UNIVERSIDADE FEDERAL FLUMINENSE
INSTITUTO DE CIÊNCIA DA COMPUTAÇÃO
CIÊNCIA DA COMPUTAÇÃO

CYNTHIA LOURDES OLIVEIRA PEREIRA

MODELAGEM DE PLANTAS UTLIZANDO L-SYSTEM

Geometria Fractal simplificando formas complexas

NITERÓI
2004

UNIVERSIDADE FEDERAL FLUMINENSE

Modelagem de plantas utlizando L-System Geometria Fractal simplificando formas complexas

Cynthia Lourdes Oliveira Pereira

Monografia apresentada ao Departamento de Ciência da Computação da Universidade Federal Fluminense como parte dos requisitos para obtenção do Grau de Bacharel em Ciência da Computação.

Banca Examinadora:

Aura Conci (orientadora)

Inhaúma Neves Ferraz

Isabel Cafezeiro

Departamento de Ciência da Computação

Niterói, 19 de Fevereiro de 2004

À minha mãe, pelo incentivo para vencer desafios e obstáculos, pela credibilidade no meu potencial e pela compreensão de muitas ausências em prol da minha dedicação aos estudos.

AGRADECIMENTOS

À Aura Conci – minha orientadora, pela credibilidade e confiança.

A Silvan Luiz da Costa Vale – amigo da Universidade, pelas idéias e ajudas técnicas.

SUMÁRIO

RESUMO

Algumas idéias surgiram ao longo da graduação para o Projeto de Final de Curso. A imensa simplicidade da Geometria fractal em gerar imagens complexas e belas chamou a atenção e despertou a possibilidade de um estudo mais completo e detalhado sobre o assunto. Pesquisas sobre este tema revelam a deficiência de sistemas voltados para usuários com pouco ou nenhum conhecimento no assunto. Este trabalho tenta introduzir de forma incremental os conceitos da Geometria Fractal e o modelo L-System, de forma que o leitor possa com o sistema desenvolvido colocar em prática exemplos e resultados obtidos em sua leitura. Para que o sistema desenvolvido fosse capaz de atender a dois diferentes grupos de usuários: Usuários avançados (ligados à computação gráfica, especificamente a abordagem de L-Systems) e usuários iniciantes (conhecimento de computação em geral) uma pesquisa sobre interface homem-máquina foi realizada de forma atendê-lo sem prejudicar a funcionalidade e objetivos iniciais do sistema.

ABSTRACT

In the course of graduation some ideas had appeared about course end work. The large facility of Fractal geometry to generate complex and beautiful images have given a chance for a complete and specific study for this topic. Researchs about this topic have indicated the deficiency of systems for users have little or no knowledge for fractal geometry. This work try introducing of incremental form the main idea of Fractal geometry and L-System model, the way that the reader can pratice with system developed the examples end results gotten with your reading. So that the developed system was able to attend for two differents user groups: Advanced users (knowledge in the graphic computer area, specifically in the L-System models) and beginning users (knowledge in science computer in general) a Human/Computer Interface was research for attend two groups without damage system inicial purpose and activity.

1 GEOMETRIA FRACTAL

1.1 INTRODUÇÃO

A palavra fractal foi criada por Benoit Mandelbrot para descrever um objeto geométrico que nunca perde a sua estrutura qualquer que seja a distância de visualização. Derivada do adjetivo fractus, do verbo frangere, que significa quebrar. Mandelbrot classificou desta forma os seus objetos de estudo pois possuíam uma dimensão fracionária.

As dimensões fracionárias tornaram-se uma forma de quantificar dimensões até o momento imprecisas pelo seu alto grau de irregularidade ou tortuosidade. Uma costa sinuosa, por exemplo, impossibilita a sua medição em termos de comprimento, mas possui um grau determinado de irregularidade.

A palavra fractal acima de tudo significa auto-semelhante. A auto-semelhança é a simetria através da escala, ou seja, um objeto possui auto-semelhança se apresenta sempre o mesmo aspecto a qualquer escala em que seja observado. Se repararmos, todas as formas geométricas conhecidas, perdem a sua estrutura quando são ampliadas ou diminuídas. Um círculo numa escala muito maior não é nada mais do que uma reta. Basta lembrar que ha apenas 600 anos se pensava que a Terra era plana.

Isto acontece porque na escala humana não vemos mais do que uma linha reta no horizonte. No entanto a maior parte dos objetos em que lidamos no dia-a-dia não são retas, esferas, nem cones. Olhando por exemplo para um tronco de árvore, verificamos que é extremamente rugoso e irregular. Se olharmos um pequeno pedaço desse tronco ao microscópio observaremos novas rugosidade

e irregularidades que antes não tinham sido notadas. No entanto, esta imagem assemelha-se bastante a anterior e suas irregularidades caracterizam um fractal.

As imagens fractais geradas por computador são o resultado de iterações executadas num sistema não linear, de forma recursiva que possibilita a quem observa imagens de grande beleza.

O conceito de sucessão e especialmente de limite, estão na base da Análise Infinitesimal, talvez o mais importante ramo da Matemática, pelas suas prodigiosas aplicações às ciências experimentais. Podemos dizer que o progresso tecnológico do século XIX e do século XX foi em grande parte pelo desenvolvimento vertiginoso da Análise Infinitesimal nas duas vertentes: Cálculo Diferencial e Cálculo Integral.

Imagens construídas utilizando-se técnicas fractais são definidas por regras muito simples. Conseguimos obter apenas meia dúzia de termos, mesmo recorrendo aos melhores instrumentos de desenho. Mas, com um computador, o processo pode continuar indefinidamente, obtendo-se figuras com pormenores invisíveis a olho nu. Ainda podemos contar com a enorme capacidade de ampliação dos modernos computadores que torna possível visualizar os termos avançados destas sucessões, fornecendo imagens incrivelmente belas.

A forma fractal de Mandelbrot, pode ser obtida por uma sucessão do tipo x, x^2+x, $(x^2+x)^2 + (x^2+x)^3$ .... Qualquer que seja o nível de ampliação, a figura desenhada apresenta infinitas formas menores idênticas à primitiva.

Na natureza encontramos muitos exemplos de sistemas que apresentam um caráter fractal: nos vegetais temos a couve-flor e o brócolis, no corpo humano temos os pulmões e os brônquios e em elementos geográficos temos as redes hidrográficas, costa de um país, nuvens e rochas.

A Geometria Fractal é a geometria da irregularidade da natureza. É um novo ramo da Matemática, uma nova forma de encarar a Ciência e dessa forma tentando explicar certos fenômenos para os quais a geometria euclidiana e a Física de Newton se revelam insuficientes.

Uma imagem obtida por técnicas fractais pode ter uma aparência estranha: um vírus visto ao microscópio, paisagens de outro planeta, delírio de um pintor abstracionista, porém, é estranhamente bela.

As aplicações das noções de fractal e de “caos” revelam-se de grande valia em Meteorologia, Hidráulica, Física, Geologia, Geografia e até em História, Economia e Lingüística. Os lingüistas aplicam a teoria dos fractais no estudo da

evolução dos dialetos. Em Medicina encontramos características fractais em fenômenos cardíacos e pulmonares.

Figuras que no início do século eram vistas como meras anomalias matemáticas têm hoje um papel notável na interpretação da realidade.

1.2 CARACTERÍSTICAS DE UM FRACTAL

Uma delas, pelo seu próprio nome é a sua dimensão não inteira. Um fractal pode ter dimensão superior a 1, dimensão da reta, mas inferior a 2, dimensão do plano. A dimensão fractal de um objeto mede o seu grau de irregularidade, sua estrutura e comportamento, se tratando de uma figura ou de um fenômeno físico, biológico ou social.

Um exemplo clássico de fractal é a costa de um país, pois seu comprimento é sempre um valor aproximado, em geral calculado a partir de fotografias de satélite. Caso estas fotografias fossem tiradas de um avião as irregularidades seriam mais visíveis e obteríamos um outro valor.

Se em vez de fotografias medíssemos diretamente todas as saliências e reentrâncias, obteríamos dessa vez um novo valor muito maior. Se em seguida tomássemos uma régua de 1 cm para executar essa tarefa, obteríamos maior precisão nas medidas dos contornos e o comprimento final obtido seria ainda maior.

Na verdade, o comprimento da costa de um país tende para o infinito, embora a área que limita seja finita. O mesmo acontece com o fractal Triadic de Von Koch ou floco de neve (*figura 1.1*), pois gera uma progressão geométrica de razão 4/3 (logo tende para o infinito) mas a área que limita é finita visto que cabe dentro de um círculo circunscrito ao triângulo inicial.

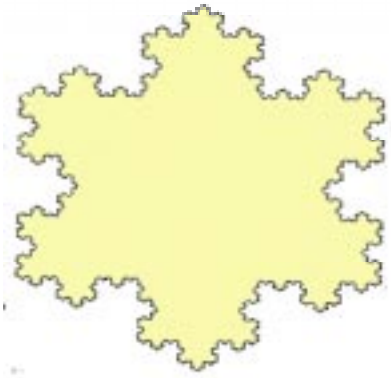

Figura 1.1 *Triadic de Von Koch ou floco de neve.*

Outra característica já comentada é a auto-semelhança. Repare que para cada pequeno “bico” do fractal de Von Koch é possível gerar uma sucessão de imagens semelhante, assim como uma pequena rocha é semelhante a uma montanha vista de longe, ou um pedaço de couve-flor é semelhante a couve inteira.

1.3 GEOMETRIA EUCLIDIANA X GEOMETRIA FRACTAL

Euclides (330-260 a.C.) é reconhecido como o matemático mais importante da Grécia clássica. Dele unicamente se sabe que ensinou e fundou uma escola em Alexandria, por volta do ano 300 a.C.. Seus trabalhos matemáticos chegaram até nós quase completos, pois já haviam sido traduzidos inicialmente para o árabe, e em seguida para o latim e outras línguas européias.

Segundo o historiador grego, Heródoto (século V a.C.), a geometria nasceu provavelmente no antigo Egito, pelas sucessivas medições de terra devido às inundações periódicas do rio Nilo, sendo rapidamente alargada à navegação. Mas é certo que muitas outras civilizações antigas possuíam conhecimentos de natureza geométrica, da Babilônia à China, passando pelas civilizações Hindus.

A palavra geometria é um vocábulo que deriva do grego *"geometrein"*, que significa medição da terra (*geo* = terra e *metrein* = medir).

No final do século XVII, iniciou-se um movimento crítico com os matemáticos John Wallis (1616-1703), Saccheri (1667-1733) continuando com Lambert (1728-1777) e Gauss (1777-1855). Os estudiosos perceberam algumas deficiências lógicas na obra "Elementos de Geometria" de Euclides, que passaram desapercebidas durante mais de dois milênios. Conseqüentemente, algumas propostas de geometrias alternativas surgiram.

Apesar destas críticas a geometria euclidiana e ao surgimento de inovações que foram responsáveis pela mudança de concepção dos matemáticos sobre a natureza da matemática, não se retira o valor à monumental obra de Euclides.

Durante muito tempo, o homem observou a natureza para entender conceitos de formas, figuras planas, volumes, retas e curvas. A Lua e o Sol foram representados como discos; o raio de luz deu-lhe a idéia de linha reta; o contorno de uma folha e o arco-íris, a idéia de curva; os troncos de algumas árvores e montanhas deram-lhe idéias de formas muito variadas.

Atualmente, um novo ramo da geometria surge, contemplando as irregularidades da natureza: a Geometria Fractal; também explicando certos fenômenos de turbulência em que a geometria euclidiana se revela insuficiente. Figuras que no início do século eram vistas como meras anomalias matemáticas, têm hoje um papel notável na interpretação da realidade.

Assim a Geometria fractal surge, batizada por Mandelbrot, rompendo o determinismo matemático e complementando a Geometria Euclidiana, possibilitando contemplar a natureza, trabalhando com as imperfeições que comumente são encontradas a nossa volta.

1.4 TEORIA DO CAOS

Muitos fenômenos não podiam ser previstos por leis matemáticas. Os fenômenos ditos "caóticos" são aqueles onde não há previsibilidade. Por exemplo: o gotejar de uma torneira; nunca se sabe a freqüência com que as gotas de água caem e não podemos determinar uma equação que possa descrevê-la. As variações climáticas e as oscilações da bolsa de valores também são caóticas.

Atualmente, com o desenvolvimento da Matemática e das outras Ciências, a Teoria do Caos surgiu com o objetivo de compreender e dar resposta às flutuações irregulares que se encontram na natureza. Hoje se sabe muito a respeito de fenômenos imprevisíveis, e já é possível ver os resultados. Por exemplo, em 1997, dois americanos conseguiram encontrar uma fórmula para prever aplicações financeiras e com isso ganharam o prêmio Nobel de Economia. O caos tem aplicações em todas as áreas.

Uma lei básica da Teoria do Caos afirma que a evolução de um sistema dinâmico depende crucialmente das suas condições iniciais. O comportamento do sistema dependerá então da sua situação de início. Se analisarmos o mesmo sistema, sob outras condições iniciais, logicamente ele assumirá outros caminhos e mostrar-se-á totalmente diferente do anterior.

1.4.1 EXEMPLOS DE CAOS NA VIDA COTIDIANA

O trânsito é um exemplo. Já se deve ter observado que há dias em que o congestionamento é maior. É bem provável que o transtorno tenha sido causado por um carro acidentado, ou uma empresa que dispensou seus funcionários mais cedo dando origem a um maior fluxo num cruzamento. Mesmo assim, o número de variáveis é grande e o comportamento do sistema depende muito das condições iniciais. Nunca se sabe quando o trânsito está bom ou mau.

Um exemplo tradicional é o "efeito borboleta", que diz essencialmente: "Uma borboleta bate asas na China e causa um furacão na América". Por mais absurdo que pareça, é a realidade; os fenômenos climáticos são de comportamento caótico e de difícil previsibilidade.

Muitos outros exemplos poderiam ser citados, mas não podemos esquecer que na natureza existem também fenômenos simples como a queda de um objeto, o som, o movimento dos astros, etc. Nem tudo é caótico. Quando falamos num sistema complexo não nos referimos somente à complexidade operacional, mas também à complexidade de elementos (as sutilezas do meio em que se passa e a pluralidade de variáveis).

1.5 HOLOFRACTAL, A ARTE NO FUTURO

A exposição Holofractal, de Eduardo Kac e Ormeo Botelho, foi inaugurada em Novembro de 1988 na galeria de Fotografia da Funarte. Esta foi a primeira obra de arte a unir a tecnologia holográfica, a computação gráfica, e a geometria fractal. A geometria fractal, que vem sendo desenvolvida pelo matemático polonês Mandelbrot, revoluciona os conceitos da geometria euclidiana e se dedica a estudar as formas irregulares da natureza. Assim, os fractais são unidades que ampliam infinitamente as possibilidades da percepção humana, acostumada até então a abordar a realidade apenas com as noções de reta, plano e espaço. Reconhecendo no universo as dimensões fracionárias, Mandelbrot trabalha com números aleatórios que pretendem dar conta das irregularidades existentes nos objetos e nos fenômenos naturais. E foi a partir de um *software* fractal (um programa capaz de gerar formas irregulares) que Kac e Ormeo se basearam para construir o holopoema "Quando?", trabalho pioneiro na aplicação da matemática fractal às artes visuais.

2 L-SYSTEM

2.1 INTRODUÇÃO

Sistemas Lindenmayer, mais conhecidos por L-Systems, são um formalismo matemático proposto pelo biólogo Aristid Lindenmayer em 1968 como um fundamento para a teoria axiomática do desenvolvimento ou crescimento biológico. L-Systems são um tipo particular de sistema simbólico dinâmico que dão uma interpretação geométrica para a evolução dos sistemas. Mais recentemente, L-Systems tem sido tema de muitas aplicações em computação gráfica. As duas principais áreas de estudos são geração de fractais e modelagem realística de plantas.

Um L-System é um autômato para modelar o desenvolvimento celular. Células são representadas por símbolos e a subdivisão é modelada pela troca desses símbolos por *strings* de símbolos. Para iniciar, considere um exemplo simples de L-System com dois tipos de células representadas pelas letras A e B. A célula A se subdivide em duas células representadas pela *string* AB. A célula B se subdivide em duas células representadas pela *string* BA. A ordem dos símbolos é relevante em L-Systems. O organismo abstrato modelado por este L-System cresce pela repetição de subdivisões das células. No nascimento, o organismo é uma única célula A. Depois de uma subdivisão o organismo tem duas células representadas pela *string* AB. Após duas subdivisões, cada célula se subdivide de acordo com as regras de subdivisão acima e o organismo segue com quatro células dadas pela *string* ABBA. Depois de três subdivisões o organismo é representado pela *string* ABBABAAB e depois de quatro subdivisões o organismo tem 16 células representadas pela *string* ABBABAABBAABABBA.

A linguagem utilizada para gerar L-Systems é axiomatizada em caracteres, promovendo uma conexão entre o problema lógico e a solução visual. Ela usa a reescrita de símbolos como técnica básica, podendo ser realizada utilizando-se recursividade, criando assim objetos complexos pela iteração de um conjunto de regras ou produções para substituir partes de uma geometria inicial simples. A geometria inicial é chamada de inicializador (ou axioma) e as regras que instruem as substituições são chamadas de geradores ou produções.

O trabalho de Chomsky em linguagens formais (1957) gerou um grande interesse em sistemas de reescrita. Subseqüentemente, passou-se por um período de fascinação por sintaxe, gramáticas e suas aplicações em ciência da computação, dando início ao campo de linguagens formais.

