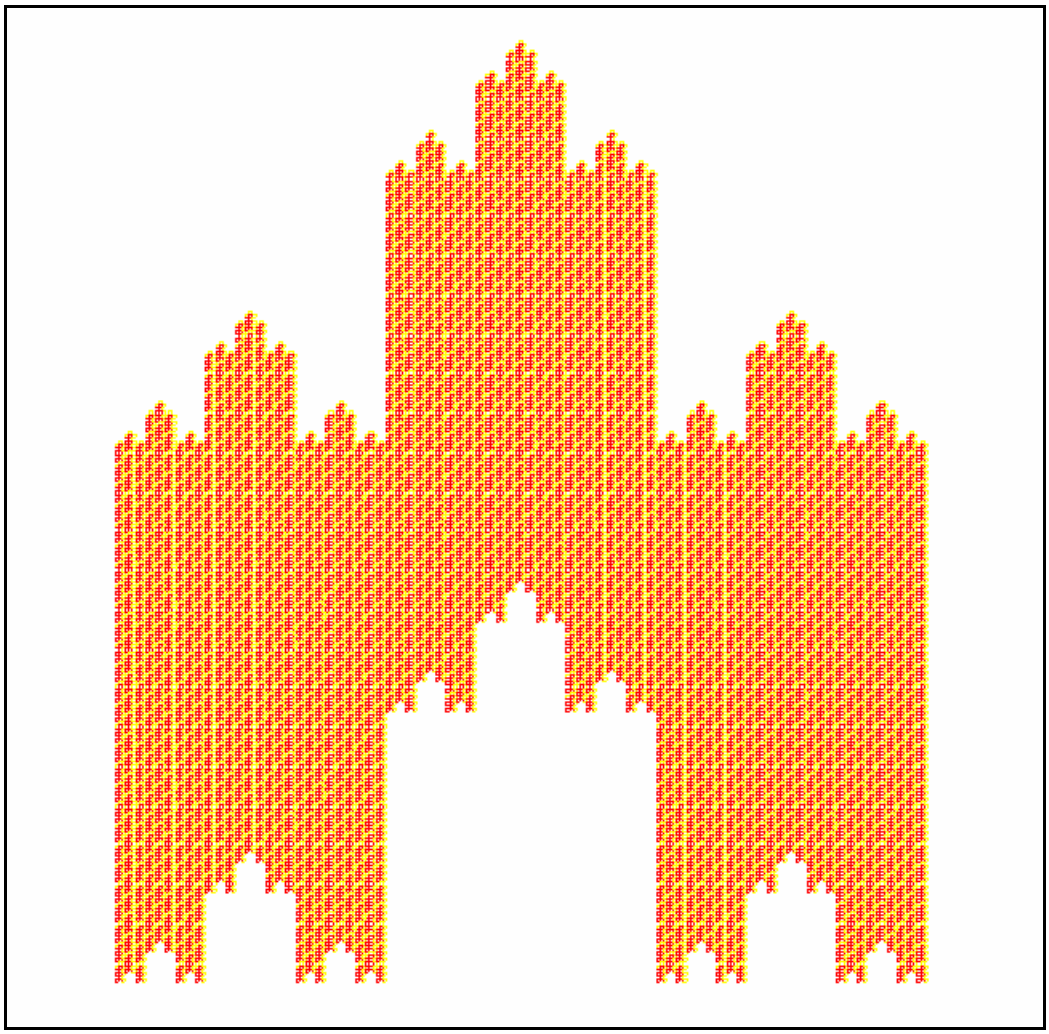
Figura 62 – Curva Dragão.



Direções	**4**
Axioma	**X**
Regras	p_1 : **X→X+YF+** p_2 : **Y→-FX-Y**

A curva de preenchimento de espaço de Dekking, representada na Figura 63 em 9 iterações, foi descoberta recentemente por Dekking. Uma forma que lembra uma igreja. Sua complexa definição [NIE99] Sistema-L segue a imagem.

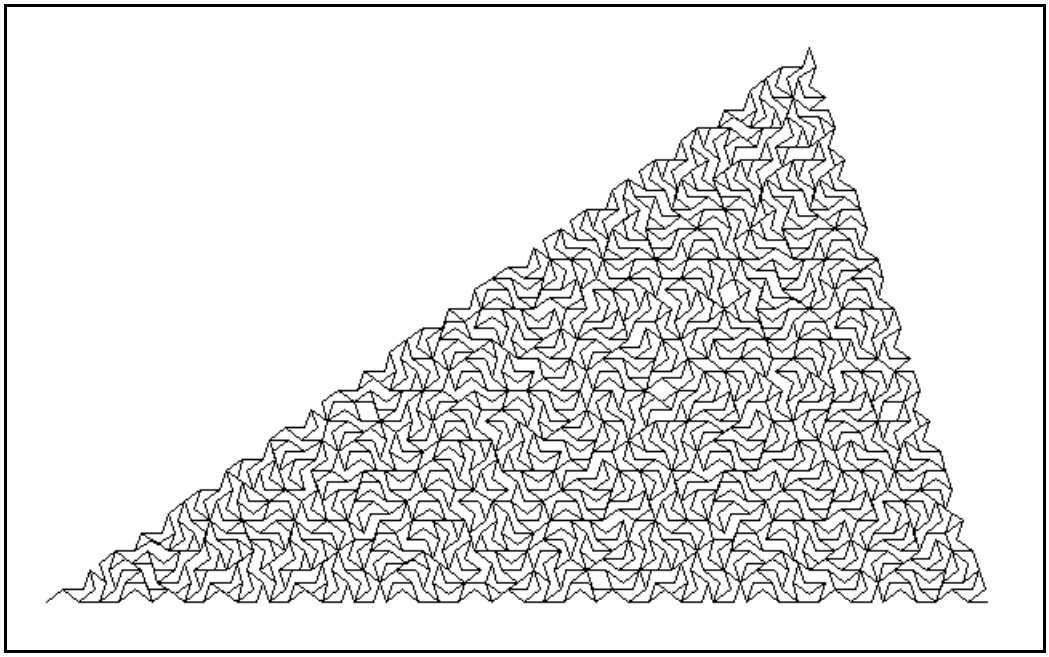
Figura 63 – Curva de preenchimento de espaço de Dekking em 9 iterações de reescrita.



Direções	**4**
Axioma	**X**
Regras	p_1 : **W→7FW+F-XFW-F+Z** p_2 : **X→++F--Y-F+Z++F--Y-F+Z** p_3 : **Y→++8F--Y+F-X** p_4 : **Z→8FW+F-X** p_5 : **F→M**

A imagem da Figura 64, chamada de Pentive, é uma curva de preenchimento baseada em uma textura. Sua complexa definição Sistema-L está apresentada logo após.

Figura 64 – Pentive em 8 iterações.