A diferença essencial entre as gramáticas de Chomsky e os L-Systems é o método de aplicação das produções. Percorrendo uma *string* seqüencialmente, nas gramáticas de Chomsky as produções são aplicadas também de forma seqüencial, enquanto que nos L-Systems elas são aplicadas em paralelo. Esta diferença reflete a motivação biológica dos L-Systems. As produções pretendem simular as divisões celulares em organismos multicelulares, onde muitas divisões podem ocorrer ao mesmo tempo.

Como os L-Systems tem a propriedade única de gerar as produções em paralelo, qualquer problema que dependa de permutações para sua solução pode ser calculado simultaneamente ao invés de seqüencialmente, muitas vezes reduzindo o tempo médio para o cálculo dos resultados.

Por usar uma geometria simples para ilustrar resultados paralelos, tornou-se possível olhar para todos os valores produzidos por uma equação. Conseqüentemente, L-Systems tornam-se uma ferramenta útil para avaliar resultados e observar padrões surgidos no estudo de um dado problema.

2.2 GRAMÁTICAS E L-SYSTEMS

Muitas coisas se desenvolvem de tal maneira que produzem modelos fractais. Podemos citar como exemplo ramificações de árvores. Neste caso a própria similaridade se apresenta em diferentes escalas porque crescimento envolve

repetição do mesmo processo simples. Esses processos simples e repetitivos podem freqüentemente serem resumidos como um conjunto de regras simples.

L-Systems são conjuntos de regras e símbolos (também conhecidas como gramáticas formais) que modelam processos de crescimento. Um simples L-System L é uma gramática como a abaixo, que contém três elementos:

L = <V, w, P>

- ***ALFABETO:*** O alfabeto é um conjunto finito V de símbolos formais, normalmente letras a, b, c,..., porém pode conter outros caracteres. Eles podem ser de dois tipos:
    - *VARIÁVEIS*: São símbolos que denotam elementos que podem ser substituídos e são chamados de símbolos Não-Terminais.
    - *CONSTANTES*: São símbolos que denotam elementos que permanecem fixos, isto é, não podem ser substituídos. Dessa forma são chamados de símbolos Terminais.

- ***AXIOMA***: São expressões que definem como o sistema começa. O axioma (também chamado de inicializador) é uma *string* de símbolos de V. Dado V = {a,b,c} alguns exemplos de palavras são aabca, caab, b, bbc. O comprimento |w| de uma palavra w é o número de símbolos na palavra.

- ***PRODUÇÕES OU REGRAS:*** Definem como as variáveis devem ser trocadas por constantes ou outras variáveis. Em nossa notação, P é o conjunto de produções ou regras.

São permitidos nas regras mapeamentos de uma letra para a palavra vazia, ou para ela mesma. Se um símbolo não tem uma produção explicitamente definida, assumimos que ele é mapeado para ele mesmo. Neste caso, o símbolo é uma constante do L-System. Vejamos o exemplo abaixo.

Nosso exemplo será o Fibonacci L-System. Leonardo Fibonacci escreveu o livro “Liber Abaci” (Livro do Ábaco), onde representava a forma Hindu-arábica de números que proporcionava maior rapidez nas operações feitas através da numeração decimal quando comparadas às operações com números romanos, utilizados na época pela sociedade européia.

Fibonacci enunciou o seguinte problema: “Se tivermos um casal de coelhos que gera um novo casal ao fim de dois meses e depois um novo casal todos os meses, que também geram novos casais, quantos casais de coelhos teremos ao fim de n meses?”

A resposta é dada pela série 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233..., designada como Números de Fibonacci ou Seqüência de Fibonacci. Esta série tem a particularidade de à medida que se avança no número de casais, a razão entre um valor e o seu antecessor se aproxima de 1,6153.... Este valor é denominado número de ouro e era ensinado nas escolas de arte, pois era usado para se obter uma proporção adequada em obras de arte para garantir a harmonia e perfeição. Curiosos com este número, matemáticos começaram a pesquisar e descobriram que ele pode ser traduzido por um número irracional.

A descoberta da relação entre a seqüência de Fibonacci e o número de ouro, desencadeou uma série de pesquisas em busca de seqüências e padrões matemáticos na natureza. As descobertas permitem hoje que biólogos utilizem os padrões para organizar e reconhecer espécies. Dentre as descobertas relacionadas com este número estão exemplos como a da concha do *Nautilus*. À medida que o molusco vai crescendo constrói uma nova câmara maior que a anterior na proporção do número de ouro.

Considere a gramática simples, definida por L = <{a,b}, a, {a -> b, b -> ab}>. Nosso L-System tem a seguinte forma:

V = {a,b}
w = a
p_1: a -> b
p_2: b -> ba

Note que a e b são variáveis e que não existem constantes. O comportamento fascinante acontece quando observamos o L-System em movimento, evoluindo momento a momento. Vamos considerar um conjunto que contém todas as palavras possíveis formadas a partir do alfabeto V. Esse conjunto é denominado V*. A evolução de um L-System é definida como uma seqüência g_n, onde n = 0, 1, 2, 3,..., que em cada iteração g_n é uma palavra de V* que

evolui da iteração anterior g_{n-1} pela aplicação de todas as regras de produção para cada símbolo. A primeira iteração g_0 é o axioma w. As primeiras iterações do sistema de Fibonacci são respresentadas pela *figura 2.1*.

g_0	a
g_1	b
g_2	ba
g_3	bab
g_4	babba
g_5	babbabab
g_6	babbababbabba
g_7	babbababbabbababbabab

Figura 2.1 *Iterações de Fibonacci.*

Podemos interpretar a's e b's como formas de vida individuais e as produções como estágios de suas vidas. Após uma iteração, a forma de vida imatura a matura-se em um adulto b. Depois o adulto b será capaz de produzir um bebê ab a cada iteração. Dessa forma esse sistema simbólico dinâmico modela um tipo muito simples de dinâmica populacional.

A seqüencia de Fibonacci é definida por uma relação recorrente de segunda ordem:

$$F_0 = F_1 = 1$$

$$F_{n+2} = F_{n+1} + F_n \text{ para } n \geq 0$$

Pode-se verificar que para as poucas iterações apresentadas acima o número de organismos na n-ésima iteração g_n é $|g_n| = F_n$. A razão é que este sistema dinâmico, definido por um processo local que afeta cada organismo individual (regras de produção), exibe um processo global funcionando ao mesmo tempo, definido em termos da população inteira. Depois de algumas iterações conseguimos visualizar o padrão:

$$g_{n+2} = g_{n+1}g_n$$

Isso significa que a iteração n+2 consiste da iteração n+1 seguida da iteração n, por exemplo:

$g_7 = g_6g_5 = (babbababbabba)(babbabab)$

2.2.1 TIPOS DE L-SYSTEMS

O processo global para a seqüência de Fibonacci funciona porque este L-System é livre de contexto, isto é, cada regra de produção toma conta apenas de um único símbolo, não se importando com seus vizinhos. É possível considerar L-Systems sensíveis ao contexto, onde as regras de produção aplicam-se a um símbolo somente se este tem certos vizinhos. Os vizinhos à esquerda e à direita do símbolo considerado são identificados por < e > respectivamente. Vejamos o exemplo:

V = {a,b,c}
w = bbb
p_1: a>∅ -> b
p_2: a>b -> ∅
p_3: b>a -> c
p_4: b>b -> ba
p_5: c -> ∅

O L-System descreve uma população com uma dinâmica elaborada. Se um a não tem nada a direita >∅ (isto é o que a notação significa), ele matura um b. Se ele tem um adulto b à direita, o adulto o "mata" e ele desaparece. Se um adulto b tem uma criança a no lado direito, ele fica "velho" e se torna um c. Pela produção p_4 se dois b's ocorrem consecutivamente, eles se reproduzem e surge um a entre eles. Um organismo velho c morre depois de uma iteração. Em todos os outros casos os símbolos ficam os mesmos. A evolução deste L-System pode ser vista na *figura 2.2*.

g_0	bbb
g_1	babab
g_2	ccb
g_3	b

Figura 2.2 *Crescimento e declínio da população.*

Vamos introduzir a notação de ramificação nos L-Systems. Essa notação é dada pelos símbolos [e]. Quando o símbolo [é encontrado pelo L-System seu estado é salvo e o processo de ramificação se inicia. Ao encontrar o símbolo] o L-System recupera o estado anteriormente salvo, finalizando a ramificação e voltando ao seu estado anterior.

Considerando os L-Systems com ramificações a noção de vizinhança existe, porém de forma diferente. O vizinho a um determinado símbolo pode não ser o mais próximo na *string* gerada. Consideremos primeiramente um exemplo simples:

```
V = {F,+,-}
w = F
F -> FF[+F]F[-F]F
```

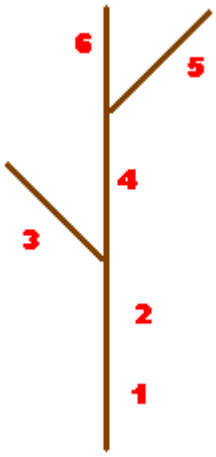

Figura 2.3 *Estrutura de ramificações no L-System.*

Vamos adotar a seguinte notação:

$$F \rightarrow F_1F_2[+F_3]F_4[-F_5]F_6$$

Notamos que o predecessor de F_3 é F_2. Porém o predecessor de F_4 não é F_3 e sim F_2! O mesmo ocorre com F_6 onde seu predecessor é F_4.

Com os sucessores ocorre o mesmo. O sucessor de F_2 é F_4. Em contra partida F_3 não tem sucessor. Vejamos agora um exemplo mais elaborado:

```
V = {+, -, F}
w = F1F1F1
0<0>0 -> 1
0<0>1 -> 1[-F1F1]
0<1>0 -> 1
0<1>1 -> 1
1<0>0 -> 0
1<0>1 -> 1F1
1<1>0 -> 1
1<1>1 -> 0
+ -> -
- -> +
#Ignore: +-F
```

Caso os símbolos `+`, `-` e `F` apareçam na *string* analisada serão ignorados para o cálculo dos predecessores e sucessores, conforme o `#Ignore`. Dessa forma, a *figura 2.4* representa algumas iterações.

g_0	F1F1F1
g_1	F1F0F1
g_2	F1F1F1F1
g_3	F1F0F0F1
g_4	F1F0F1[-F1F1]F1
g_5	F1F1F1F1[+F0F1]F1
g_6	F1F0F0F0[-F1F1F1]F1
g_7	F1F0F1F1[-F1F1][+F1F0F1]F1

Figura 2.4 *Iterações de um L-System sensível ao contexto.*

Até o momento, não estipulamos que para cada símbolo no alfabeto haja exatamente uma produção, apesar disto ser verdade para nossos primeiros exemplos. Se há de fato exatamente uma produção para cada símbolo, então o L-System é chamado de determinístico e a seqüência de iterações g_n é unicamente definida como uma seqüência de elementos de V*. Se houvesse mais do que uma

produção para um símbolo dado, digamos a -> w_1 e a -> w_2, precisaríamos de um critério para decidir quando aplicar cada regra. Uma possibilidade é usar uma das possíveis produções com probabilidades. Esses L-Systems são chamados de não-determísticos.

Baseado nos tipos de L-System podemos definir diversas classes. A presença da letra D na classe indica que o L-System é Determinístico. Os algarismos 0, 1 e 2 representam L-Systems livres de contexto, sensíveis ao contexto (somente à direita ou à esquerda) e sensíveis ao contexto (à direita e à esquerda) respectivamente. Observe abaixo:

- D0L-System: D representa L-System Determinístico e 0 livre de contexto.
- D1L-System: D representa L-System Determinístico e 1 sensível ao contexto somente à esquerda ou à direita.
- 1L-System: L-System não-determinístico e 1 representa L-System sensível ao contexto somente à esquerda ou à direita.
- 2L-System: L-System não-determinístico e 2 representa L-System sensível ao contexto à esquerda e à direita.

A classe mais simples é D0L-System. Para fornecer um entendimento intuitivo da principal idéia dos D0L-Systems, vamos considerar este exemplo dado por Prusinkiewicz e Lindenmayer. Considere *strings* constituídas de duas letras a e b (eles podem ocorrer muitas vezes em um *string*). Para cada letra especificamos uma regra de substituição. A regra a -> ab significa que a letra a deve ser substituída pela *string* ab, e a regra b -> a que a letra b deve ser substituída por um a.

Vamos assumir que o axioma seja constituído de uma letra simples b. No primeiro passo de derivação o axioma b é trocado através da produção b -> a. No segundo passo a é substituído por ab usando a produção a -> ab. A palavra ab consiste de duas letras, ambas são simultaneamente trocadas no próximo passo de derivação. Então, a é substituído por ab, b é substituído por a, e a *string* resultante será aba. Da mesma maneira (por trocas simultâneas de todas as letras), a *string* aba gera abaab que por sua vez gera abaababa, depois abaababaabaab, e assim por diante.

O estudo das iterações de subconjuntos formais de V* são originários da teoria das linguagens formais, advinda de Chomsky como uma forma matemática para discutir a formação e evolução de linguagens naturais. Por essa razão, qualquer subconjunto S de V* é chamada de linguagem. Linguagens L-Systems são exemplos de linguagens recursivamente enumeráveis. Alguns aspectos e características novas de L-Systems são a natureza paralela da evolução de uma palavra e a natureza dinâmica da linguagem que nos permite imaginar o crescimento no tempo.

2.2.2 L-SYSTEM DE THUE-MORSE

Com o nosso próximo exemplo de L-System apresentamos o sistema de Thue-Morse:

V = {a,b}
w = a
p_1: a -> ab
p_2: b -> ba

Podemos analisar a seqüência de iterações na tabela representada pela *figura 2.5*.

g_0	a
g_1	ab
g_2	abba
g_3	abbabaab
g_4	abbabaabbaababba

Figura 2.5 *Iterações de Thue-Morse.*

O número de organismos claramente dobra a cada iteração, então: $|g_n| = 2^n$. A primeira questão que devemos considerar é qual o processo global que emerge, se é que existe.

Para responder esta questão, observamos que a iteração g_n aparece no início de g_{n+1}. Se acompanharmos a seqüência, eventualmente concluímos g_n se repete e

em seguida g_n é invertido da seguinte forma: cada a é substituído por um b e cada b é substituído por um a. Para cada palavra w em V*, definimos R(w) como sendo a imagem da palavra onde trocamos a por b e b por a. Então o processo global é:

$$g_{n+1} = g_n R(g_n)$$

Já que g_n é o começo de g_{n+1}, este L-System produz no infinito uma seqüência g_∞ de a's e b's que começa:

$$g_\infty = \text{abbabaabbaababbabaababbaabbabaab...}$$

Esta seqüência tem muitas propriedades importantes e é conhecida como a seqüência de Thue-Morse. Axel Thue surgiu com esta seqüência em 1912 em seu estudo de linguagens formais. Marston Morse redescobriu a mesma seqüência em 1917 no estudo das dinâmicas das superfícies. Podemos produzir uma fórmula muito interessante para os termos desta seqüência se trocarmos a por 1 e b por -1. No caso, a aplicação da repetição de R é o mesmo que multiplicação por -1. Escrevemos a seqüência como:

$$g_\infty = c_0 c_1 c_2 c_3 ...$$

onde cada c_n é +1 ou -1. A n-ésima iteração é:

$$g_n = c_0 c_1 c_2 ... c_{2^n-1}$$

Então a iteração n+1 consiste desta *string* g_n seguida por:

$$(-c_0)(-c_1)(-c_2)...(-c_{2^n-1})$$

Isto estabelece a fórmula:

$$c_{2^n+i} = -c_i$$

para todo $0 \leq i \leq 2^n-1$. Para isto funcionar para qualquer c_m dado, devemos expressar m como uma soma de potências de 2, isto é, precisamos encontrar a expansão binária de m:

$$m = 2^{k1} + 2^{k2} + ... + 2^{kt}$$

com $K_1 > K_2 > .. > K_t$. Por exemplo,

$$37 = 2^5 + 2^2 + 2^0$$

Em dígitos binários, escreveríamos $37 = 100101_2$. O número de potências de 2 ocorrendo é igual ao número de 1's na expansão binária. Então aplicando nossa fórmula temos:

$$C_{37} = - C_5 = (-1)^2 C_1 = (-1)^3 C_0 = -1$$

já que $C_0 = 1$. Em geral,

$$C_m = (-1)^t$$

onde t é o número de 1's na expansão binária de m. Vejamos um outro exemplo:

$$C_{23} = -C_7 = + C_3 = - C_1 = + C_0 = 1$$

Thue provou um teorema muito interessante sobre a seqüência Thue-Morse.

2.2.2.1 Teorema (Thue-Morse é Cubo-Livre)

Não há *substring* da seqüência Thue-Morse da forma www em nenhuma *string* finita w de a's e b's. Isto significa que não há *substring* aaa ou ababab ou babbabbab....

2.2.2.2 Teorema (Thue-Morse é Recorrente)

Seja n um inteiro positivo. Existe um outro inteiro positivo N (maior que n normalmente) tal que, dada alguma *substring* w de comprimento n na seqüência Thue-Morse podemos garantir que toda *substring* W de comprimento N contêm uma cópia da *string* w. Isto significa que toda *string* finita que ocorre na seqüência Thue-Morse ocorre várias vezes infinitamente. Uma estimação explícita de N pode ser provada: se k é o menor inteiro tal que $2^k \geq n$, então podemos garantir que qualquer *substring* de comprimento 2^{k+4} na seqüência Thue-Morse contém a *string* dada w de comprimento n.

2.3 A CURVA DO DRAGÃO

Como dissemos anteriormente, uma das metas originais ao se introduzir L-Systems foi modelar crescimentos (desenvolvimentos) de vários tipos biológicos. Para tal, temos que assumir alguns significados naturais para os símbolos formais que são manipulados no L-System. Nessa seção, começaremos a verificar como as regras de produção dos L-Systems podem produzir interpretações geométricas; os L-Systems podem produzir incríveis objetos geométricos dentre vários tipos de curvas fractais, formas que imitam o mundo natural e *tilings* (preenchimento de superfícies) de vários tipos.