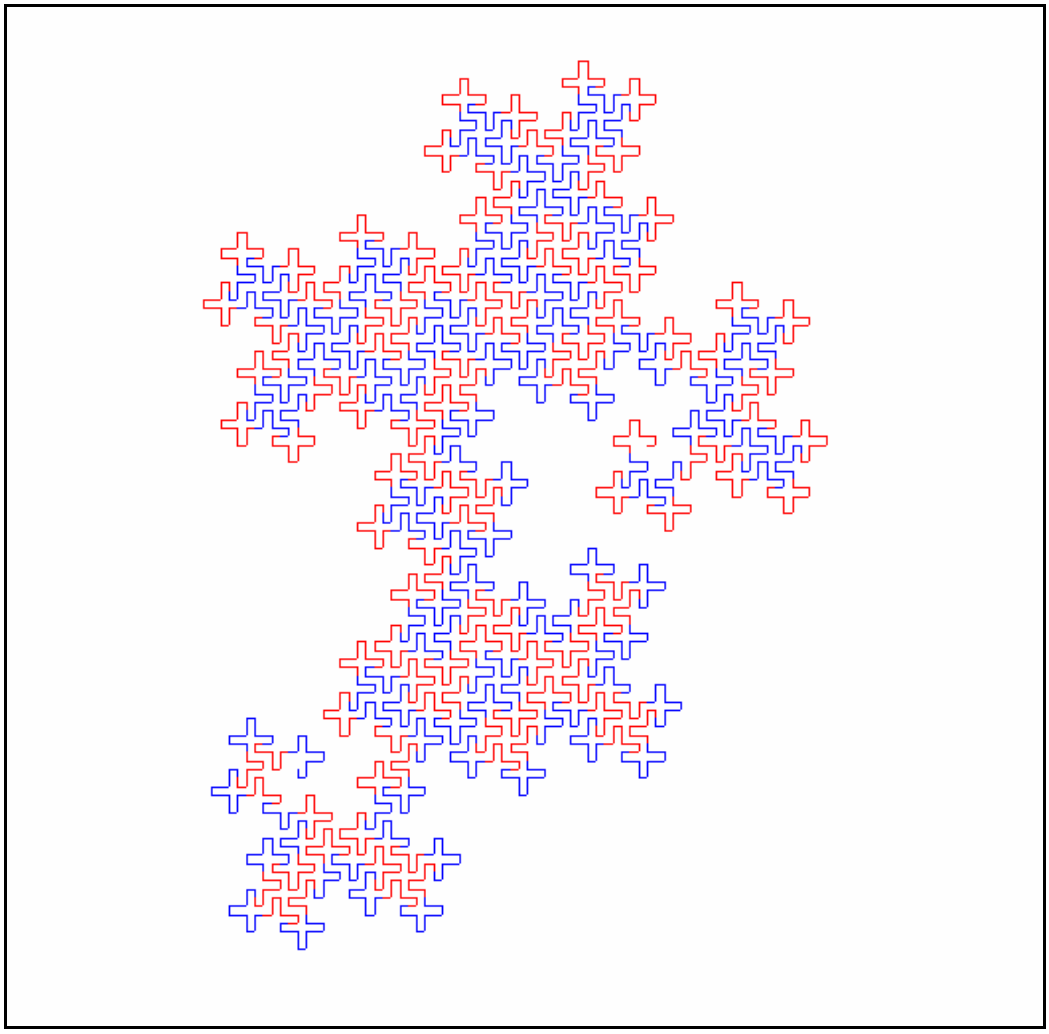


Direções	**10**
Axioma	**Q**
Regras	p_1 : **F→M** p_2 : **P-->--FR++++FS----FU** p_3 : **Q→FT++FR----FS++** p_4 : **R→++FP----FQ++FT** p_5 : **S→FU--FP++++FQ----** p_6 : **T→>+FU--FP+** p_7 : **U→>-FQ++FT-**

A Figura 65 chamada de *Cross* em 5 iterações. Sua definição [NIE99] Sistema-L a segue. É uma extensão da Curva Dragão, representada na Figura 62.

Após a descoberta de muitas estruturas fractais, outras foram criadas através de derivações por outros pesquisadores.

Figura 65 – Cross em 5 iterações.

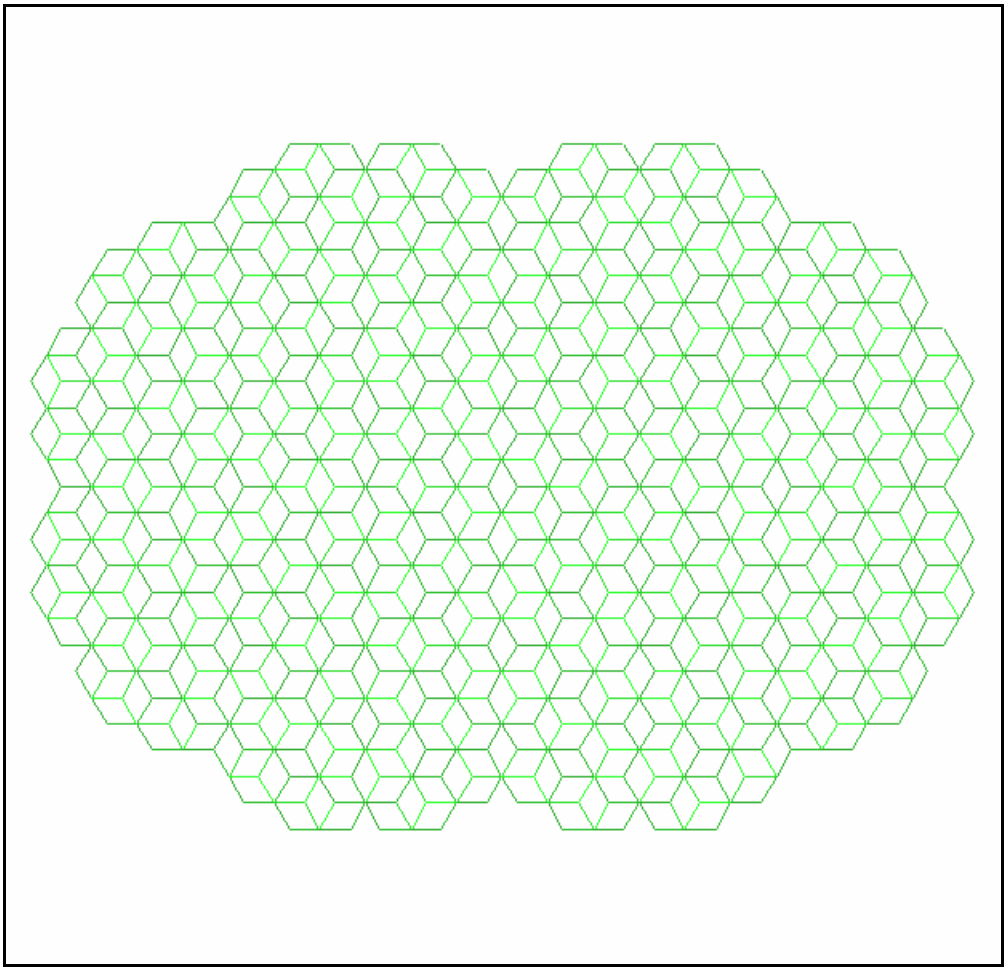


Direções	**4**
Axioma	**FX**
Regras	p_1 : **X→7FX+FX+FXFY-FY-** p_2 : **Y→+9FX+FXFY-FY-FY** p_3 : **F→M**

5.1.3 LADRILHOS (*TILLINGS*)

Uma das maneiras de gerar ladrilhos é dissecar a figura em pequenas cópias de si mesmo. Um ladrilho é uma subdivisão de uma região em sub-regiões não sobrepostas, onde cada sub-região é uma cópia de um conjunto finito de formas. Os ladrilhos, um dos objetos mais antigos da matemática, foram vistos como Sistemas Dinâmicos há pouco mais de 20 de anos, pois encontraram muitos aspectos dinâmicos nesta teoria [WRI96]. Um exemplo é mostrado na Figura 66. Sua definição [NIE99] Sistema-L é bem simples perto do que pode produzir.

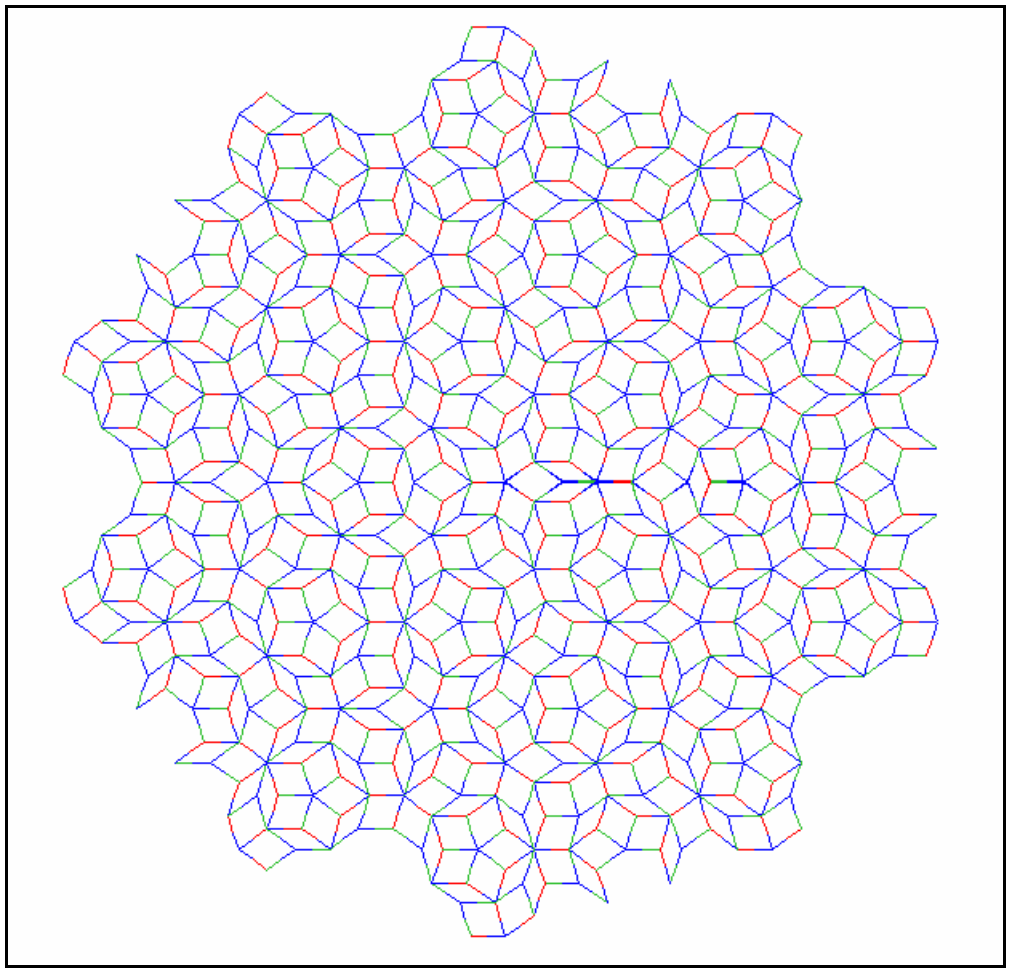
Figura 66 – Ladrilho baseado em hexágonos (7 iterações).



Direções	**6**
Axioma	**F**
Regra	p_1 : **F→-2F+3F+[+4F+5F]-**

A definição Sistema-L [NIE99] da Figura 67 é, sem dúvida, uma das mais difíceis apresentadas até o momento. Nesta definição foi utilizado o símbolo ∅ que significa um símbolo que não tem influência na interpretação. Pode ser um espaço em branco ou um caracter que não tenha sido utilizado ainda pela definição.

Figura 67 – Penrose em 4 iterações.



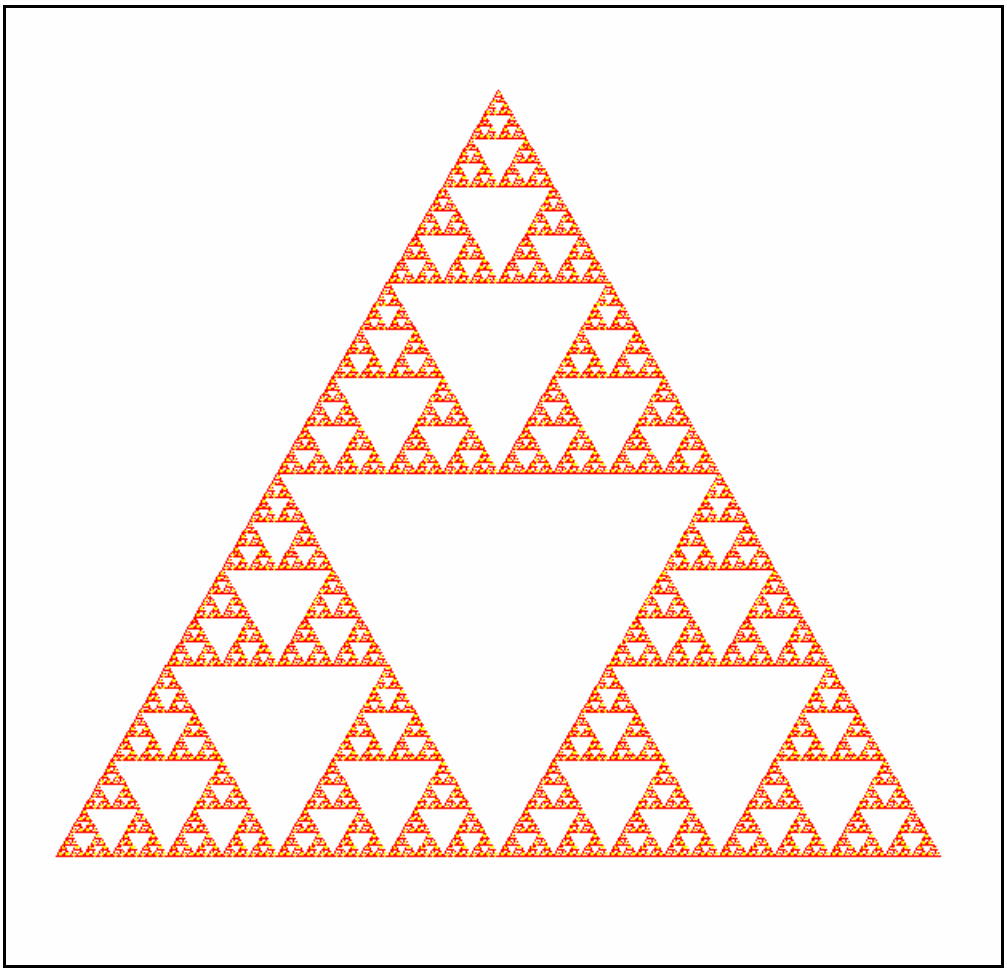
Direções	**10**
Axioma	**[Y]++[Y]++[Y]++[Y]++[Y]**
Regras	p_1 : **W→Y9F3F++Z7F9F----X3F9F[-Y9F3F----W9F7F]++** p_2 : **X→+Y9F3F--Z7F9F[---W9F7F--X3F9F]+** p_3 : **Y→-W9F7F++X3F9F[+++Y9F3F++Z7F9F]-** p_4 : **Z→--Y9F3F++++W9F7F[+Z7F9F++++X3F9F]--X3F9F** p_5 : **F→∅** p_6 : **C→∅**

Como já foi exposto antes, o mapeamento das regras que regem uma dinâmica é objeto de constante pesquisa dos matemáticos e biólogos. Uma definição Sistema-L não é facilmente deduzida, mesmo para quem é dedicado totalmente a esse estudo.

5.1.4 CURVAS FRACTAIS

Sistemas-L demonstraram eficiência na geração algumas curvas fractais descobertas há séculos atrás. O Triângulo de Sierpinski, representado na Figura 68, é um exemplo.

Figura 68 – Triângulo de Sierpinski em 8 iterações.

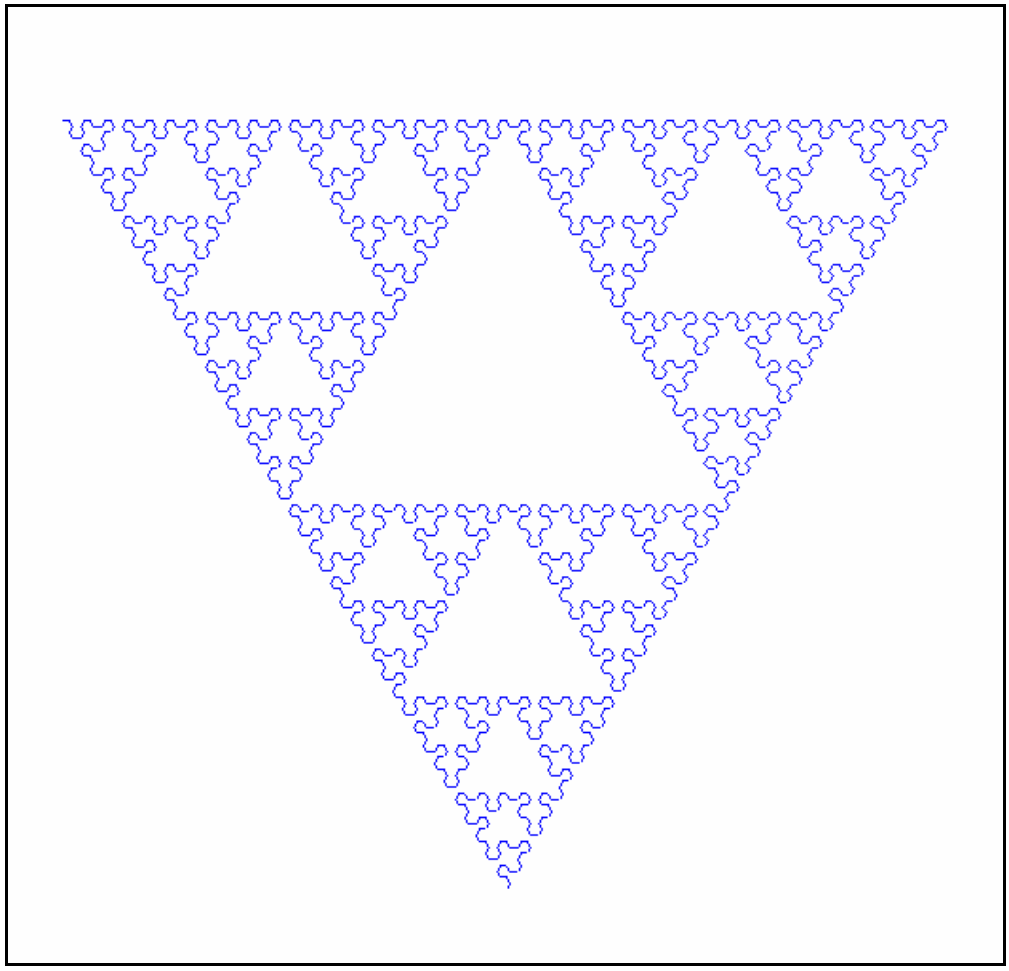


Direções	**6**
Axioma	**FXF++FF++FF**
Regras	p_1 : **F→7FF** p_2 : **X→++8FXF--FXF--FXF++**

Se formos pesquisar detalhes das curvas fractais, iremos nos deparar com diversos formalismos matemáticos utilizados para representá-las. No entanto, aqui é necessário apenas um conjunto de caracteres formais, os quais não foram descobertos ao acaso.