Começaremos com um L-System simples:

```
V = {a,b}
w = a
p1: a -> ab
p2: b -> ab
```

As iterações simplesmente duplicam as *strings* geradas anteriormente: a, ab, abab, abababab, etc. Para a e b, associamos as formas da *figura 2.6*.

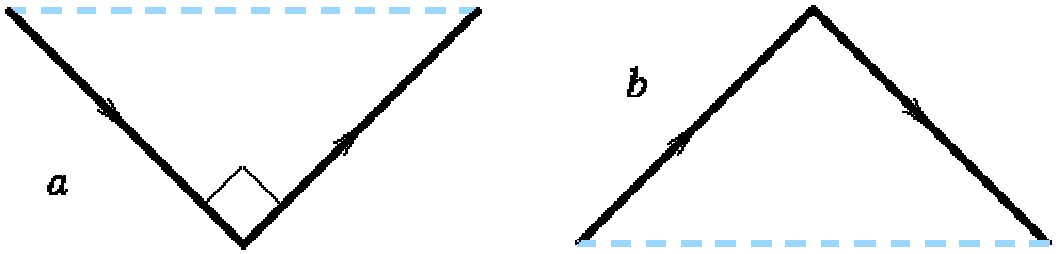

Figura 2.6 *Formas iniciais da curva do Dragão.*

As linhas em negrito indicam a curva desenhada. A linha azul pontilhada é para referência. Cada triângulo é retângulo e isósceles. As produções de nosso L-System representam uma mudança na geometria dessas configurações. Quando uma curva em negrito é desenhada formam-se triângulos isósceles com as linhas azuis pontilhadas nas antigas linhas em negrito. As duas novas configurações são observadas na *figura 2.7*.

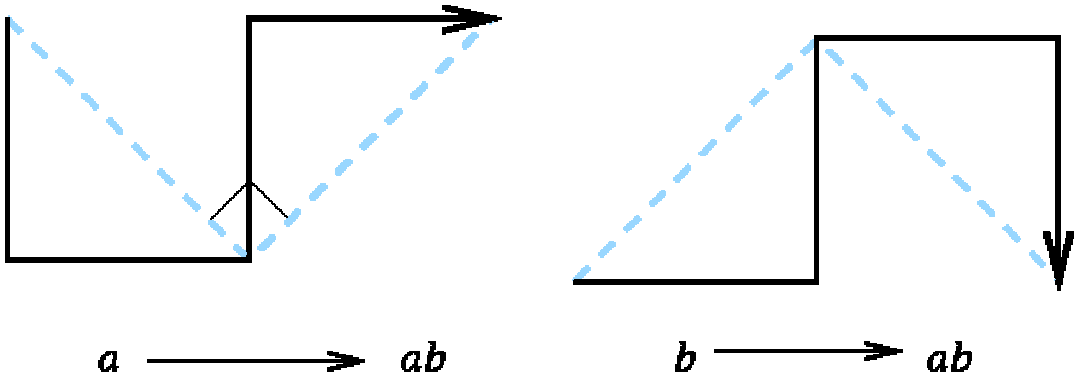

Figura 2.7 *Novas configurações da curva do Dragão.*

As quatro primeiras iterações desse L-System são exibidas na *figura 2.8*, começando com um triângulo a.

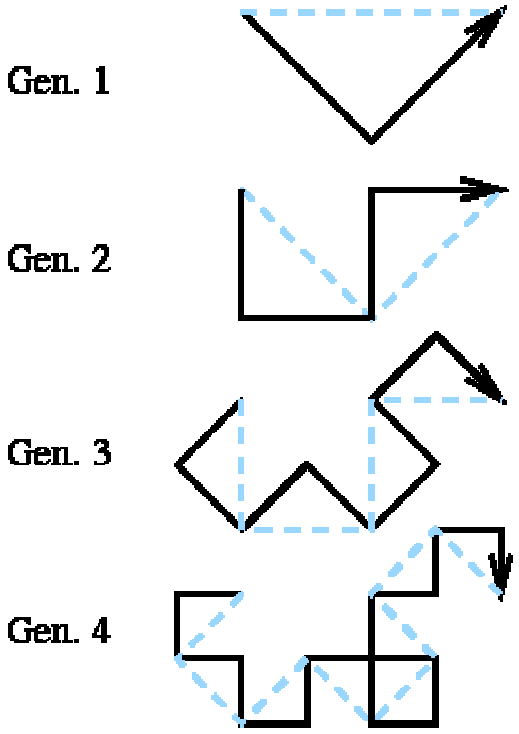

Figura 2.8 *Iterações da curva do Dragão.*

À medida que construímos as iterações, uma curva conhecida como Dragão aparece.

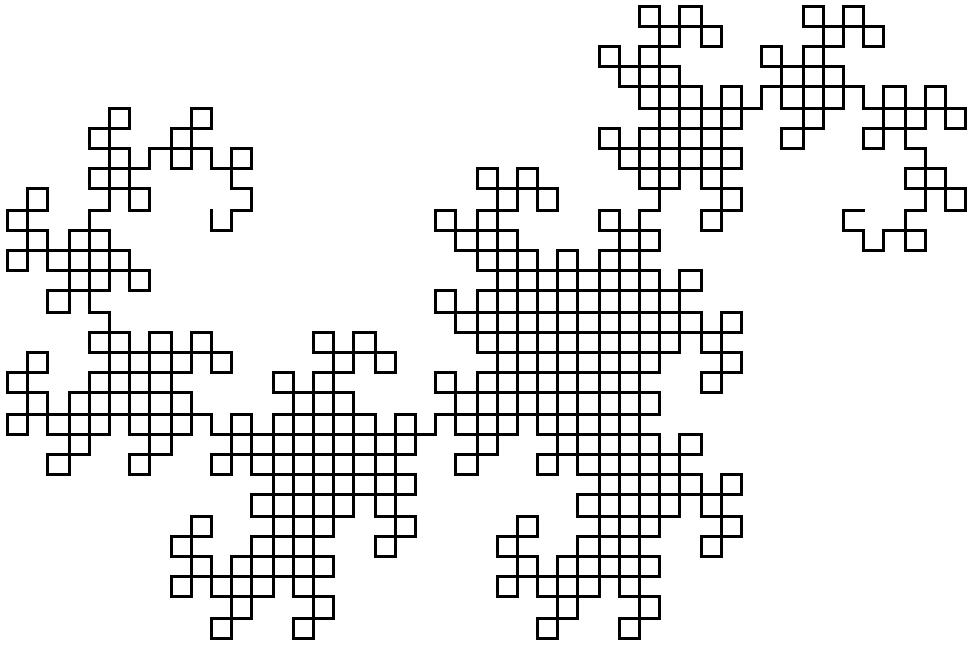

Figura 2.9 *Décima iteração da Curva do Dragão.*

Começamos a ver um comportamento emergente nesse sistema dinâmico. A curva aparentemente gera infinitamente vários quadrados pequenos. Estes

quadrados fecham-se em uma seqüência de "ilhas", onde cada ilha é conectada a outra por um segmento simples. A forma final é o fractal Dragão (*figura 2.10*).

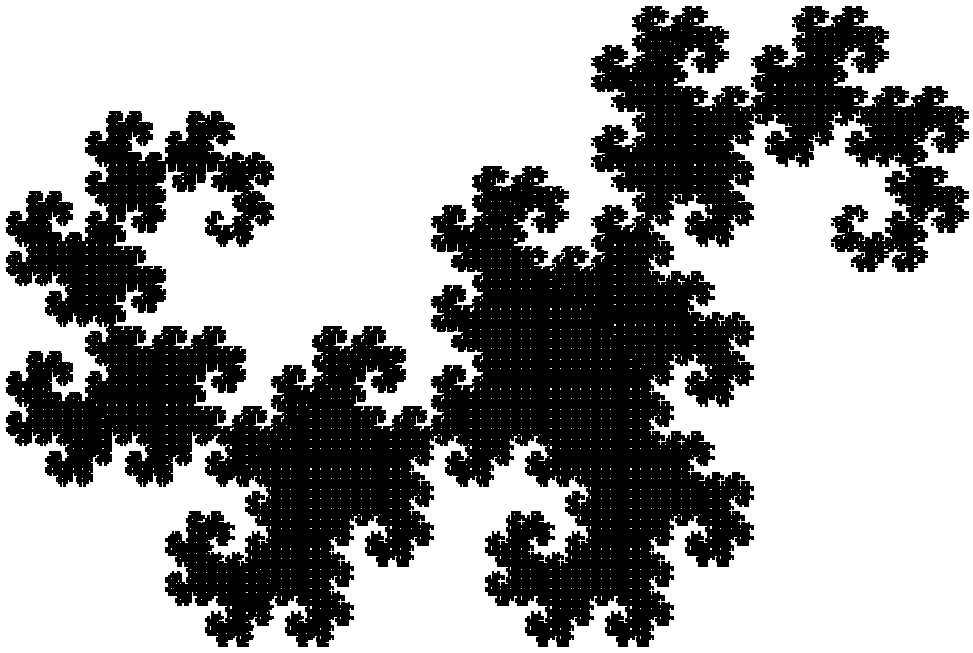

Figura 2.10 *Fractal Dragão.*

Existe um processo natural e dinâmico de dobramentos na criação de curvas do Dragão. Comece com um pedaço retangular de papel. Dobre a metade direita do papel sobre a metade esquerda, com uma dobra grossa no meio. Pegue o papel dobrado e o dobre novamente. Continue esse processo de dobramento mais algumas vezes. A aparência da aresta é mostrada no lado esquerdo da *figura 2.11*. Após alguns dobramentos, desdobre o papel, e estique cada dobra em um ângulo de exatamente 90º. A figura resultante é o nosso Dragão. A metade direita da *figura 2.11* mostra os resultados para as primeiras iterações.

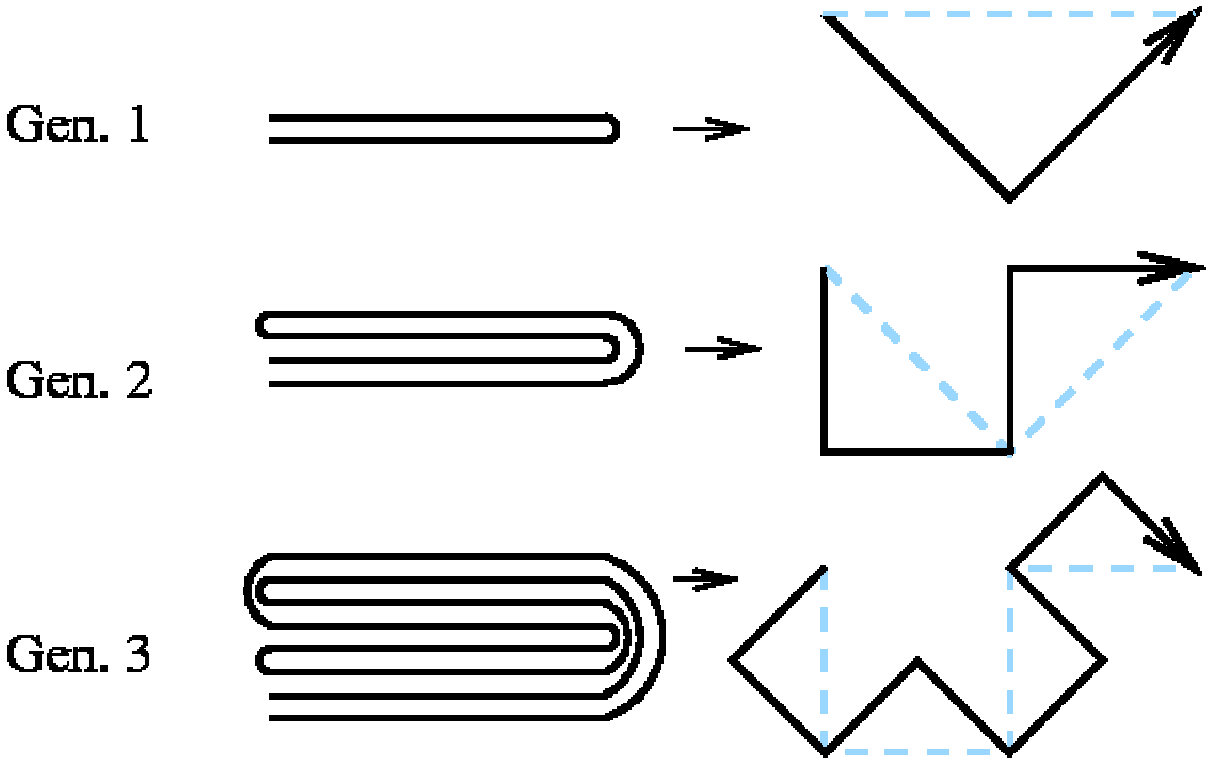

Figura 2.11 *Dobramento de Papéis.*

Existe uma outra codificação natural de dragões que podemos ver a seguir. Percorrendo a curva do começo ao fim, cada mudança de direção é para esquerda ou para direita. Portanto, cada iteração do Dragão corresponde a seqüências de L′s (esquerda) e R′s (direita). Na *figura 2.12* podemos ver a quarta iteração com todas as suas mudanças de direções codificadas como L ou R.

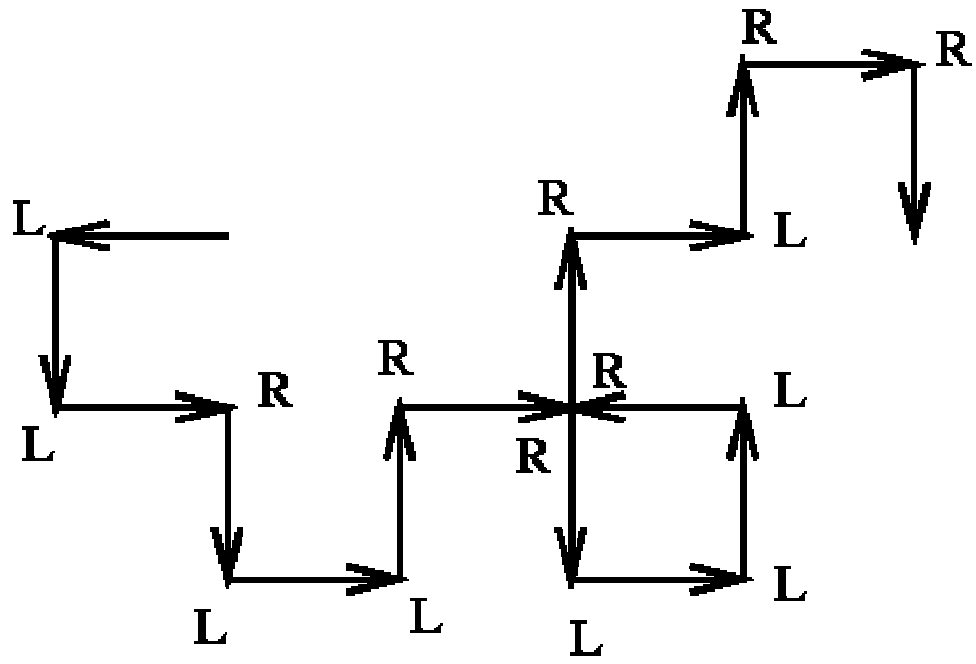

Figura 2.12 *Quarta iteração codificada como mudanças L e R*

As seqüências correspondentes até a quarta iteração estão listadas na *figura 2.13*.

1 dobra	L
2 dobras	LLR
3 dobras	LLRLLRR
4 dobras	LLRLLRRLLLRRLRR

Figura 2.13 *Iterações uilizando mudanças L e R.*

Há uma vaga similaridade com a seqüência Thue-Morse. De fato, existe uma relação. A relação com o processo de dobramento de papéis sugere muitas variações interessantes de curvas do Dragão. Por exemplo, ao invés de sempre dobrar o papel colocando a metade direita em cima da metade esquerda, podemos dobrar a metade esquerda sobre a direita. Dependendo da seqüência de instruções de dobramento de papéis, podemos construir diferentes curvas do Dragão.

2.4 OS GRÁFICOS DE TARTARUGA

Seymour Papert inventou “Os Gráficos de tartaruga” que são um sistema para traduzir uma seqüência de símbolos em movimentos em um autômato, que nesse contexto seria a tartaruga, para uma visão gráfica. A idéia inicial foi desenvolver algo

capaz de ajudar crianças a entenderem com mais facilidade geometria. Isto se tornou um sistema ideal para dar uma interpretação geométrica a dinâmica dos L-Systems. Definimos d como distância percorrida pela tartaruga em um passo. Também definimos δ como ângulo, normalmente 360°/n, sendo n inteiro. Temos as seguintes definições para o novo sistema:

- **F:** Desenha para frente um passo de tamanho d na direção corrente;
- **f:** Move para frente um passo de tamanho d na direção corrente;
- **+:** Gira um ângulo δ no sentido anti-horário;
- **-:** Gira um ângulo δ no sentido horário;
- **[:** Empilha o estado atual da tartaruga;
- **]:** Desempilha o topo para estado atual da tartaruga.

Como um primeiro exemplo do uso dos símbolos acima, vejamos o L-System da curva de Koch:

V = {F, +, -}
w = F
p_1: F+F--F+F

A iteração 0 é apenas uma linha reta; vamos assumir que a cabeça da tartaruga está na direção do eixo horizontal positivo. Assumimos o ângulo δ como 60° = 360°/6. Na *figura 2.14* são mostrados todos os passos para a construção da curva e na *figura 2.15* podemos visualizar algumas iterações.