A Figura 69 é uma extensão do Triângulo de Sierpinski. Sua definição [NEI99] Sistema-L segue a figura.

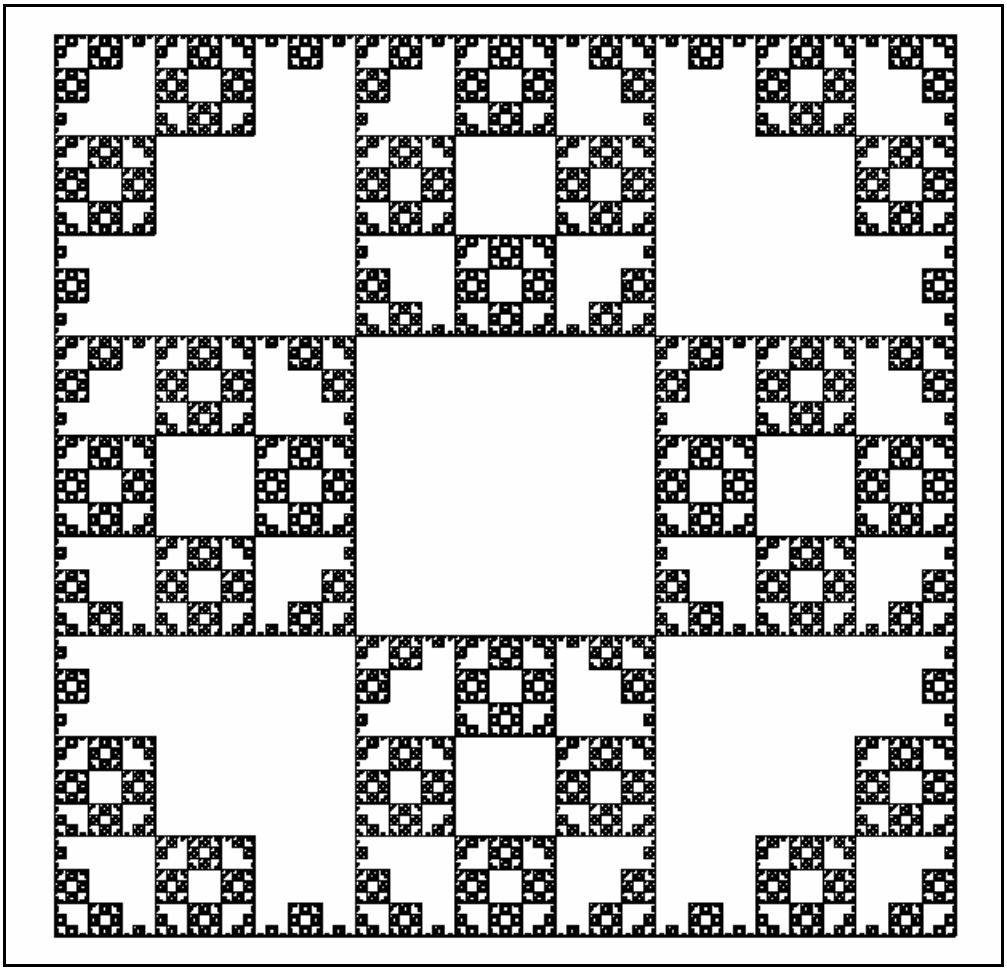
Figura 69 – Triângulo de Sierpinski modificado em 7 iterações.



Direções	**6**
Axioma	**9YF**
Regras	p_1 : **X→YF+XF+Y** p_2 : **Y→XF-YF-X**

A Figura 70 é uma construção realizada por Sierpinski chamada de Tapete de Sierpinski. Sua definição [PEI88] Sistema-L, bem simples por sinal, segue a figura.

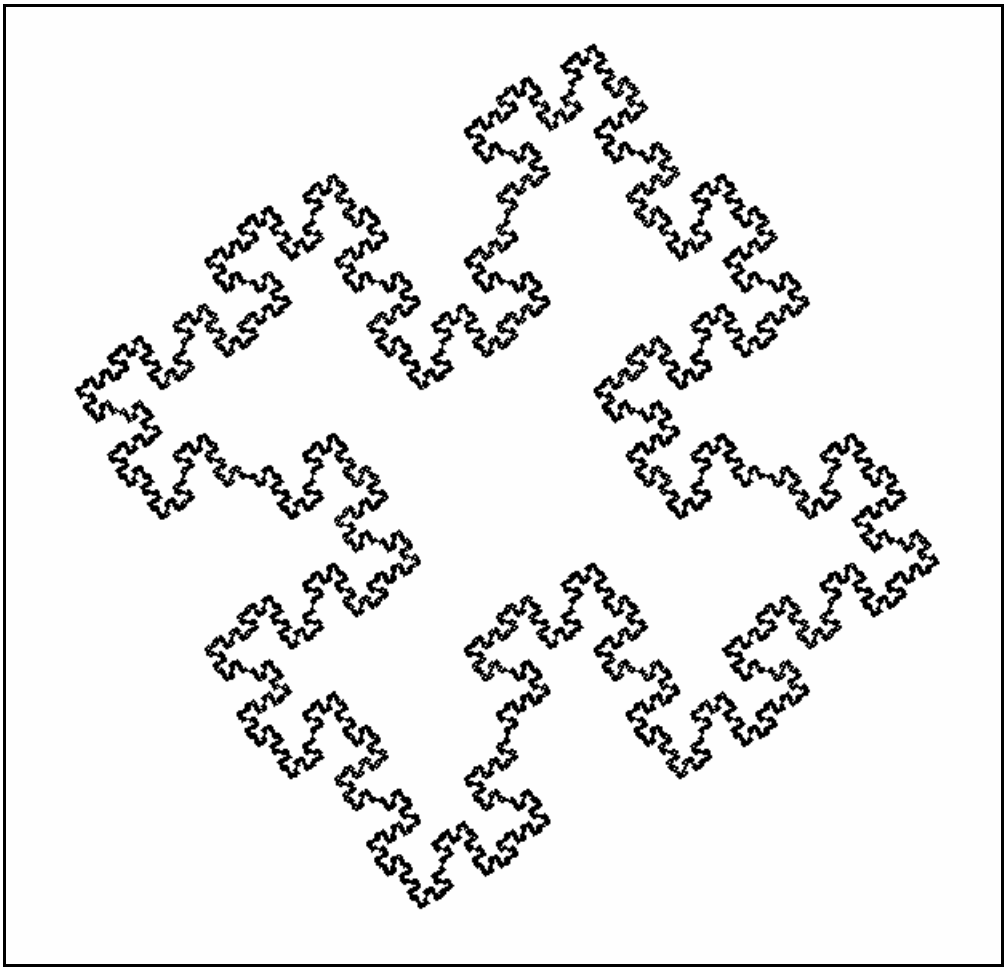
Figura 70 – Tapete de Sierpinski em 6 iterações.



Direções	**4**
Axioma	**F+F+F+F**
Regra	p_1 : **F→FF+F+F+F+FF**

Também baseada nas construções von Koch, a Figura 71 mostra a Curva Quadrática de von Koch. Sua definição [PEI88] Sistema-L segue a figura.