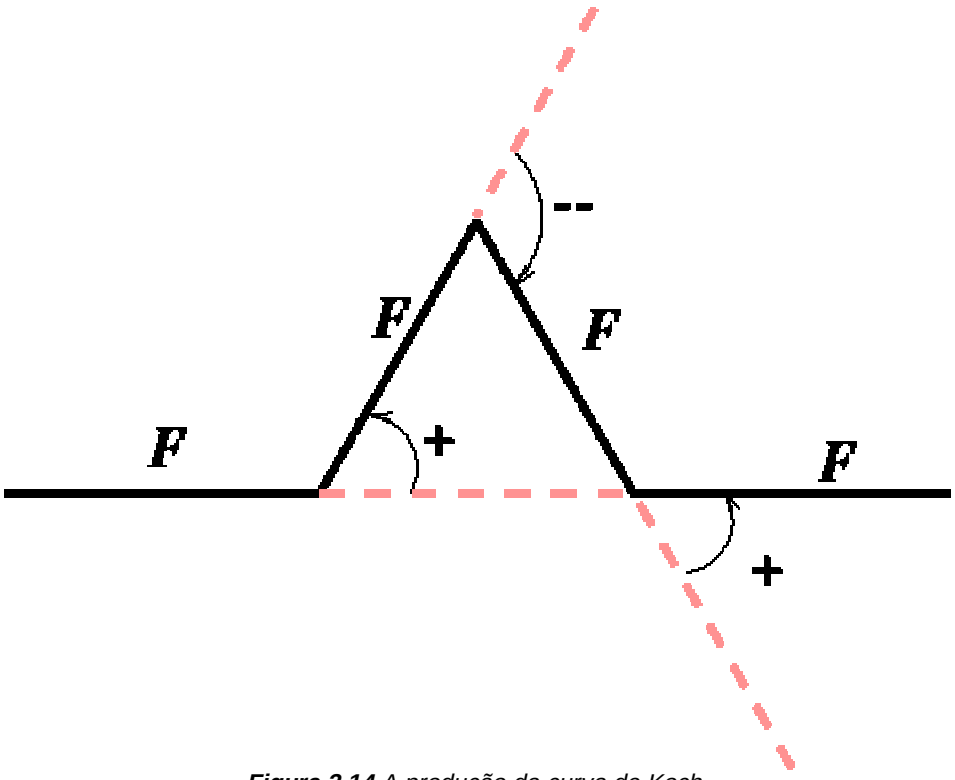

Figura 2.14 *A produção da curva de Koch.*

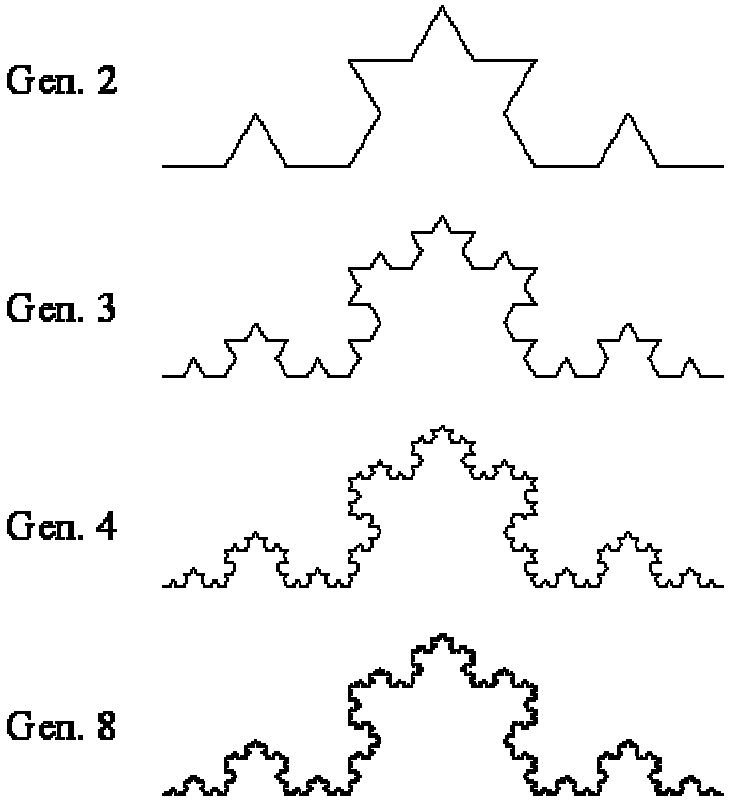

Figura 2.15 *Iterações da curva de Koch.*

O limite de uma curva fractal causou preocupações entre os matemáticos do século dezenove que estavam trabalhando para construir os rigorosos fundamentos do Cálculo. A grande preocupação era com a reta tangente em qualquer ponto de uma curva. A curva fractal representada pela *figura 2.14* é um exemplo de curva que não tem uma reta tangente bem definida em qualquer ponto. De fato, a curva parece estar dando voltas de 60^o ou 120^o em todos os pontos.

O L-System de Koch cresce rapidamente em termos de número de segmentos de reta desenhados. Podemos comprovar que cada segmento de reta é substituído por outros quatro na próxima iteração, a n-ésima iteração tem 4^n segmentos. Por exemplo, a oitava iteração da *figura 2.15* tem 65536 segmentos.

2.4.1 RAMIFICAÇÕES E AGRUPAMENTOS

Muitas formas biológicas são ramificadas ou fragmentadas tanto em aparência quanto no crescimento. Para permitir que ramificações ocorram e sejam interpretadas pelos gráficos de tartaruga, utilizamos os símbolos "[" (*push*) e "]" (*pop*). Dessa forma a tartaruga desenha a ramificação e depois volta a posição antes do ramo desenhado. Começaremos com uma ramificação simples onde um tronco principal gera uma ramificação de apenas um lado.

$\delta = 36^o$
Axioma = F
F = F[+F]F

Isso pode ser visto como a substituição de cada aresta pela configuração da *figura 2.16*.

Figura 2.16 *Ramificação Simples.*

A mudança de ângulo e um passo ocorrem pela interpretação dos operadores *pop* e *push*. Podemos verificar a seqüência de iterações na *figura 2.17*.

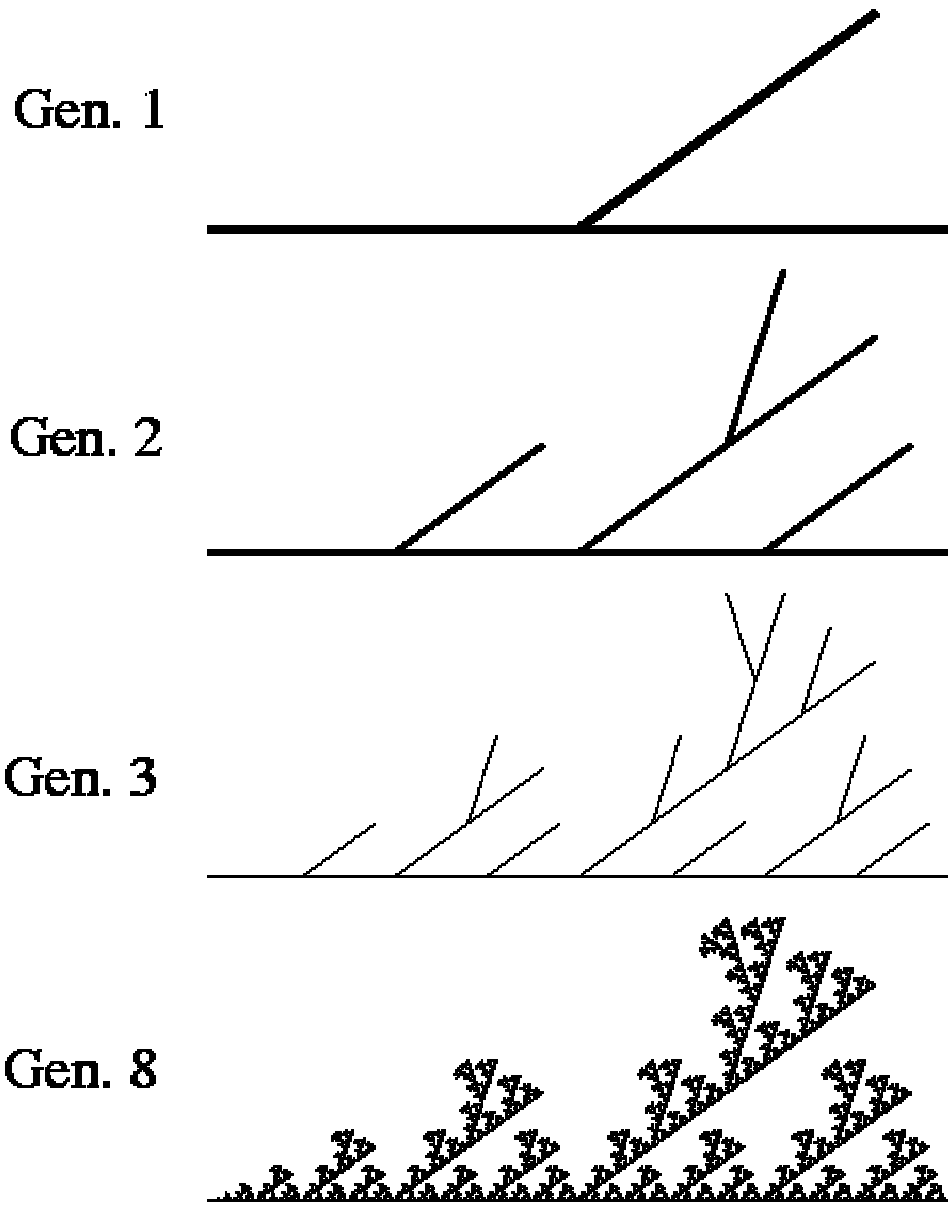

Figura 2.17 *Ramificações nas diversas iterações.*

Mesmo com esse simples L-System, pode-se visualizar um ramo de um arbusto ramificando-se recursivamente. Notavelmente, muitas imagens realísticas da biologia podem ser desenhadas por L-System de forma fácil.

Para finalizar essa seção, mostramos uma figura (*figura 2.18*) do sistema implementado. O modelo é uma ramificação simples, levemente assimétrico e com ramificações levemente encurvadas .

δ = 22.5º

Axioma = F

F = FF-[-F+F+F]+[+F-F-F]

O ângulo 360º/16º = 22.5º permite um encurvamento suave. O tronco principal existe somente antes da ramificação.

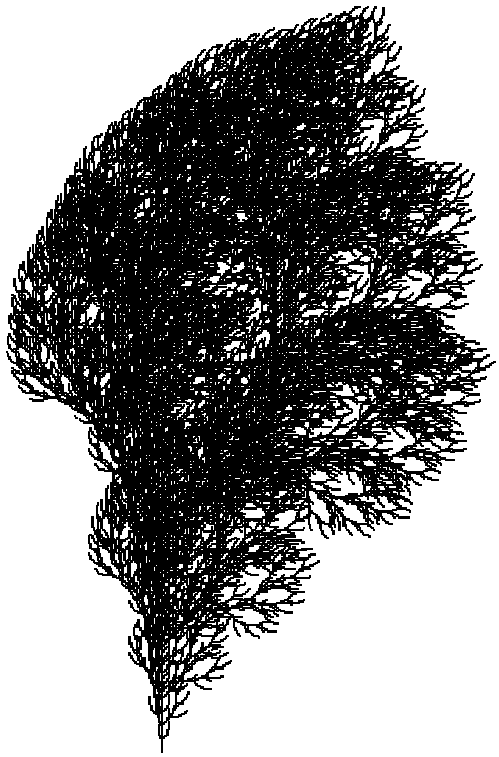

Figura 2.18 *L-System arbusto, quinta iteração.*

2.5 SIMILARIDADE PRÓPRIA DOS L-SYSTEMS

Vamos recordar algumas figuras desenhadas por L-Systems que consideramos. O objeto geométrico limitante é um exemplo do que geralmente chamamos, na teoria de sistemas dinâmicos, de indutor. Geralmente, isso significa alguma situação física na qual o sistema de alguma forma converge. Para alguns sistemas dinâmicos simbólicos, os indutores são seqüências infinitas produzidas. Para os L-System derivados dos gráficos de tartaruga, o indutor é a curva ou forma

limite. Para se chegar à forma final, devemos em cada iteração calcular o tamanho do passo d o decrementando de um número apropriado.

Quando vemos a forma final (ou aproximada), a evolução muitas vezes é visualmente aparente. Considere a primeira Curva do Dragão da *figura 2.19*. Nessa figura, mostramos como a forma final pode ser dividida em duas figuras congruentes.

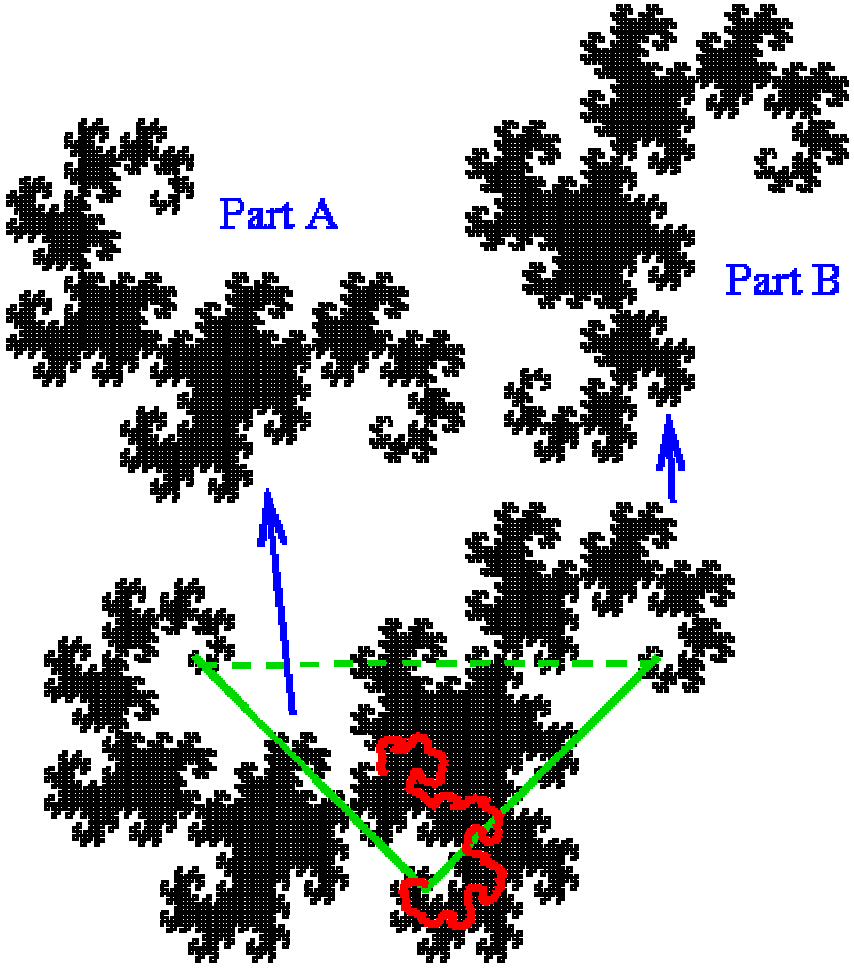

Figura 2.19 *As duas metades do Dragão inferior são marcadas como partes A e B. A divisão é marcada em vermelho.*

O triângulo retângulo isósceles original é marcado em verde na *figura 2.19*. As duas metades menores são cada uma construída por um segmento de comprimento $\sqrt{2}/2$ vezes maior que o Dragão original. Como as duas metades são congruentes, uma metade deve encaixar exatamente no topo da outra metade depois de uma translação e uma rotação de $90°$. Esse é um exemplo de similaridade própria: similaridade de um objeto inteiro com seus pedaços menores.

Para muitos L-Systems gráficos listados em vários programas, podemos associar dois números: um "número de pedaços congruentes" que exprime o número de partes em que se pode dividir um L-System, e um "fator de escalonamento" que mede cada parte proporcionalmente ao objeto original. Para o Dragão fractal, o número de pedaços é 2 e o fator de escalonamento é $\sqrt{2}$. Esses números podem ser parcialmente deduzidos pelas regras dos L-Systems, embora um método universal para se fazer tal cálculo não seja conhecido. Para o Dragão, vemos nas regras de produção do L-System que cada segmento reproduz dois pedaços de tamanho $\sqrt{2}/2$ menor que o original.

Como um segundo exemplo, considere a curva de Koch, mostrada na *figura 2.20* dividida em partes similares. A regra de produção converte cada segmento em 4 novos segmentos de comprimento igual a $1/3$ do original. Portanto, não é surpresa ver quatro pedaços similares, cada um com um terço do tamanho da curva de Koch original.

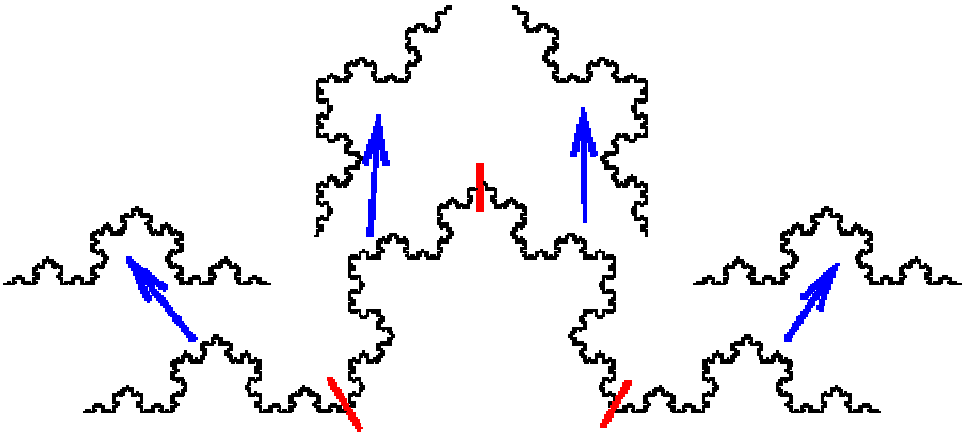

Figura 2.20 *Dividindo a curva Koch.*

Similaridade própria é um conceito chave na teoria de sistemas dinâmicos e da geometria fractal resultante.