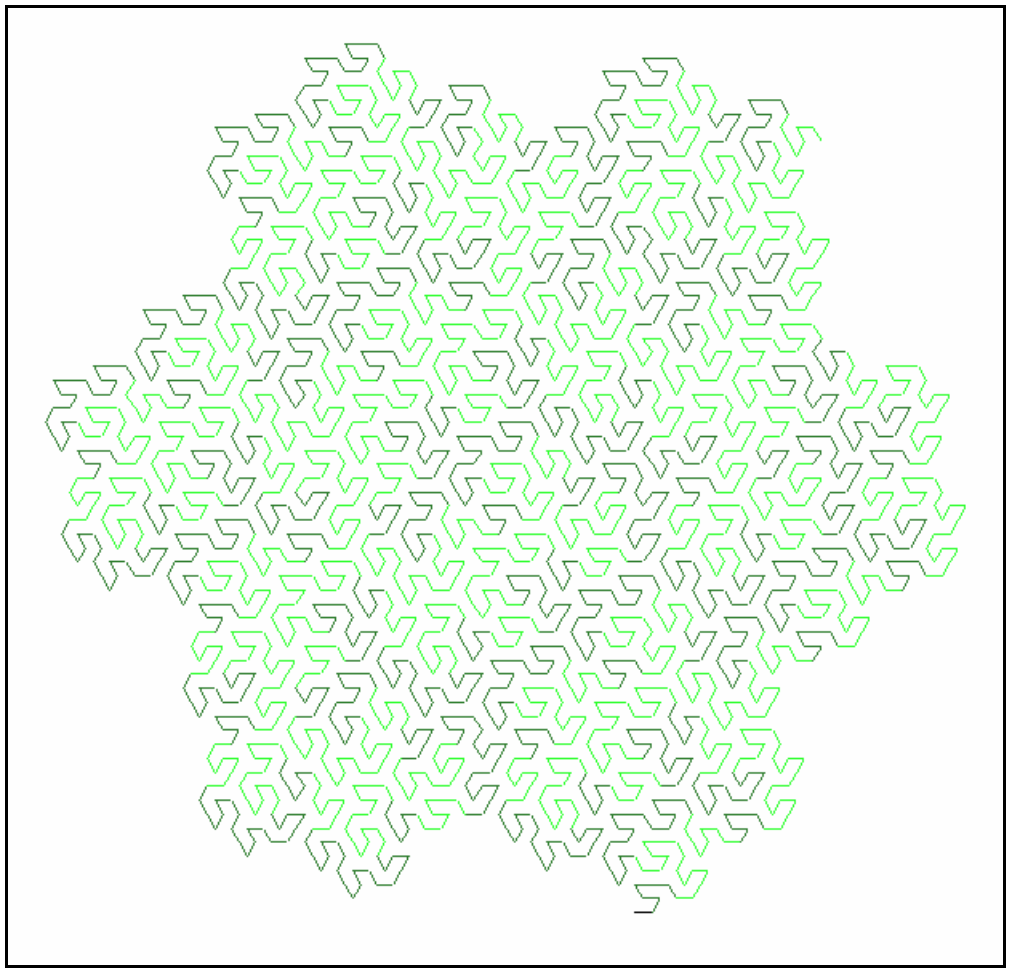
Figura 71 – Curva Quadrática de von Koch em 5 iterações.



Direções	**4**
Axioma	**F-F-F-F**
Regra	p_1 : **F→F-F+F+FF-F-F+F**

A imagem da Figura 72, chamada de Curva de Peano, foi gerada em 4 iterações de reescrita. Sua definição [NEI99] Sistema-L segue a figura. Novamente, cada regra foi colorida com uma cor diferente, possibilitando visualizar onde cada uma aparece na imagem.

Figura 72 – Curva de Peano em 4 iterações.

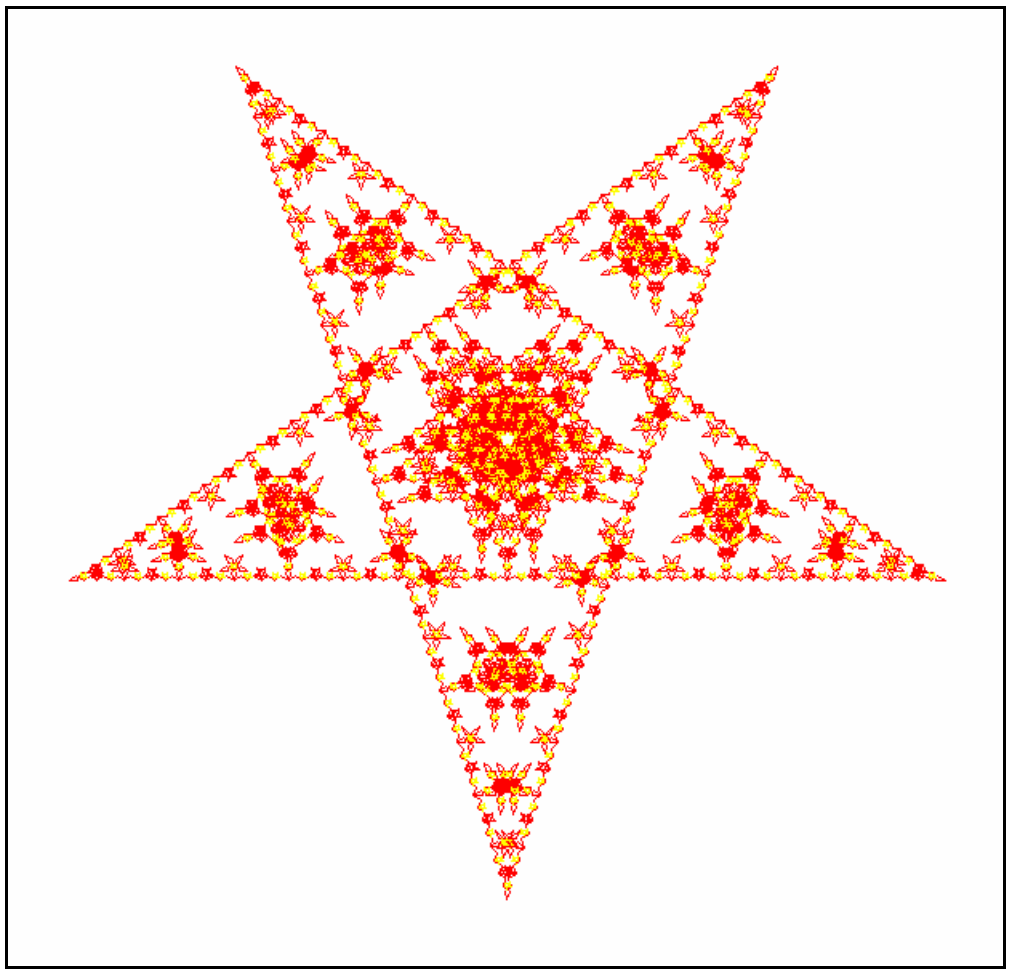


Direções	**6**
Axioma	**FX**
Regras	p_1 : **X→0X+YF++YF-FX—FXFX-YF+** p_2 : **Y→-5FX+YFYF++YF+FX--FX-Y**

5.1.5 OUTRAS IMAGENS

Recentes usos dos Sistemas-L estão na modelagem de estruturas compostas pela natureza, especialmente as plantas e desenvolvimento de células. Uma vez que, qualquer dinâmica que possa ter seu comportamento mapeado para um conjunto de caracteres formais, qualquer tipo de imagem pode ser gerada. As figuras a seguir são exemplos disso e demonstram que a principal característica dos Sistemas-L é poder representar complexos objetos partindo de um pequeno conjunto de informação. A Figura 73 é um exemplo.

Figura 73 – Estrela fractal.

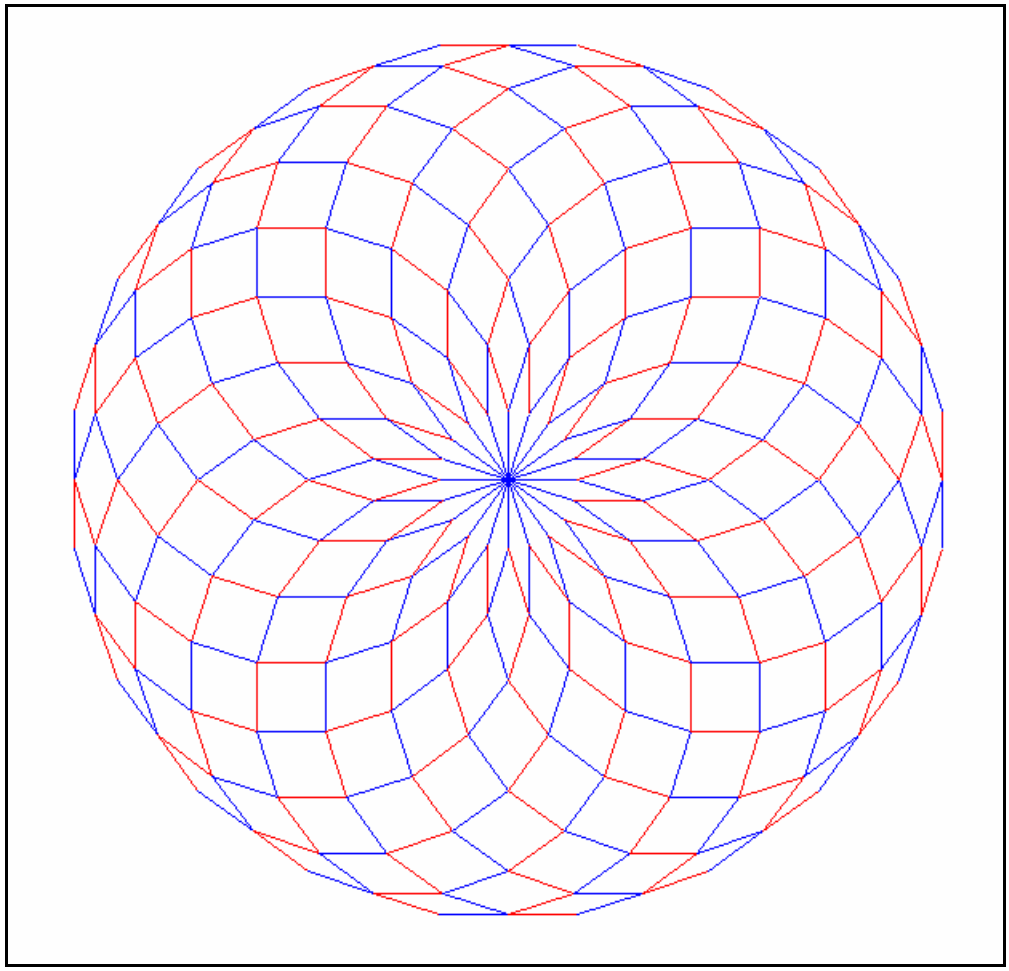


Direções	**5**
Axioma	**X**
Regras	p_1 : **X→8F++F++F++F++F** p_2 : **F→7F--X—F**

Através de 8 iterações e o pequeno conjunto de informações contido na definição Sistema-L [NIE99], a Figura 73 parece que teve origem de um complicado algoritmo matemático. É uma verdadeira obra de arte.

A Figura 74 também parece ter origem de um complicado processo. Mesmo que em sua definição Sistema-L apresente apenas uma regra aparentemente difícil, não é possível prever que em 20 iterações possa gerar tal imagem. O momento em que está sendo desenhada as imagens fractais é mais interessante do que vê-las desenhadas.

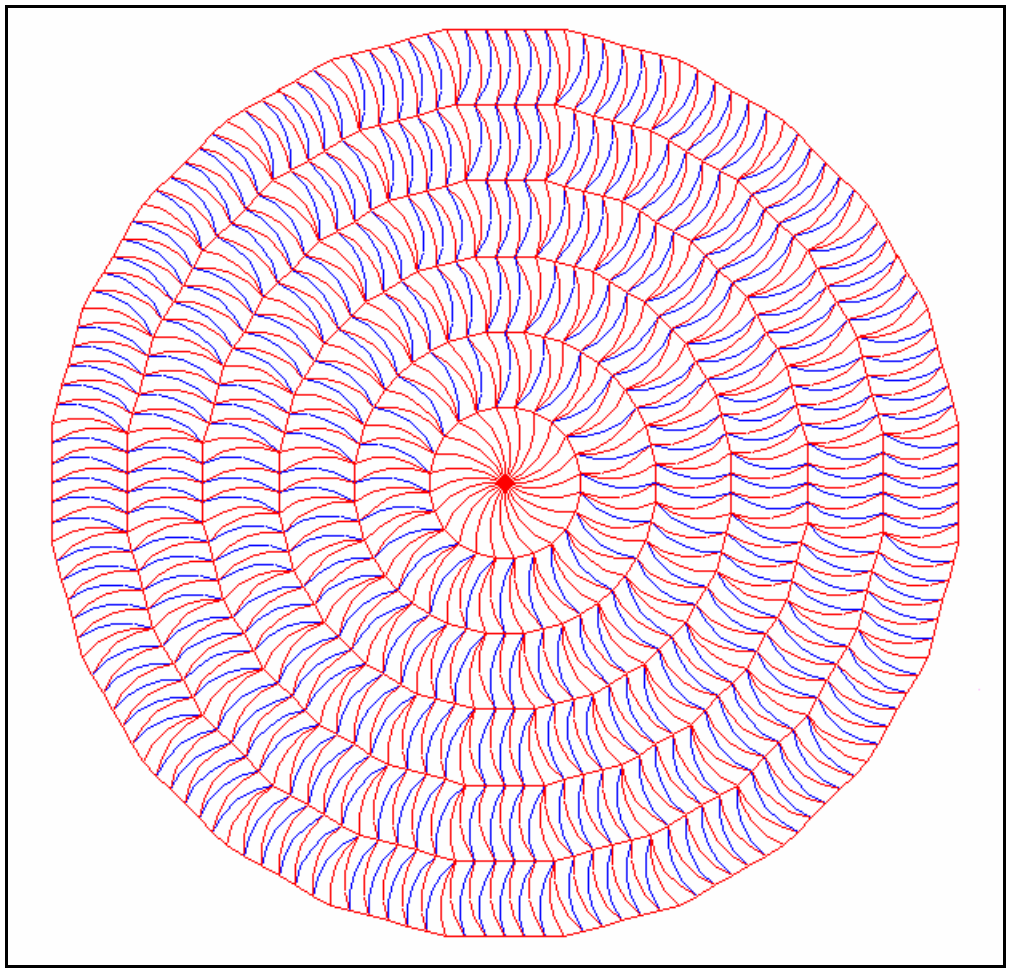
Figura 74 – Espiral 1.



Direções	**20**
Axioma	**A**
Regra	p_1 : **A→[-A]9F+7F+9F+7F+9F+7F+9F+7F+9F+7F+9F+7F+9F+7F+9F+7F+9F+7F+9F**

A Figura 75 foi gerada em 7 iterações. Sua definição [NIE99] Sistema-L segue a figura. A forma concêntrica faz pensar que foi usado círculos, mas tudo foi feito apenas com seguimentos de reta.

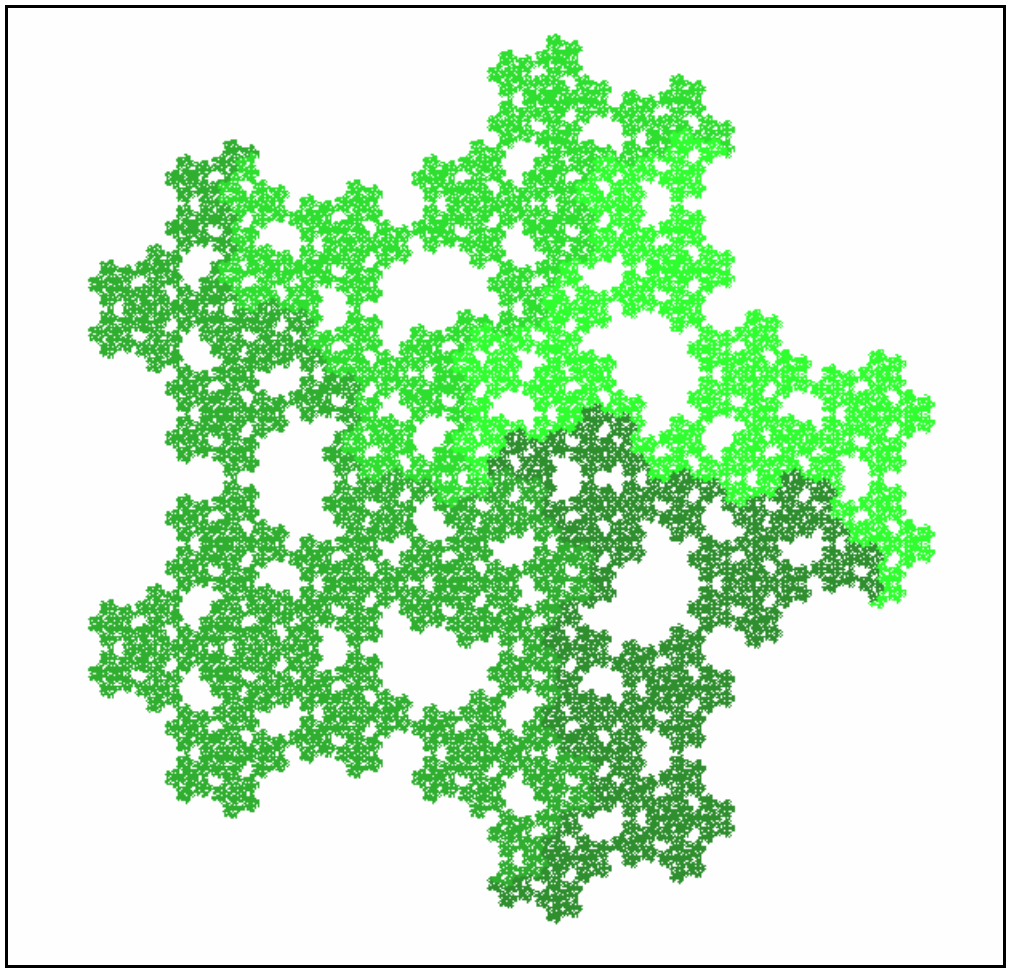
Figura 75 – Espiral 2.