2.6 UMA LINGUAGUEM PARA OS L-SYSTEMS

O Futuro para L-System é promissor. Podemos nos deparar com pesquisas e várias aplicações voltadas para esse assunto. Uma das grandes inovações é uma linguagem simples chamada LinF, que descreve L-Systems 3D, para ser interpretada pelo método da tartaruga.

Um programa interpretado por essa linguagem é formado por seis partes: ângulos, linhas, brancos, axioma, regras e cores. Podem aparecer em qualquer ordem no programa. Vejamos:

- *Axioma:* Define o axioma do L-System.
- *Linhas:* Define quais símbolos representam linhas nas regras de produção.
- *Brancos:* Define quais símbolos representam linhas em branco (pulos da tartaruga).
- *Ângulos:* Nessa seção o ângulo é definido. A primeira parte define o ângulo e a segunda a curvatura.
- *Regras:* Nesse módulo é descrito todas as regras de produção da forma fractal. A parte da esquerda é composta por um identificador e a da direita por uma seqüência de expressões. Podem existir não-terminais para facilitar a descrição de um fractal.
- *Cores:* Nessa seção é especificado a lista de cores utilizadas na construção do fractal.

2.6.1 REPRESENTAÇÃO DE FRACTAIS NO LINF

O LinF como é baseado no formalismo de L-System, permite que fractais sejam descritos por *strings* de símbolos. Estes símbolos podem ser terminais, não-terminais, números e palavras reservadas. Vejamos as 9 palavras reservadas do LinF:

- `[`: Salva o estado em pilha.
- `]`: Restaura o estado de topo da pilha e desempilha.
- @: Multiplica a espessura da linha corrente pelo número real que o precede.
- ?: Este símbolo não tem utilidade no processo de desenho. Ele é útil na aplicação das regras, significa que o próximo símbolo tem chance de não ser expandido. É precedido por um número real do intervalo `(0,1)` que especifica a probabilidade da expansão.
- +: Gira de um ângulo definido pelo identificador que segue este símbolo.

- `-`: Mesmo significado do símbolo anterior, porém este causa uma rotação menos o ângulo especificado.
- `{`: Este símbolo troca a cor corrente para a próxima cor da lista de cores. Caso a cor corrente seja a última da lista o símbolo não ocasiona nenhuma modificação na cor.
- `}`: restaura a cor anterior.
- `!`: Inverte a interpretação dos símbolos `+` e `-`.

2.6.2 EXEMPLO

```
angles
  a: (.0, [.1 .. .7])
  b: ([((pi/3) - 0.3) .. ((pi/3) + 0.3)], .0)
  c: (.0, [.1 .. .6])
  d: (.0, pi)

lines f

axiom 2.0@ f

rules
  f = 0.5 @ g v;
  v = { u + b u + b u + b u + b u + b u};
  u = 0.5 ? t;
  t = [+a f];
  j = + c

colors
        0: (4, 4, 2)
        1: (90, 90, 46)
        2: (156, 156, 78)
        3: (188, 188, 120)
        4: (209, 209, 163)
```