Direções	**24**
Axioma	**AAAA**
Regras	p_1 : **A-->X+X+X+X+X+X+X+** p_2 : **X-->7[F+F+F+F[---X-Y]+++++F++++++++F-F-F-F]** p_3 : **Y-->9[F+F+F+F[---Y]+++++7F++++++++F-F-F-F]**

A imagem da Figura 76, em 6 iterações, foi descoberta acidentalmente na tentativa de criar uma curva de preenchimento. Seu criador nomeou-a de Pentigree. Sua definição [NEI99] Sistema-L a segue.

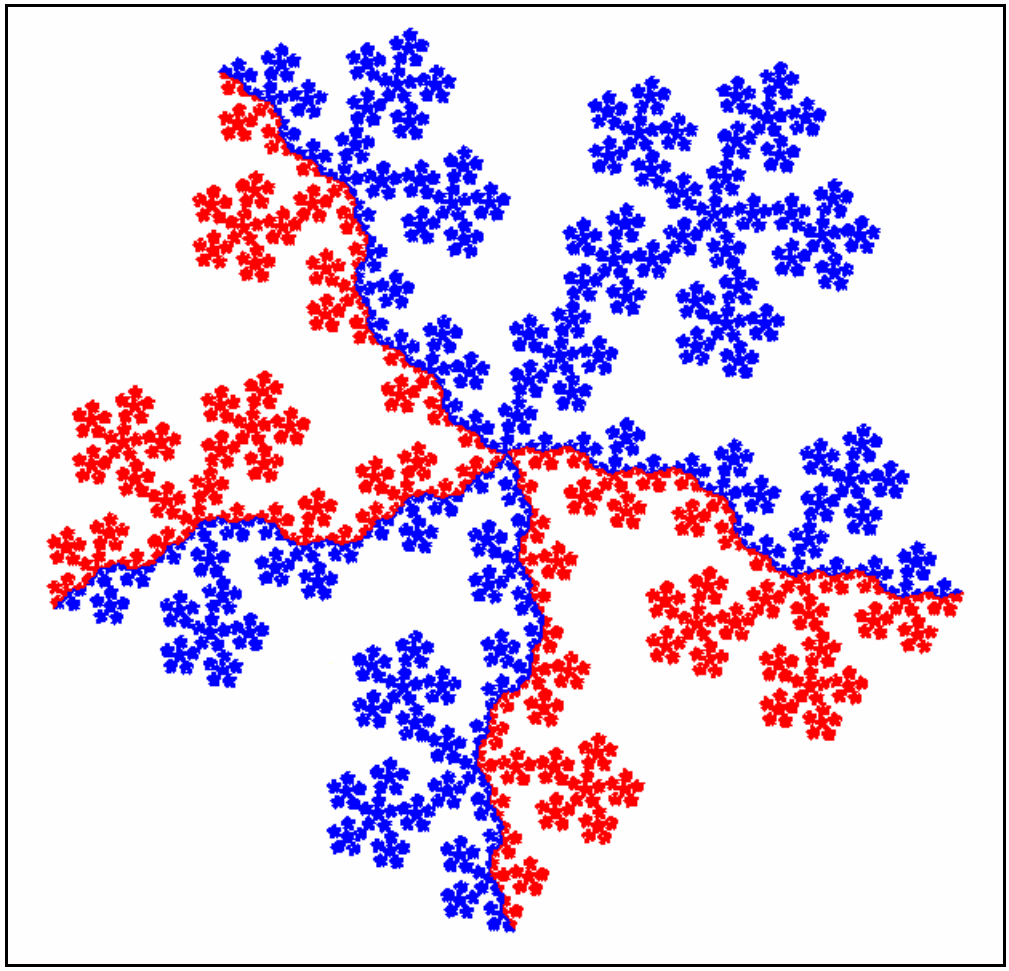
Figura 76 – Pentigree.



Direções	**5**
Axioma	**1F-2F-2F-4F-5F**
Regra	p_1 : **F→F-F++F+F-F-F**

Baseada em pentágonos, a imagem da Figura 77 foi gerada em 6 iterações de reescrita com a definição [NEI99] Sistema-L que segue a figura.

Figura 77 – Estrutura baseada em pentágonos.



Direções	**5**
Axioma	**9F-7F-9F-7F-9F**
Regra	p_1 : **F→F-F-F++F+F-F**

5.2 COMENTÁRIOS

Para a validação da funcionalidade do protótipo foram utilizadas definições Sistemas-L que já foram criadas durante os estudos por matemáticos ou biólogos, encontradas em livros e na Internet. A maioria das imagens têm raiz no século passado e geralmente possui o nome de seu criador.

O estudo das propriedades matemáticas ou topológicas das imagens não são do escopo deste trabalho. A intenção é utilizar reescrita para representá-las através de símbolos formais ao invés de complicados algoritmos e fórmulas matemáticas.

5.3 PONTOS POSITIVOS

De acordo com a análise das imagens e validação dos algoritmos implementados, observaram-se os seguintes pontos positivos:

a) Analisando a implementação como um todo, é uma ferramenta de fácil uso e de interface amigável. Pode ser utilizada em estudos de reescrita paralela determinística por qualquer tipo de usuário com propósitos que vão desde científicos até a diversão, pois as imagens geradas possuem formas atraentes;

b) A utilização de alocação dinâmica para o processo de reescrita fez com a que a performance fosse melhorada. Em uma tentativa de utilizar comandos para manipulação de cadeias de caracteres disponíveis pela linguagem de programação, percebeu-se a queda de performance à medida que o axioma se expandia, impedindo que determinadas definições chegassem a um nível de iteração razoável;

c) A possibilidade gravar, imprimir as imagens ou copiá-las para a área de transferência faz com que sejam aproveitados os resultados em outros ambientes gráficos;

d) A gravação do axioma expandido permite uma análise do processo de reescrita nos primeiros níveis com o objetivo de depurar o axioma que está sendo expandido. Com os primeiros níveis de iteração gravado em um arquivo texto, é possível interpretar manualmente o axioma;

e) A utilização de cores torna mais fácil o entendimento de onde as regras estão sendo utilizadas;

f) Não a preocupação de ajuste da imagem na tela. Sempre a imagem será desenhada de forma a preencher o melhor espaço possível e centralizada. Ações que maximizam e minimizam a tela faz com que a imagem seja redesenhada novamente;

g) A utilização da opção que permite que a imagem seja em preto e branco deixa o processo mais rápido. Com isso o espaço em memória que seria ocupado pelos caracteres de cores, pode ser ocupado por caracteres relevantes da descrição.

5.4 LIMITAÇÕES

Durante a utilização do protótipo na validação dos algoritmos implementados notou-se as seguintes limitações:

a) De um modo geral, a maior limitação do protótipo durante o processo de reescrita é o total de memória física disponível no computador. Definições muito complexas necessitam que seja feito *Swap*, decaindo a performance e consumindo todos os recursos do sistema;

b) O processo de reescrita que utiliza Sistema-L Estocástico é limitado apenas a um tipo de caracter no axioma, que deve ser uma letra **F** (efe) maiúscula. Também não é utilizado valores randômicos no incremento angular e variações do comprimento de cada reta. Portanto, mesmo que sejam encontradas definições Sistemas-L em livros e, cujas definições consistem de caracteres que não são reconhecidos pelo processo de interpretação aqui implementado, não haverá resultados inesperados. O protótipo não trata eventuais enganos na informação das definições;

c) Uma limitação da técnica de reescrita para geração de imagens seria a falta clareza das imagens em um nível de iteração muito alto ou no processo de reescrita de definições muito complexas. Definições complexas outras abordagens de reescrita que ainda estão sendo estudas pelos pesquisadores.