Figura 2.21 *Exemplo de um programa escrito em LinF*

Figura 2.22 *Plantas desenhadas a partir do programa da Figura 2.21.*

2.7 EXEMPLOS DE L-SYSTEM

2.7.1 FIGURAS EM GERAL

2.7.1.1 Quádrica de Koch

```
Axioma: F+F+F+F
δ: 90º
Regra: F -> F+F-F-FF+F+F-F
```

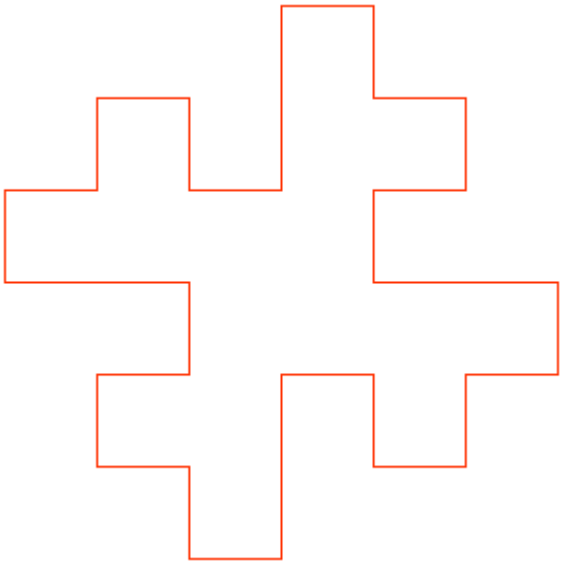

Figura 2.23 *1 iteração da Quádrica de Koch.*

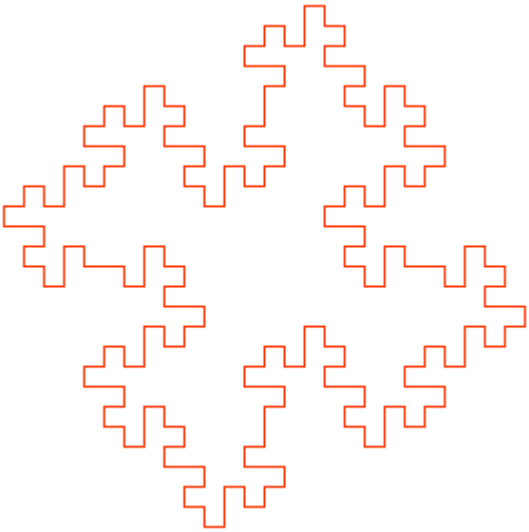

Figura 2.24 *2 iterações da Quádrica de Koch.*

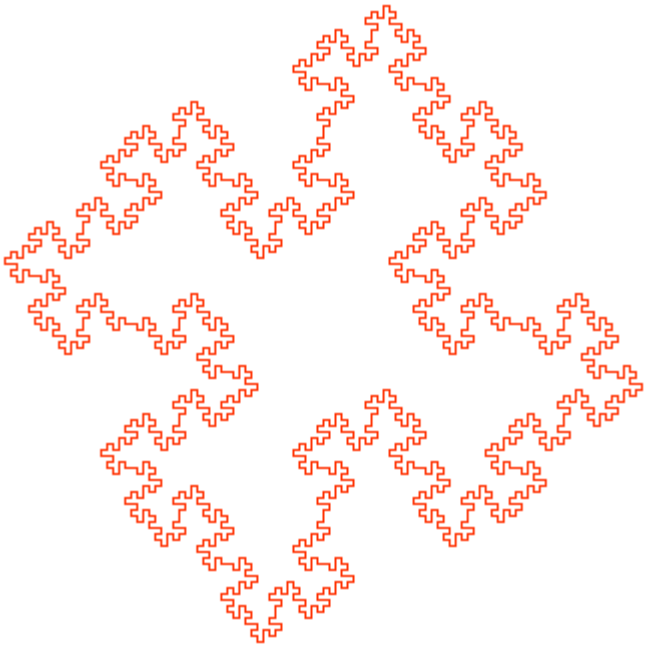

Figura 2.25 *3 iterações da Quádrica de Koch.*

2.7.1.2 Lagoa de Koch

```
Axioma: F-F-F-F
δ: 90º
Regras:
F -> F-a+FF-F-FF-Fa-FF+a-FF+F+FF+Fa+FFF
a  -> ffffff
```

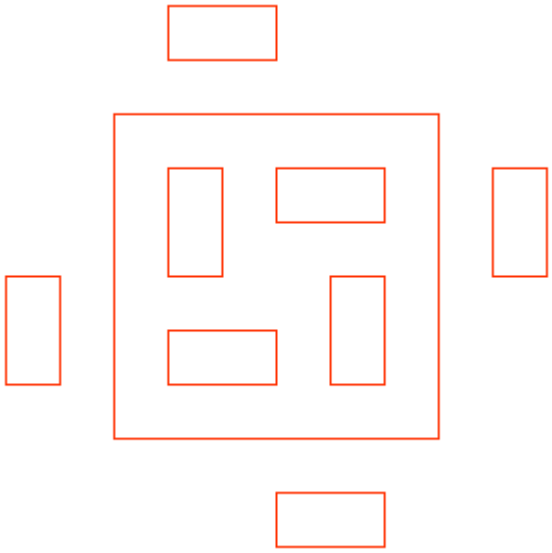

Figura 2.26 *1 iteração da Lagoa de Koch.*

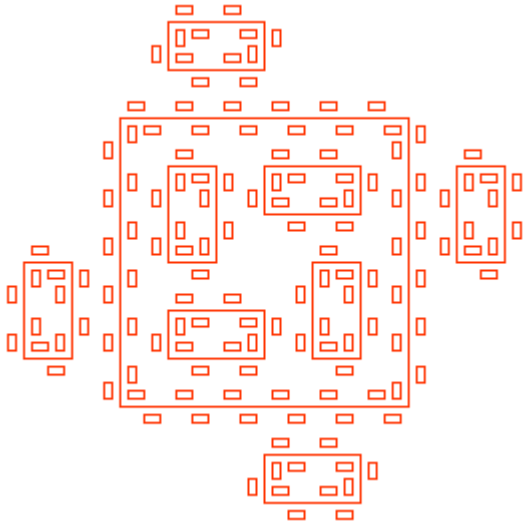

Figura 2.27 *2 iterações da Lagoa de Koch.*

2.7.1.3 Curva de Koch

```
Axioma: F+F+F+F
δ: 90º
Regra: F -> FF+F+F+F+F+F-F
```

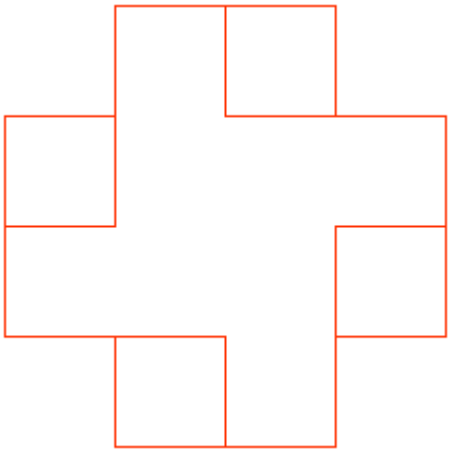

Figura 2.28 *1 iteração da Curva de Koch.*

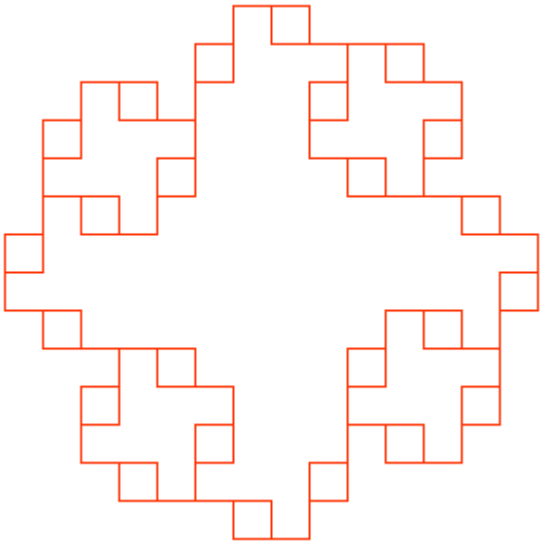

Figura 2.29 *2 iterações da Curva de Koch.*

Figura 2.30 *4 iterações da Curva de Koch.*

2.7.1.4 Curva de Koch

`Axioma: F+F+F+F`

δ: 90º

Regra: F -> FF+F+F+F+FF

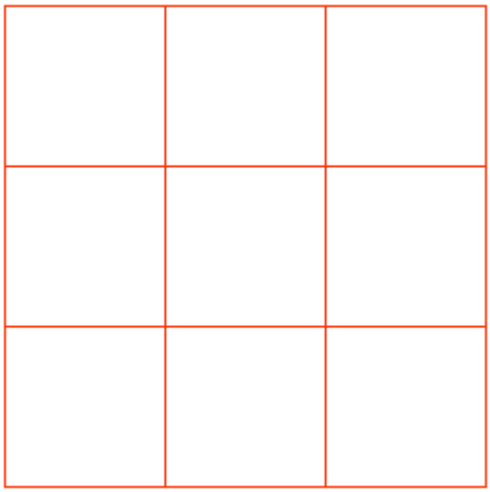

Figura 2.31 *1 iteração da Curva de Koch.*

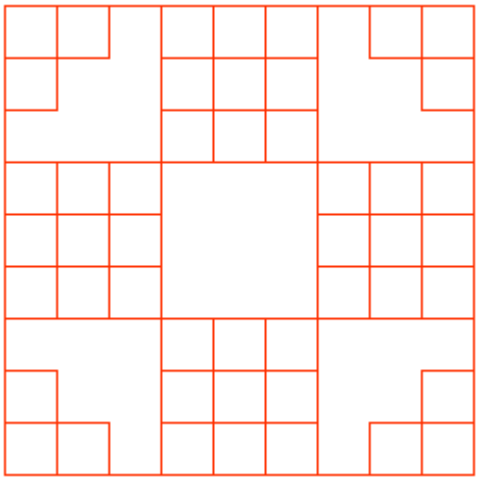

Figura 2.32 *2 iterações da Curva de Koch.*

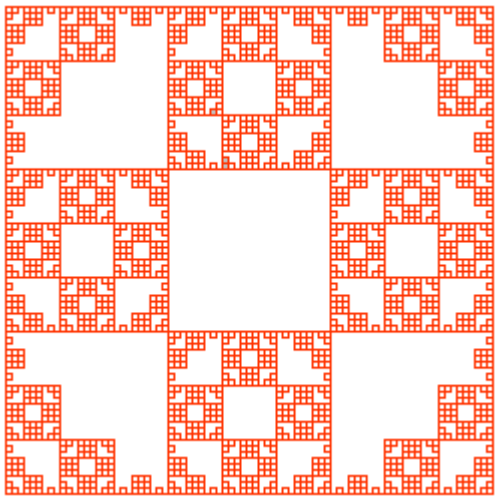

Figura 2.33 *4 iterações da Curva de Koch.*

2.7.1.5 Curva de Koch

```
Axioma: F+F+F+F
δ: 90º
Regra: F -> FF+F++F+F
```

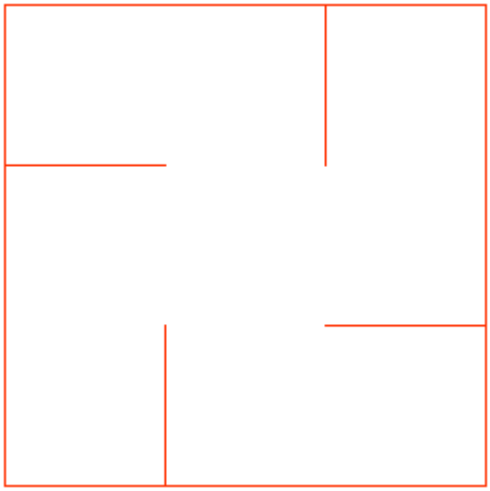

Figura 2.34 *1 iteração da Curva de Koch.*

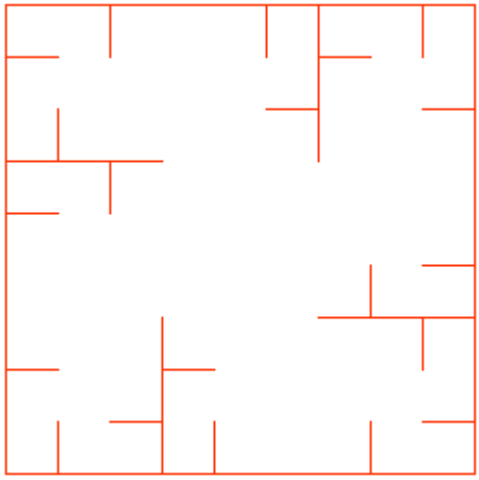

Figura 2.35 *2 iterações da Curva de Koch.*

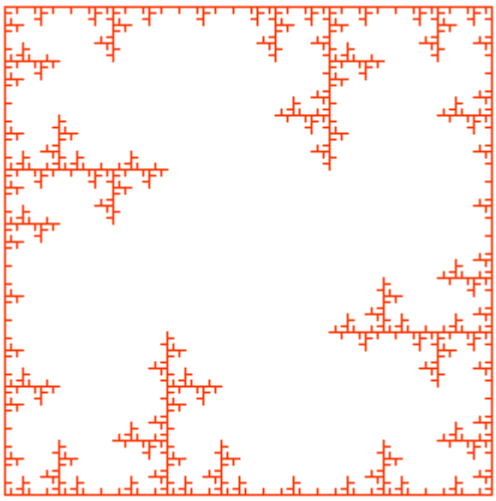

Figura 2.36 *4 iterações da Curva de Koch.*

2.7.1.6 Curva do Dragão

```
Axioma: Fl
δ: 90º
Regras:
l -> l+rF+
r -> -Fl-r
```

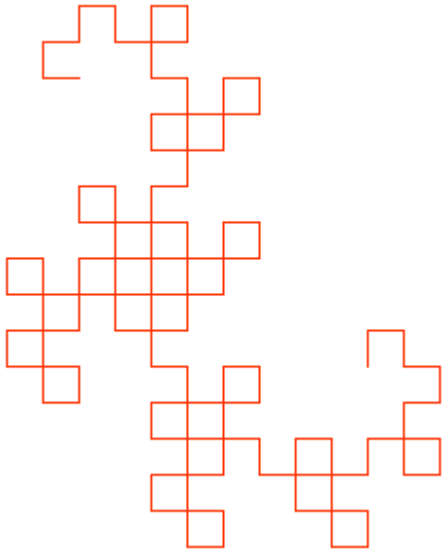

Figura 2.37 *7 iterações da Curva do Dragão.*

Figura 2.38 *11 iterações da Curva do Dragão.*

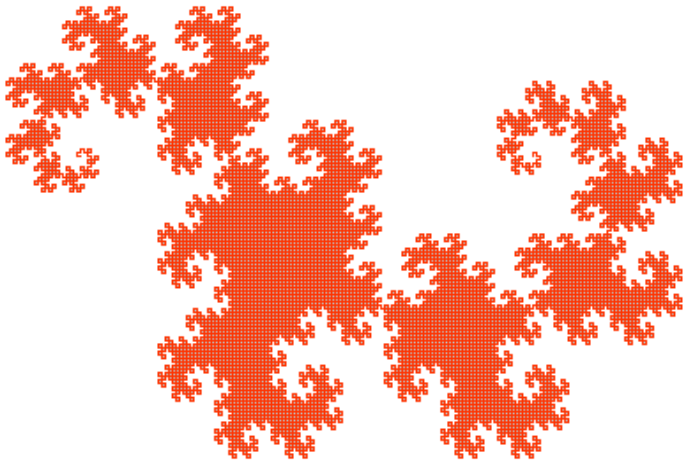

Figura 2.39 *14 iterações da Curva do Dragão.*

2.7.1.7 Curva de Peano

```
Axioma: X
δ: 90º
Regras:
X -> XFYFX+F+YFXFY-F-XFYFX
Y -> YFXFY-F-XFYFX+F+YFXFY
```

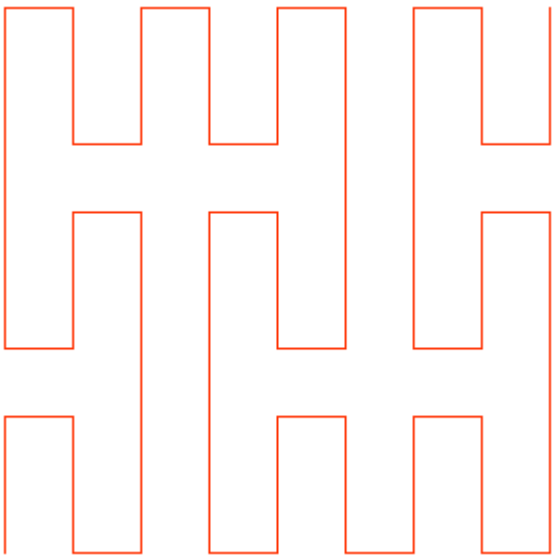

Figura 2.40 *2 iterações da Curva de Peano.*

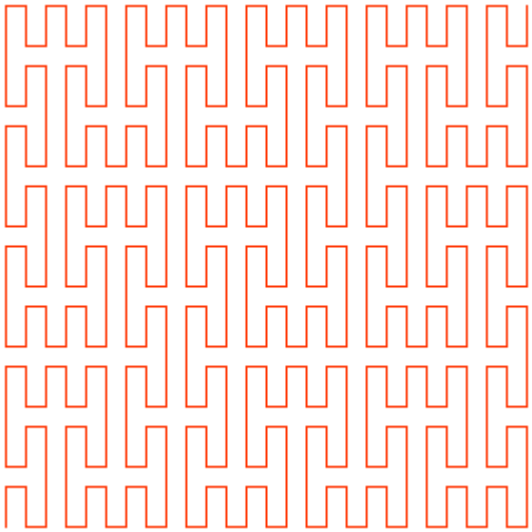

Figura 2.41 *3 iterações da Curva de Peano.*

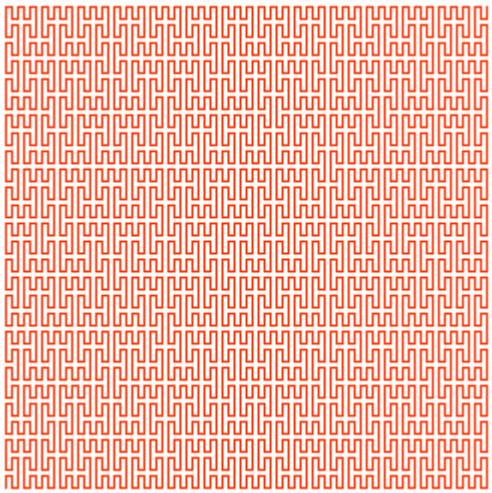

Figura 2.42 *4 iterações da Curva de Peano.*

2.7.1.8 Espiral de Tiling

```
Axioma: AAAA
δ: 15º
Regras:
A -> X+X+X+X+X+X+
X -> [F+F+F+F[---X-Y]+++++F++++++++F-F-F-F]
Y -> [F+F+F+F[---Y]+++++F++++++++F-F-F-F]
```

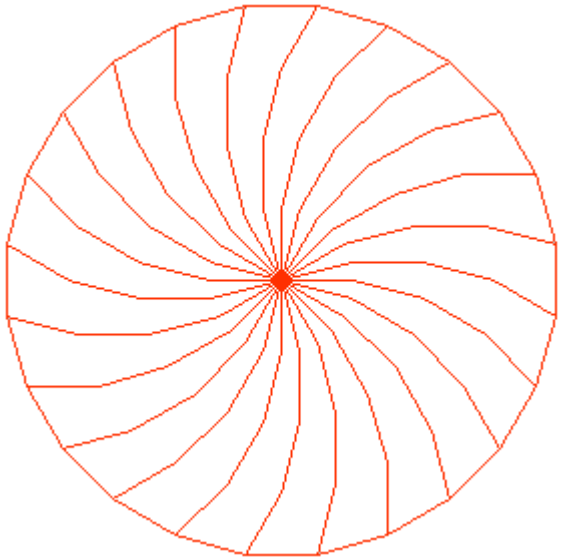

Figura 2.43 *2 iterações da Espiral de Tiling.*

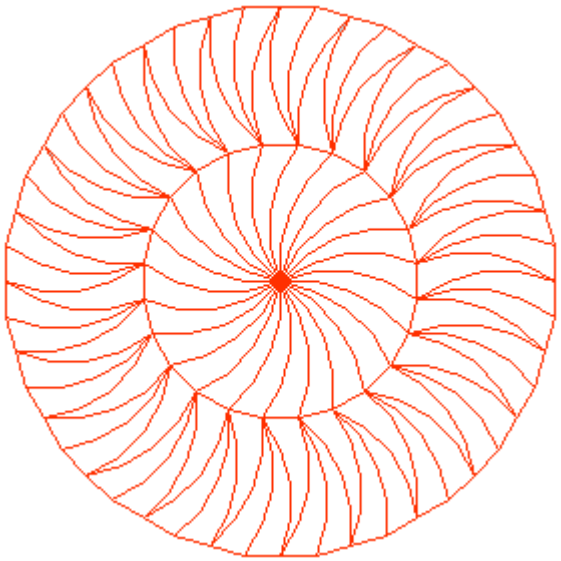

Figura 2.44 *3 iterações da Espiral de Tiling.*

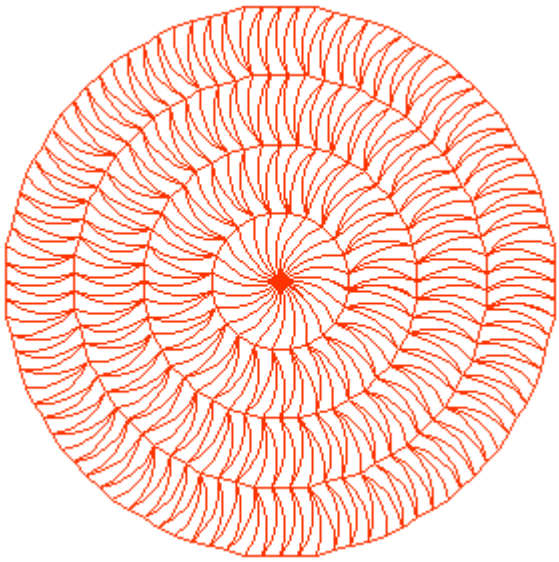

Figura 2.45 *5 iterações da Espiral de Tiling.*

2.7.2 PLANTAS

2.7.2.1 Planta 1

Axioma: F

δ: 22,7º

Regra: F -> F[+F]F[-F]F

Figura 2.46 *2 iterações da Planta 1.*

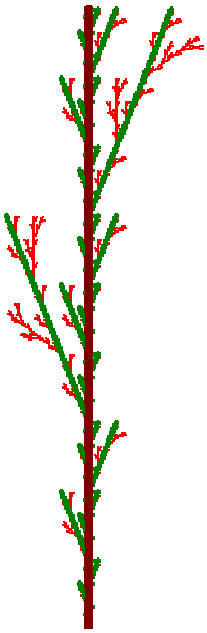

Figura 2.47 *4 iterações da Planta 1.*

2.7.2.2 Planta 2

Axioma: X

δ: 22,5º

Regras:

X -> F-[[X]+X]+F[+FX]-X

F -> FF

Figura 2.48 *3 iterações da Planta 2.*

Figura 2.49 *4 iterações da Planta 2.*

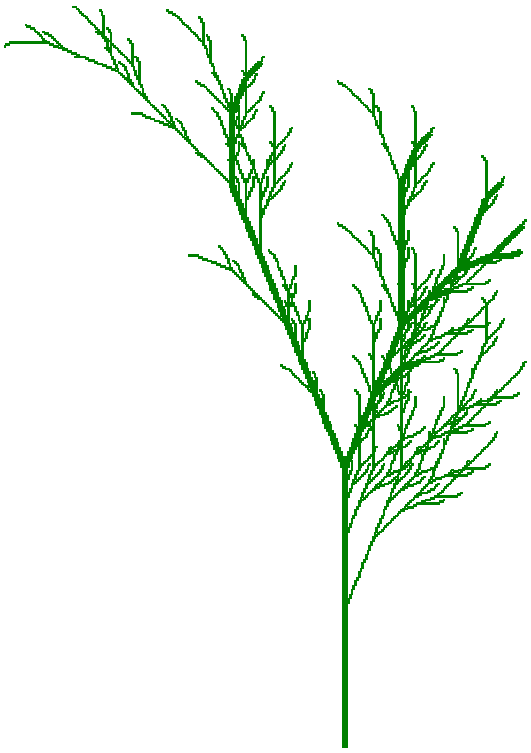

Figura 2.50 *5 iterações da Planta 2.*

2.7.2.3 Planta 3

Axioma: X

δ: 25,7°

Regras:

X -> F[+X][-X]FX

```
F -> FF
```

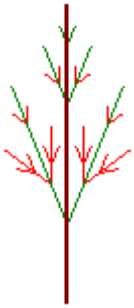

Figura 2.51 *4 iterações da Planta 3.*

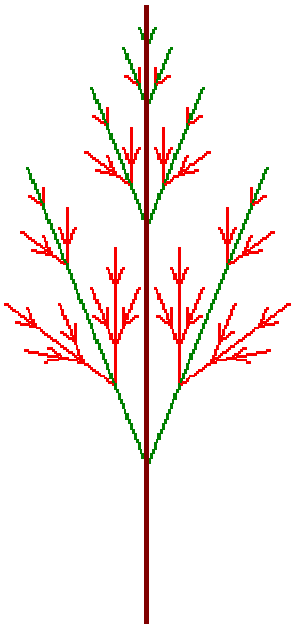

Figura 2.52 *5 iterações da Planta 3.*

2.7.2.4 Planta 4

```
Axioma: F1F1F1
δ: 22,5º
Regras:
0<0>0 -> 0
0<0>1 -> 1[-F1F01]
```

```
0<1>0 -> 1
0<1>1 -> 1
1<0>0 -> 0
1<0>1 -> 1F1
1<1>0 -> 1
1<1>1 -> 0
+ -> -
- -> +
Ignore: +-F
```

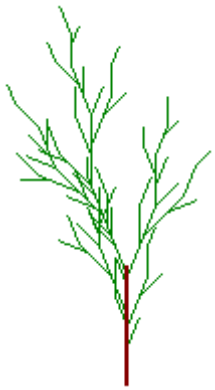

Figura 2.53 *17 iterações da Planta 4.*

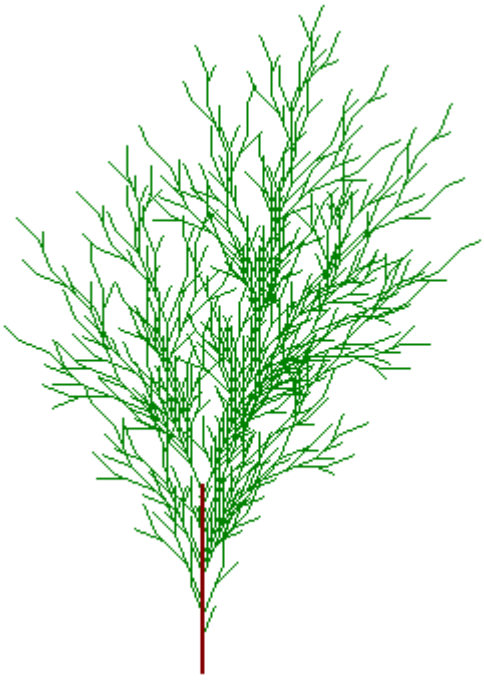

Figura 2.54 *24 iterações da Planta 4.*

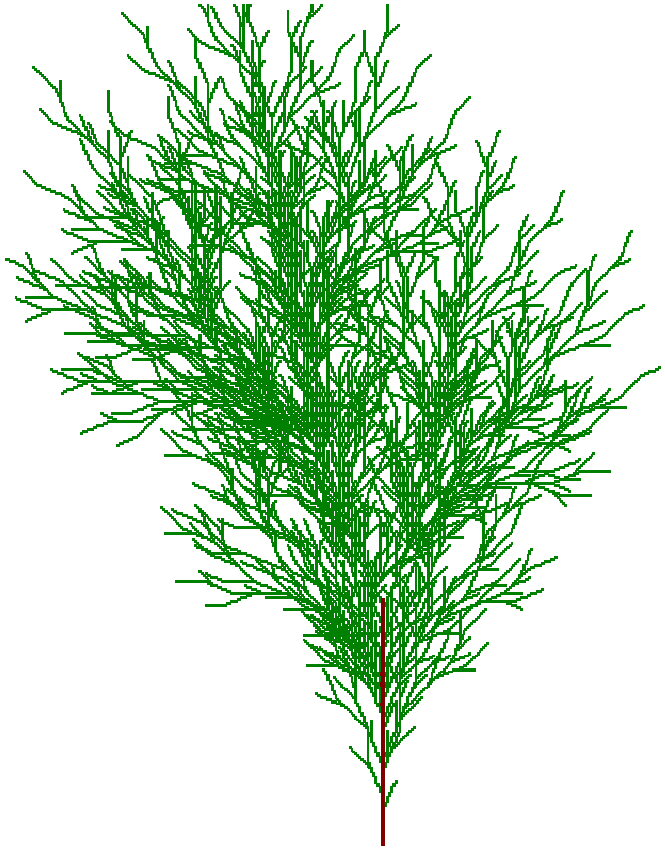

Figura 2.55 *27 iterações da Planta 4.*

2.7.2.5 Planta 5

Axioma: F1F1F1

δ: 22,5º

Regras:

0<0>0 -> 0

0<0>1 -> 1[+F1F1]

0<1>0 -> 1

0<1>1 -> 1

1<0>0 -> 0

1<0>1 -> 1F1

1<1>0 -> 0

1<1>1 -> 0

\+ -> -

\- -> +

```
Ignore: +-F
```

Figura 2.56 *15 iterações da Planta 5.*

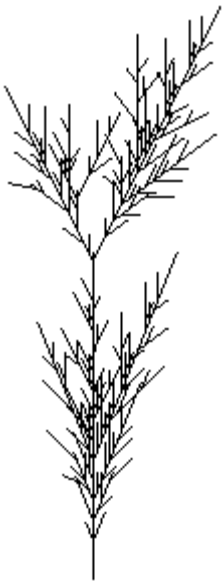

Figura 2.57 *25 iterações da Planta 5.*

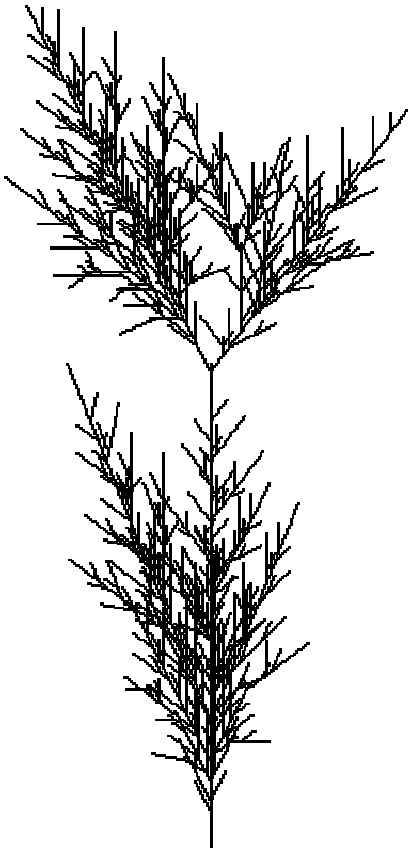

Figura 2.58 *30 iterações da Planta 5.*

2.7.2.6 Planta 6

Axioma: F1F1F1

δ: 25,75º

Regras:

0<0>0 -> 0

0<0>1 -> 1

0<1>0 -> 0

0<1>1 -> 1[+F1F1]

1<0>0 -> 0

1<0>1 -> 1F1

1<1>0 -> 0

1<1>1 -> 0

\- -> +

\+ -> -

Ignore: +-F

Figura 2.59 *22 iterações da Planta 6.*

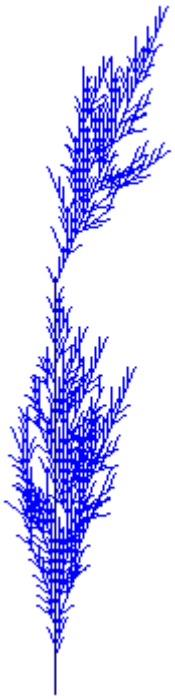

Figura 2.60 *28 iterações da Planta 6.*

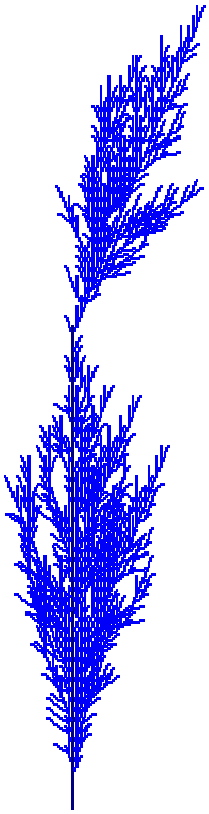

Figura 2.61 *30 iterações da Planta 6.*

3 IMPLEMENTAÇÃO DE INTERFACE

3.1 INTRODUÇÃO

No início da era da computação eram poucos os sistemas interativos, predominando os sistemas *batch*, e os maiores esforços de projeto e desenvolvimento de *software* estavam voltados para aspectos internos dos mesmos, isto é, algoritmos rápidos, estruturas de dados adequadas, utilização eficiente dos recursos de hardware.

Além disso, assim como ocorreu com outras inovações tecnológicas no seu início (como automóveis e máquinas fotográficas), os computadores somente estavam disponíveis para aqueles que se dedicassem ao domínio da nova tecnologia, ou seja, os usuários típicos dos sistemas computacionais eram especialistas em computação.

Com o surgimento do computador pessoal e sua conseqüente popularização, esta comunidade de usuários cresceu enormemente, abrangendo um novo perfil de usuário, com pouco ou nenhum conhecimento técnico.

Este novo usuário é hoje maioria absoluta nesta comunidade, e por não ser um especialista em computação exige que os esforços de projeto e desenvolvimento de *software* se voltem para os aspectos externos do mesmo, que se voltem para a interface homem-computador (IHC) ou interface com o usuário (IU).

A interface homem-computador é a porta de entrada para uma aplicação de *software* interativa (1) e abrange tudo que diz respeito à interação do ser humano com o computador, incluindo: estilos de interação (menus, formulários, manipulação direta), apresentação da informação (gráficos, cores, texto, áudio, vídeo) e

dispositivos de E/S (teclados, *displays*). A interação usuário-computador passou então a ter uma importância fundamental para o sucesso dos sistemas computacionais, sendo atualmente uma área de pesquisa em processo de larga expansão e que por envolver tanto aspectos tecnológicos quanto humanos, possui uma característica interdisciplinar, com contribuições de áreas como: Ciência da Computação, Psicologia, Desenho Gráfico, Literatura Técnica, Ergonomia, Antropologia e Sociologia.

3.2. METAS DA ENGENHARIA DE *SOFTWARE*

É importante ressaltar que os projetistas devem pensar em ir além da vaga noção de *software* "amigável", se comprometendo antes de mais nada em servir aos seus usuários e isto implica em uma identificação precisa do perfil da sua comunidade de usuários e das tarefas que devem ser desempenhadas.

Sistemas eficazes geram sentimentos positivos de sucesso, competência, domínio e clareza na comunidade de usuários. Quando um sistema interativo é bem projetado, a interface quase desaparece, permitindo aos usuários que se concentrem em suas tarefas. Isto se consegue através de planejamento, percepção das necessidades dos usuários e testes cuidadosos.

Para tanto, alguns aspectos da engenharia de *software* que vão além da interface com o usuário, devem ser observados:

- Análise da tarefa para determinar a funcionalidade apropriada. Este passo é essencial, pois sistemas com funcionalidade inadequada, não importando quão bem projetada foi a interface com o usuário, frustram o usuário e são quase sempre rejeitados ou subtilizados.

- Confiabilidade, disponibilidade, segurança e integridade de dados. Os comandos devem funcionar como especificados, os dados apresentados devem refletir exatamente o que está armazenado no banco de dados. Em geral, a confiança dos usuários nos sistemas é frágil; uma experiência com perda de dados ou com resultados inesperados poderá diminuir por muito tempo a disposição de uma pessoa em usar um sistema. A arquitetura de *software*, o *hardware* e o suporte

de rede devem garantir alta disponibilidade. Se o sistema não está disponível ou gera erros, não importa quão bem projetada foi a interface com o usuário. Os projetistas devem ainda garantir privacidade, segurança e integridade de dados, protegendo o sistema contra acesso desautorizado, destruição inadvertida de dados ou adulterações mal intencionadas.

- Padronização: Usar, se possível, padrões da indústria de *software*.
- Integração: O produto deve ser integrável a diferentes ferramentas de *software* e pacotes.
- Consistência:
    - Usar seqüências de ações, termos, unidades, cores, *layouts*, tipografia comuns dentro da aplicação de *software*.
    - Manter a compatibilidade com sistemas relacionados, baseados ou não em computador.
    - Manter a compatibilidade entre diferentes versões do produto.
- Portabilidade: Permitir ao usuário a conversão de dados entre múltiplas plataformas.

3.3 FATORES HUMANOS E DIVERSIDADE HUMANA

Preenchidos os requisitos da engenharia de *software* definidos no item anterior, os projetistas podem focar sua atenção no processo de projeto e teste. As diferentes alternativas de projeto devem ser avaliadas em função da comunidade de usuários e de um conjunto de tarefas selecionadas para avaliação.

3.3.1 FATORES HUMANOS

Sendo a interface com o usuário o mecanismo por meio do qual um diálogo entre o programa e o ser humano é estabelecido, para cada usuário e cada tarefa, o projetista deve se guiar por fatores humanos mensuráveis. Shneiderman (2) considera cinco fatores humanos centrais para a avaliação da melhor alternativa de projeto da IHC:

- Tempo de aprendizado: Tempo necessário para que membros típicos da comunidade de usuários aprendam a executar as tarefas.
- Performance: Tempo necessário para executar as tarefas.
- Taxa de erros do usuário: Quantidade e tipo de erros cometidos pelos usuários ao executar as tarefas.
- Retenção através do tempo: Determina a capacidade dos usuários em manter o conhecimento adquirido através do tempo. Está intimamente associada ao tempo de aprendizado, devendo-se levar em consideração a freqüência de uso.
- Satisfação subjetiva: Mede quanto os usuários gostam de usar diversos aspectos do sistema. Medido através de entrevistas, formulários e escalas de satisfação. Todo projetista gostaria de ter o melhor resultado nas diversas categorias, mas sempre existirão *tradeoffs* e estes devem ser considerados pelos gerentes de projeto e pelos projetistas, que farão suas escolhas de acordo com a categoria e os objetivos do sistema.

3.3.2. DIVERSIDADE HUMANA

A grande diversidade humana quanto a habilidades, conhecimento, motivações, personalidades e estilos de trabalho constitui-se em um desafio ao projetista de interfaces com o usuário. A IHC ideal seria projetada para acomodar estas diferenças ou, alternativamente, seria projetada para acomodar um perfil considerado típico entre uma comunidade de usuários.

3.4 DIRETRIZES PARA O PROJETO DE INTERFACE COM O USUÁRIO

Diversas fontes da literatura apresentam conjuntos de diretrizes de projeto de IHC que resultarão em uma interface “amigável” e eficiente.

Shneiderman (2) identifica um conjunto de princípios gerais e de diretrizes de ordem prática que são apresentados nos próximos dois tópicos.

3.4.1 PRINCÍPIOS

3.4.1.1 Princípio 1: Reconhecer a diversidade

Os projetistas de interface com o usuário devem reconhecer a diversidade de estilos de interação para responder a diversidade de perfis de usuários e de tarefas.

3.4.1.1.1 *Perfil dos Usuários*

Todos os projetos devem começar com a identificação dos usuários potenciais, incluindo perfis de sua idade, sexo, habilidades físicas, nível educacional e cultural, etnia, treinamento, motivação, metas e personalidade.

- Usuários principiantes: Usuários com pouco conhecimento da tarefa em si ou usuários que conhecem a tarefa, mas não conhecem a interface. O vocabulário de termos e o número de ações devem ser restrito, de modo que os usuários possam realizar tarefas simples, reduzindo-se a sua ansiedade. Atenção especial deve ser dada a mensagens, manuais e *helps*.
- Usuários instruídos intermitentes: Conhecem bem a tarefa e a interface, mas têm dificuldade em reter alguns aspectos em função da baixa freqüência de uso. A carga de memorização deve ser diminuída através de uma estrutura ordenada de menus, de terminologia consistente, de facilidades de exploração para relembrar seqüências de operação e de *helps on-line*, além de manuais de referência.
- Usuários especialistas freqüentes: Usuários altamente familiarizados com os conceitos da tarefa e da interface que visam a rápida execução do seu trabalho. Demandam respostas rápidas e pouco *feedback*, exigindo ainda “poderes” especiais, como atalhos e formas abreviadas de interação.

3.4.1.1.*2 Perfil das Tarefas*

Após a cuidadosa identificação do perfil do usuário, os projetistas devem identificar as tarefas. As ações de tarefas de alto nível devem ser decompostas em ações de tarefas de diferentes níveis médios e estas, por sua vez, podem ser refinadas em ações atômicas que serão executadas pelo usuário com um comando simples ou pela seleção de um menu.

Assim como a decomposição das tarefas, também é importante a freqüência relativa de execução das mesmas. Esta freqüência determinará, por exemplo, a estrutura da árvore de menus. Tarefas freqüentemente executadas devem ter acesso simples e rápido, mesmo que isto custe um prejuízo no acesso de tarefas infreqüentes.

3.4.1.1.3 *Estilos de Interação*

Quando a análise da tarefa estiver cumprida, o projetista pode escolher entre os tipos primários de estilos de interação:

- Manipulação direta: Quando o projetista pode criar uma representação visual do ambiente de ação do usuário, se utilizando de uma metáfora, as tarefas do usuário ficam simplificadas na medida em que este pode manipular diretamente objetos familiares. Manipulação direta é atraente para principiantes, é fácil de lembrar para os usuários intermitentes e, com projeto adequado, pode ser rápida para os usuários especialistas.
- Seleção por menus: Em sistemas de seleção por menus, usuários lêem uma lista de itens, selecionam o mais apropriado a sua tarefa e observam o efeito. O maior benefício permitido por estes sistemas é a estruturação clara das tarefas, favorecendo a tomada de decisão. Este estilo também é apropriado para usuários principiantes e intermitentes, podendo inclusive ser interessante para usuários especialistas, desde que os *displays* e mecanismos de seleção sejam rápidos.
- Preenchimento de formulários: É o estilo de interação mais apropriado para a entrada de dados. Usuários devem entender os nomes dos campos, conhecerem os valores permitidos e o método de entrada de dados. Mais apropriado para usuários instruídos intermitentes ou freqüentes.
- Linguagem de comandos: Para usuários freqüentes, linguagens de comando permitem uma forte sensação de controle e iniciativa, porém as taxas de erro são normalmente altas, por isso é necessário treinamento e a retenção pode ser baixa. São o domínio de usuários especialistas freqüentes, que têm grande satisfação em dominar um conjunto complexo de semântica e sintaxe.

- Linguagem natural: A esperança de que computadores respondam apropriadamente a sentenças arbitrárias em linguagem natural reúne muitos cientistas e desenvolvedores de sistemas, apesar dos resultados limitados até o momento. De fato, onde os usuários são especialistas em uma determinada tarefa cujo escopo seja limitado, há espaço para interfaces com linguagem natural.

A associação de diversos estilos de interação pode ser o mais apropriado quando também são diversos os usuários e as tarefas.

3.4.1.2 Princípio 2: Usar as oito regras para projeto de interfaces

- Lutar pela consistência;
- Permitir que usuários freqüentes tenham acesso a atalhos;
- Oferecer informação de *feedback*;
- Projetar diálogos modulares;
- Oferecer prevenção e tratamento de erro;
- Permitir fácil reversão de ações;
- Suportar a sensação de controle;
- Reduzir a carga de memorização de curto-prazo.

3.4.1.3 Princípio 3: Prevenir erros

Além da melhoria das mensagens de erro, deve-se procurar evitá-los através da criação de seqüências completas de ações que reduzam o número de interações do usuário e através da implementação de ações de correção.

3.4.2 DIRETRIZES

3.4.2.1 Diretrizes para apresentação de dados

Vejamos cinco metas de alto nível para a apresentação de dados que permanecem vitais:

- Consistência na apresentação dos dados;
- Assimilação eficiente da informação pelo usuário;
- Carga mínima de memorização pelo usuário;
- Compatibilidade da apresentação de dados com a entrada de dados;
- Flexibilidade para que o usuário controle a apresentação de dados.

3.4.2.2 Diretrizes para entrada de dados

Vejamos cinco metas de alto nível para a entrada de dados:

- Consistência;
- Ações mínimas de entrada (Menos ações de entrada, maior produtividade do usuário);
- Carga mínima de memorização pelo usuário;
- Compatibilidade da entrada de dados com a apresentação de dados;
- Flexibilidade para que o usuário controle a entrada de dados.

3.5 ESTILOS DE INTERAÇÃO

A evolução dos estilos de interação tem ocorrido na medida em que evoluem o *hardware* e as técnicas de IHC. Na medida em que o *hardware* e, conseqüentemente, os dispositivos de interação (dispositivos de *E/S* como vídeos, teclados, *mouses*, etc.) se tornam mais sofisticados, crescem as opções de estilos de interação.

Nesta seção, alguns estilos são reapresentados em detalhes, com especial atenção aos estilos mais utilizados em interfaces gráficas: menus, formulários e manipulação direta.

3.5.1 MENUS, FORMULÁRIOS E CAIXAS DE DIÁLOGO

Os menus surgiram como uma evolução da linguagem de comando, inicialmente como menus simples numerados usando toda a tela e atualmente como modernos menus *pull-down*, *popup*, *check boxes* e botões de rádio em caixas de diálogo ou links em páginas da *World Wide Web*, todos selecionáveis com cliques de *mouse*. Quando os itens do menu são escritos com terminologia familiar e são organizados em estrutura e seqüência convenientes, os usuários podem selecionar um item facilmente.

Menus são efetivos pois oferecem as deixas que induzem o usuário ao reconhecimento de sua tarefa, ao invés de forçarem o usuário a recuperar sintaxes de comandos de sua memória. A seleção por menus é especialmente efetiva quando usuários têm pouco treinamento, usam o sistema intermitentemente, não estão familiarizados com a terminologia ou necessitam de ajuda para a estruturação de seu processo de tomada de decisão.

3.5.1.1 Ordem de Apresentação de Itens

Uma vez que os itens de um menu tenham sido escolhidos, o projetista deve tratar da escolha da ordem de apresentação dos mesmos. Se os itens têm uma ordem natural esta decisão será trivial. As bases para a ordenação dos itens incluem:

- Tempo: Ordenação cronológica;
- Ordenação numérica: Ordem ascendente ou descendente;
- Propriedades Físicas: Comprimento, área, volume, temperatura, peso, velocidade, crescentes ou decrescentes.

Em muitos casos não há uma ordenação natural (relacionada a tarefa) e o projetista deve escolher entre possibilidades como estas:

- Ordenação alfabética dos termos;
- Agrupamento de itens relacionados ou agrupamento funcional (com linhas em branco ou outra marcação entre os grupos);
- Itens mais freqüentemente usados primeiro;

- Itens mais importantes primeiro (a importância pode ser difícil de determinar e pode variar entre os usuários).

3.5.1.2 *Layout* de Menus

Tem sido feita pouca pesquisa experimental sobre *layouts* de menus. Abaixo verificamos um guia com alguns itens subjetivos, que ainda necessitam de validação empírica (2).

- Use a semântica das tarefas para organizar os menus (menu simples, seqüência linear, estrutura em árvore, redes acíclicas ou cíclicas);
- Prefira menus largos e rasos a menus estreitos e profundos;
- Agrupe os itens de forma significativa: respeitando sua semântica;
- Ordene os itens também de forma significativa;
- Itens que correspondam a comandos potencialmente perigosos (deletar, formatar) devem ser colocados afastados de itens inofensivos usados com muita freqüência e devem exigir confirmação do usuário;
- Permita a utilização de atalhos (usuários experientes);
- Permita o salto para o menu principal e para o menu anterior;
- Use os nomes dos itens como títulos dos submenus correspondentes;
- Use gramática, *layout* e terminologia consistentes;
- Use itens curtos, começando pela palavra-chave;
- Identifique a posição do menu na árvore através de números, títulos, menus em cascata ou mapas de menus;
- Considere o uso de *help on-line*, de novos mecanismos de seleção e de restrições como tempo de resposta, taxa de *display* e tamanho de tela.

3.5.1.3 Formulários

Seleção por menus é eficiente para a escolha de um item de uma lista, mas se tarefas de entrada de dados como nomes de pessoas ou valores numéricos são necessárias, a entrada via teclado torna-se mais atraente. Quando muitos campos

de dados são necessários, o estilo de interação apropriado é o preenchimento de formulários. Os formulários já eram uma importante estratégia no tempo dos displays em modo texto de *80X24* e prosperaram no mundo das interfaces gráficas, assim como na *World Wide Web*.

O uso de formulários é interessante porque toda a informação está visível, dando aos usuários uma sensação de estar com o controle do diálogo. Poucas instruções são necessárias, já que a tela assemelha-se a formulários de papel já familiares aos usuários. Basta apenas que estes estejam familiarizados com o uso do teclado.