5.5 RESUMO

A utilização da reescrita paralela, proposta por Aristid Lindenmayer, no protótipo permite a criação de imagens como estruturas naturais determinísticas e estocásticas, formas matemáticas abstratas, curvas de preenchimento, curvas fractais e ladrilhos (*tiling*). Permite também que seja utilizado para desenhar formas geométricas simples como triângulos, quadrados, hexágonos, pentágonos, etc.

Basicamente, o algoritmo implementado para a realização do processo determinístico nos Sistema-D0L é eficaz e validou todos os modelos, com isso, o protótipo pode ser utilizado por pessoas que desejam estudar a reescrita paralela.

Já o algoritmo implementado para a realização do processo estocástico está limitado a um conjunto muito pequeno de símbolos no axioma, além de não permitir a variação do incremento angular e os segmentos de reta que formam a imagem.

A utilização de alocação dinâmica no processo de reescrita fez com que se ganhasse na performance, mas está limitado aos recursos disponíveis da máquina, como memória física, por exemplo.

6 CONCLUSÕES E EXTENSÕES

Neste capítulo serão apresentados as conclusões do estudo e implementação. Em seguida serão mencionadas as extensões para novos trabalhos que incluem novos tipos Sistemas-L e melhoramentos do protótipo implementado.

6.6 CONCLUSÕES

Considerando o processo utilizado para a geração imagens fractais e o estudo feito, os Sistemas-L são um método aberto à várias interpretações. A utilização dos Sistemas-L nas ciências da computação já é antiga, mas o seu uso na computação gráfica, especificamente, é recente. A interdisciplinaridade no estudo dos fractais é bastante abrangente.

O protótipo, chamado de L-Draw, implementado apresentou bons resultados nos algoritmos utilizados, gerando imagens que demonstram formação topológica das plantas, construção de formas matemáticas abstratas e curvas fractais, bem como outras imagens. Houve um ganho de performance no processo de substituição das regras devido ao uso de alocação dinâmica. Mesmo estando limitado ao total de memória física disponível no sistema, foi possível a geração de imagens suficientemente complexas sem necessitar muito dos recursos disponíveis da máquina.

Mesmo que tenham sido utilizadas definições já existentes, algumas definições foram levemente alteradas para a criação de novos padrões, exibindo comportamento totalmente imprevisível. Percebeu-se que, com o aumento do nível de iteração, a imagem pode apresentar-se sem nenhuma alteração ou perder totalmente a nitidez. Imagens bastante complexas podem ser criadas com pequeno conjunto de símbolos formais, levando a crer que tenha sido utilizado outro processo.

As funções implementadas no protótipo deram maior flexibilidade tanto dos resultados quanto no manuseio, interação e entendimento do processo. É uma ferramenta que serve para leigos e matemáticos no estudo da reescrita paralela.

Tudo que surge das teorias dos fractais ainda encontra-se em estado da arte. Há muita teoria e pouca prática. Isso pode ser percebido neste trabalho que deixou várias portas abertas para outras pesquisas, as quais são indicadas no item a seguir.

6.7 EXTENSÕES PARA NOVOS TRABALHOS

a) Implementação de novos ambientes baseados nos diversos tipos de Sistemas-L apresentados resumidamente no capítulo 3;

b) Implementação de um ambiente que gere as imagens em espaço com 3 dimensões, utilizando projeções ortogonais (vista de frente, de cima e de lado) em 2 dimensões. Para que fosse possível a visualização da imagem em ambientes que trabalham com 3 dimensões e utilização de técnicas de render, a imagem resultante deveria ser exportada para um formato que pudesse se lido por tal ambiente. Por exemplo, a exportação da imagem para o formato DXF (*Drawing Exchange Format*) permite que seja lido pelo AutoCAD;

c) Baseado no capítulo 2, seria possível a implementação de um ambiente que gerasse comportamentos caóticos de funções matemáticas. Neste caso envolve a teoria de mapeamento em plano complexo onde as figuras 1 e 2 são exemplos;

d) Uma extensão para o protótipo implementado seria melhorar a interpretação gráfica, fazendo com que os caracteres representados na Tabela 4 fossem interpretados pelo algoritmo. Variação no processo estocástico do incremento angular e comprimento da reta. Para melhor acompanhar o processo de reescrita e interpretação deveria ser criado uma maneira de acompanhar que está acontecendo em passo a passo;

e) Em uma aplicação prática, deveria ser feito um estudo de uma família de planta. Através disto poderia ser construído um ambiente para o estudo desta planta;

f) Implementação de um ambiente em 3D com utilização de componentes como flores e folhas e um tipo de Sistema-L apropriado;

g) Em caso de uma implementação para imagens muito complexas, que seja necessário muita memória, a saída seria implementar um algoritmo que fizesse uma seguimentação entre processo de reescrita e o processo de interpretação. Assim, o axioma seria interpretado e a memória liberada à medida que os recursos da máquina fossem se esgotando.

REFERÊNCIAS BIBLIOGRÁFICAS

[BEN93] BENVENUTTI, Vitor César. **Um ambiente para geração de imagens fractais baseadas em sistemas-l.** Blumenau, 1993. TCC (Bacharelado em Ciências da Computação) Centro de Ciências Exatas e Naturais, FURB.

[BIR94] BIRCHAL, Marco Aurélio de Souza. **Caos, fractais e o método computacional.** Revista Micro Sistemas. ano XIII, nº 142, Setembro, 1994.

[BOU91] BOURKE, Paul. ***An introduction to fractals*.** Endereço eletrônico : http://www.swin.edu.au/astronomy/pbourke/fractals/fracintro, 1991.

[BOW88] BOWSER, David D. ***Fractal computer graphics*.** Minessota : Mankato State University. Endereço eletrônico : http://www.vii.com/~bowserd/fractals/toc.html, 1988.

[CAN96] CANTÚ, Marco. **Dominando o delphi**. São Paulo : Makron Books, 1996.

[CRI91] CRILLY, A. J. e outros. ***Fractals and chaos*.** New York : Springer-Verlag, 1991.

[CUO97] CUONG, Ngugen . ***Ray traced evolution – user's manual*.** Endereço eletrônico : http://www.rz.tu-ilmenau.de/~juhu/GX/RTEvol/DOC/LATEX2HTML/ manual.html, 1997.

[DEV90] DEVANEY, Robert L. ***Chaos, fractals, and dynamics*.** New York : Addison-Wesley, 1990.

[FUR98] FURLAN, José David. **Modelagem de objetos através da UML.** Makron Books. São Paulo, 1998.

[HAM99] HAMPISHRE, Paulo Pereira. **Utilizando java como ferramenta corporativa.** Revista Developers', v. 3, nº 34, Junho, 1999.

[HIL90] HILL, Francis S. Jr. ***Computer graphics.*** New York : Macmillian Publishing Company, 1990.

[LAN99] LANIUS, Cyntia. ***Why study fractals.*** Endereço eletrônico : http://math.rice.edu/~lanius/fractals/iter.html, 1999.

[LIN96] LIN, Tong. ***Animation of l-system based 3-D plant growing in java.*** Electrical Engineering and Computer Science Departament – University of Maryland Baltimore Country. Baltimore. Endereço Eletrônico : http://www.cs.umbc.edu/~ebert/693/TLin/top.html, 1996.

[NIE99] NIELSEN, Birger. ***Lindenmayer sistemer.*** Endereço eletrônico : http://hjem.get2net.dk/bnielsen/lsystems.html, 1999.

[OCH98] OCHOA, Gabriela. ***An introduction to lindenmayer systems.*** School of Cognitive and Computing Sciences – The University of Sussex. Endereço eletrônico : http://www.cogs.susx.ac.uk/users/gabro/lsys/lsys.html#OVR, 1998.

[PEI86] PEITGEN, Heinz-Otto e outros. ***The beauty of fractals.*** New York : Springer-Verlag, 1986.

[PEI88] PEITGEN, Heinz-Otto e outros. ***The science of fractals images.*** New York : Springer-Verlag, 1988.

[PRU90] PRUSINKIEWICZ, Przemyslaw. ***The algorithmic beauty of plants.*** New York : Springer-Verlag, 1990.

[ROZ92] ROZEMBERG, Grzegorz e outros. ***Lindenmayer Systems – Impacts on theoretical computer science, computer graphics and developmental biology.*** New York : Springer-Verlag. 1992.

[WOO95] WOODS, Alan. ***Chaos theory.*** The Division of Labour Chaos and Dialectics. Endereço eletrônico : http://easyweb.easynet.co.uk/~zac/chapt17.html, 1995.

[WRI96] WRIGHT, David J. ***Dynamical systems and fractals lecture notes*.** Endereço eletrônico : http://www.math.okstate.edu/mathdept/dynamics/lecnotes/lecnotes.html, 1996.