3.5.2 MANIPULAÇÃO DIRETA

As interfaces de manipulação direta são as de popularização mais recente, tendo isto ocorrido após a larga adoção das interfaces gráficas. São identificadas pela representação visual do ambiente de ação do usuário, através da utilização de uma metáfora.

Suas idéias centrais são a visibilidade dos objetos e tarefas de interesse; ações incrementais, rápidas e reversíveis; e a substituição da complexa sintaxe da linguagem de comando pela manipulação direta do objeto de interesse.

3.5.2.1 Pensamento Visual e Ícones

O computador fornece um extraordinário ambiente visual para revelar estruturas, mostrar relacionamentos e permitir a interatividade que atraem os usuários que têm personalidades artísticas, holísticas, intuitivas, em suma, que são orientados pelo lado direito do cérebro. A crescente natureza visual das interfaces dos computadores pode desafiar ou mesmo intimidar os programadores compulsivos, orientados a texto, lineares, racionais, lógicos e, portanto, orientados pelo lado esquerdo do cérebro que formavam o primeiro grupo de usuários freqüentes de computadores.

Mas o que verdadeiramente deve importar para o projetista de interfaces é que há evidências que as pessoas possuem diferentes estilos cognitivos e está claro que as preferências individuais podem variar. Assim como existem diferentes

sabores de sorvete ou modelos de carro, então devem existir diferentes estilos de interação. Pode ser que estas preferências variem entre usuários e entre tarefas, portanto, o objetivo do projetista deve ser, de acordo com cada comunidade de usuários, fornecer o melhor de cada estilo de interação e até o modo de trocar de estilo quando desejado.

Este conflito entre texto e gráficos torna-se mais forte quando se trata de ícones. Há situações em que os ícones não são aplicáveis ou em que o uso de texto é mais apropriado por ser mais compreensível (por exemplo, em situações de emergência ou perigo). O mais provável é que a melhor estratégia seja usar um pouco de cada (como na placa de PARE). A decisão de usar ícones ou texto depende não só dos usuários e das tarefas, mas também da qualidade dos ícones e das palavras usadas. Para uma escolha adequada de utilização dos ícones devem ser consideradas:

- Represente o objeto ou ação de uma maneira familiar e reconhecível;
- Limite o número de diferentes ícones;
- Faça com que o ícone se destaque do fundo da tela;
- Considere o uso de ícones tridimensionais; eles são atraentes, mas podem causar distração;
- Certifique-se que quando um ícone estiver selecionado, isto fique evidente;
- Faça com que cada ícone seja distinguível de qualquer outro;
- Garanta a harmonia de cada ícone fazendo-o membro de uma família de ícones;
- Projete a animação a ser usada quando houver movimento: quando um ícone for arrastado, o usuário pode mover totalmente o ícone, somente um quadro, uma versão transparente, cinza ou mesmo uma caixa preta;
- Adicione informações detalhadas, como sombra para indicar o tamanho de um arquivo (quanto maior a sombra, maior o arquivo), espessura para mostrar o tamanho de uma pasta de arquivos (quanto mais espessa, maior o número de arquivos na pasta), cor para mostrar a idade de um documento (os mais velhos ficam amarelados ou acizentados), ou animação para mostrar o quanto um documento foi impresso (o documento é absorvido progressivamente pelo ícone da impressora);

- Explore o uso de combinações de ícones para criar novos objetos ou ações. Por exemplo, arrastar o ícone de um documento para o ícone de uma pasta, da lixeira ou da impressora é muito útil.

3.6 APRESENTAÇÃO DE INFORMAÇÃO

Esta seção trata da apresentação da informação para o usuário, englobando três aspectos: apresentação de mensagens, *layout* de tela e uso de cores.

3.6.1 MENSAGENS

As mensagens de um sistema, sejam elas simples *prompts*, mensagens explicativas, mensagens de erro ou de advertência, têm uma grande influência no grau de aceitação dos sistemas pelos usuários. Principalmente naqueles sistemas projetados para usuários principiantes ou não-especialistas.

Para garantir mensagens claras e não-distrativas, algumas orientações devem ser seguidas:

- Esforce-se pela clareza: Explicações complexas devem ser evitadas. As mensagens devem ser escritas para o entendimento de um público-alvo, que no caso é o usuário do sistema;
- Evite pronomes: O uso de pronomes pessoais como eu ou você caracterizam o modelo conversacional, onde há uma simulação de um diálogo humano pelo computador. Apesar deste tipo de diálogo ser atraente para alguns usuários e ser uma abordagem utilizada no projeto de agentes inteligentes de interface, muitos usuários podem se sentir desconfortáveis com esta abordagem e eventualmente superestimarem a capacidade de seu sistema, diminuindo a sua responsabilidade;
- Mantenha-se breve: Mantenha as mensagens curtas. Se isto não for possível, mantenha a legibilidade, limitando a largura do texto e utilizando maior espaçamento entre as linhas;

- Mantenha a consistência da terminologia: Use sempre os mesmos verbos, como “pressione” ou “tecle”. Evite perguntas como “abortar?” ou “terminar?”, simplesmente. Diferencie o que é cancelável do que é interrompível;
- Seja cuidadoso com a gramática e o vocabulário: Verifique a ortografia ou contrate alguém para fazê-lo. Os erros gramaticais desvalorizam a aplicação e os próprios projetistas junto aos usuários. Evite jargões e gírias;
- Evite perguntas ambíguas, por exemplo: Isto irá sobrescrever o arquivo. Cancela?

3.6.1.2 Mensagens de Erro

As mensagens de erro e de advertência são as que causam maior impacto no usuário. Mensagens de erro com um tom imperativo, condenando o usuário podem aumentar a sua ansiedade, tornando mais difícil a correção do problema e aumentando o risco de novos erros. A especificidade da mensagem, o seu tom positivo e o *layout* apropriado são três critérios que devem ser observados (2).

- Especificidade: Mensagens de erro de conteúdo muito genérico podem confundir o usuário e serem de pouca utilidade, principalmente quando se tratar de usuários principiantes. Portanto, seja o mais explícito possível sobre os problemas. Por exemplo: Ao invés de “Espaço Insuficiente em disco!”, escreva “Espaço Insuficiente em disco! São necessários 700 KB e existem 356 KB.” Ao invés de “Data Inválida!”, seria mais interessante “Dias devem estar entre 1 e 31!”. E ao invés de “Erro de Sintaxe!”, poderia se usar “Parênteses aberto e não fechado!”;
- Tom positivo: Ao invés de simplesmente condenar o usuário pelo seu erro, as mensagens devem, quando possível, indicar a solução a ser adotada. Por exemplo: “Espaço Insuficiente em disco! São necessários 700 KB e existem 356 KB. Delete alguns arquivos desnecessários e repita a instalação”. O tom positivo da mensagem implica em não se usar, ou usar com pouca freqüência. Mensagens como ERRO FATAL!, EXECUÇÃO ABORTADA! ou ERRO CATASTRÓFICO! são muito hostis e podem traumatizar usuários principiantes;

- *Layout* Apropriado: Mensagens em letras maiúsculas devem ser reservadas para advertências ou erros graves. Números de erros são inúteis para os usuários e devem ser evitados. Se forem realmente necessários, devem ser colocados em uma posição discreta entre parênteses. O posicionamento das mensagens de erro pode ser próximo de onde o problema ocorreu, em uma posição constante no rodapé da tela ou em uma caixa de diálogo *pop-up* no centro da tela. Os sinais sonoros são úteis para chamar a atenção do usuário, mas podem causar embaraço, devendo ser usados com critério.

3.6.2 *LAYOUT* DE TELA

Nos sistemas interativos, as telas apresentadas aos usuários são outro componente chave para o sucesso dos mesmos. Telas muito densas ou desorganizadas e formatos inconsistentes para os dados podem incomodar e prejudicar a produtividade do usuário.

Esta seção apresenta alguns princípios gerais para o *layout* de telas, outros específicos para o *layout* dos campos e alguns resultados de testes empíricos com *layouts* de tela (2).

3.6.2.1 Princípios Gerais

- Comece todas as telas com um título que descreva sucintamente o conteúdo ou o propósito da tela. Deixe ao menos uma linha em branco entre o título e o resto da tela;
- Ao usar a codificação por tamanho de fonte, mantenha os símbolos maiores no mínimo 1,5 vezes a altura dos símbolos imediatamente menores;
- Considere o uso da codificação por cores nas aplicações em que o usuário deve distinguir rapidamente diferentes categorias de dados, particularmente quando os itens estão dispersos na tela;
- Quando estiver usando caracteres piscando (*blinking*), faça com que a freqüência com que o caractere pisca seja de 2 a 5 Hz, mantendo-o aceso no mínimo 50% do tempo;

- Para tabelas que excedam a capacidade da tela, garanta que os usuários sempre poderão ver os nomes das colunas e das linhas.

3.6.2.2 *Layout* de Campos

- Apresente os dados para os usuários no seu formato padrão de utilização (Números de identificação, datas, etc.);
- Mantenha o formato de apresentação dos dados consistente de uma tela para outra;
- Adote um princípio de ordenação das listas, caso este não exista, adote a ordenação alfabética;
- Mantenha os *labels* de campos próximos o suficiente dos mesmos, evidenciando a associação, mas mantendo pelo menos um espaço em branco entre eles;
- Faça a distinção entre os *labels* e os campos através do uso de diferentes tamanhos, tipos de letra ou formas de destaque;
- Alinhe as colunas de dados alfabéticos à esquerda, permitindo maior rapidez de pesquisa;
- Nomeie cada página em telas de múltiplas páginas, de forma que a sua relação fique evidenciada;
- A forma estruturada da informação na tela é superior a forma narrativa (seqüencial);
- A melhoria dos *labels* de campo, o agrupamento de informações relacionadas, o uso de indentação e sublinhados apropriados, o alinhamento de valores numéricos e a eliminação de caracteres estranhos melhoram a performance;
- A performance dos usuários especialistas é melhor com telas mais densas e em menor quantidade do que com telas mais esparsas e mais numerosas;
- As telas só devem ter conteúdo relevante à tarefa do usuário;
- É importante manter consistente o posicionamento de títulos, de números de página, da área de mensagens e da área de entrada de dados entre as diversas telas;
- Da mesma forma, a consistência de terminologia é benéfica;
- A seqüência de telas deve ser similar para tarefas similares.

3.6.3 CORES

As cores têm o poder de serem rapidamente percebidas e identificadas, sendo uma parte importante no projeto de interfaces gráficas. Se bem utilizadas, podem tornar a aplicação mais atraente e melhorar a performance do usuário na execução de suas tarefas.

3.6.3.1 Princípios

- Use com moderação: Limite o número e a variedade de cores (máximo de quatro em uma tela e máximo de sete em toda a aplicação). Uma boa estratégia é sempre usar letras de cor preta em fundo branco, usando itálico ou negrito para dar ênfase e reservando as cores para destaques especiais;
- Não projete a tela muito brilhante ou muito escura: Use cores brilhantes apenas em pequenas áreas.
- Combine cuidadosamente as cores de primeiro e segundo planos: As diversas cores do espectro, por possuírem diferentes comprimentos de onda, causam sensações de profundidade ligeiramente diferentes. Portanto, evite combinar cores muito afastadas no espectro, pois isto poderá causar fadiga visual no usuário em função da necessidade constante de correção do foco;
- Aproveite-se disto e use, por exemplo, azul para segundo plano e vermelho ou amarelo para primeiro plano. Não escolha pares de cores claras ou escuras. Escolha pares que sejam cromaticamente complementares (colocadas juntas formam o branco. Ex.: Amarelo e azul);
- Use cores em telas gráficas com grande densidade de informação: Como em sistemas financeiros com suas diversas tabelas e gráficos, sistemas geográficos e ferramentas CAD;
- Tenha em mente o poder dos códigos de cores: As cores podem acelerar o processo de reconhecimento das informações;
- Garanta que o código de cores seja escolhido de acordo com a tarefa do usuário;
- Garanta o aparecimento do código de cores com o mínimo esforço: O aparecimento das cores deve ser automático;

- Permita que o usuário controle o código de cores: Quando possível, torne as cores personalizáveis;
- Seja consistente com códigos de cores: Por exemplo, se usar vermelho para mensagens de erro, use em toda a aplicação;
- Esteja alerta para as expectativas dos usuários quanto ao significado das cores: em diferentes tarefas, as cores podem ter significados diferentes. Por exemplo: vermelho x azul para temperatura, vermelho x verde para sinalização e vermelho x preto para finanças. O vermelho significa quente, pare ou perigo e perdas, respectivamente;
- Considere os significados culturais das cores: *Softwares* que tenham apelo internacional devem estar preparados para as possíveis conotações das cores nas diferentes culturas;
- Use a mudança de cores para indicar mudança de status: Por exemplo, quando determinados dados assumem valores perigosos, devem mudar de cor (normalmente para vermelho). Isto é especialmente útil em telas densas;
- Use as cores no *layout* da informação apresentada: As cores podem ser usadas para agrupar itens, facilitando a sua identificação;
- Projete para telas monocromáticas primeiro: Isto é necessário para algumas aplicações, em cujo ambiente de produção os monitores coloridos não forem um padrão;
- Considere as deficiências visuais: As cores não devem ser a parte primária do projeto, devendo ser uma informação redundante. Um percentual considerável da população tem dificuldades em distinguir cores.

3.6.3.2 Benefícios do uso de cores

- As cores podem ser agradáveis e atraentes aos olhos;
- As cores podem tornar uma tela mais atraente;
- As cores facilitam a identificação rápida em telas densas;
- Os códigos de cores podem enfatizar a organização lógica da informação;
- Certas cores podem chamar a atenção para avisos;
- As cores podem provocar reações emocionais de prazer, excitação, medo ou raiva.

4 IMPLEMENTAÇÃO DO *SOFTWARE*

4.1 OBJETIVO

Após o estudo dos L-Systems na disciplina Geometria Fractal há alguns períodos, foi implementado um sistema capaz de interpretar L-Systems simples. A partir disso, surgiu a idéia de ampliar este sistema para interpretação de L-Systems mais complexos.

Mais tarde, a idéia central do projeto do sistema implementado não foi exclusivamente desenvolver um sistema que gerasse plantas utilizando o modelo dos L-Systems. O grande avanço foi a possibilidade de todos os usuários poderem operar o sistema de forma segura, buscando imagens interessantes e também um rápido aprendizado sobre a aplicação dos L-Systems para a produção das mesmas.

O sistema é dividido logicamente em duas interfaces. Na primeira interface (tela inicial) o usuário através de uma galeria de plantas pré-existentes, escolhe a que mais se se aproxima do que deseja. O sistema oferece possibilidade de pequenas alterações na planta escolhida como causar uma rotação, mudar o ângulo de inclinação de seus galhos, mudança de cores por partes e ainda aumentar seu tamanho. Em todas essas operações os conceitos de L-System são transparentes para o usuário.

A segunda interface consta de um editor de fórmulas de L-System. Usuários iniciantes que visam entender e aprender com plantas já modeladas ou usuários avançados que tem o domínio total ou parcial dos conceitos de L-System são capazes de editar regras e axiomas.

Após o preenchimento ou alteração desses dados, o usuário é capaz de transformar suas regras em desenho na mesma tela. O sistema ainda possibilita que

o usuário seja capaz de transferir essa planta para a galeria de plantas, podendo recuperá-la e alterá-la mais tarde.

4.2 TECNOLOGIA UTILIZADA E CARACTERÍSTICAS

Desde a primeira versão do sistema, onde só era capaz de interpretar os L-Systems mais simples, foi implementado usando Borland Delphi 6. A escolha por esta ferramenta se deu pelo vasto número de componentes gráficos e operacionais que ela oferece. Além disso, a linguagem Object Pascal é bastante familiar, já que desde o início da graduação se vem praticando e implementando sistemas com ela.

Como Delphi já é uma linguagem com suporte à orientação a objetos, o sistema foi totalmente implementado baseado nos conceitos de orientação a objetos.

Outro conceito adotado foi a modularização do projeto. Funcionalidades foram dividias em vários módulos e implementados separadamente para posterior ligação entre os mesmos, facilitando assim o reconhecimento de erros, futuras manutenções e alterações no código ou estrutura.

O sistema é acompanhado de um kit de instalação, facilitando a organização de arquivos e diretórios da aplicação. Um *help* contendo um passo à passo de todas as funcionalidades também está incluso.

O sistema funciona baseado em três vetores dinâmicos: um armazena as regras (vetor regras), outro armazena os ignores (vetor ignore) e o terceiro armazena os símbolos resultantes ao final de cada iteração (vetor resultante). Inicialmente o vetor resultante contém o axioma do L-System.

A cada iteração, é criado um vetor temporário semelhante ao vetor resultante. O vetor resultante é percorrido seqüencialmente e para cada símbolo é percorrido o vetor regras em busca de uma regra para substituição. Todos os símbolos gerados ou não por substituições nessa dada iteração serão escritos no vetor temporário. No final desse processo, o vetor resultante é atualizado com informações do vetor temporário.

Caso seja um L-System não determinístico, o sistema gera randomicamente um número que será usado para escolha da regra a ser aplicada.

Em L-Systems sensíveis ao contexto, o sistema é capaz de identificar regras simples e duplas, somente à direita ou a esquerda e à direita e à esquerda

respectivamente. Com essa detecção fica fácil analisar os vizinhos de cada símbolo utlizando o vetor resultante e o vetor ignore (caso haja algum elemento) para utilização ou não da regra corrente.

Após todas as substituições, o vetor resultante é percorrido seqüêncialmente para a interpretação de cada símbolo. Podemos dizer que é nessa etapa do processo que o fractal nasce na tela do sistema.

4.3 SOBRE EASY TREE

4.3.1 COMPONENTES DA TELA PRINCIPAL

Podemos dividir a tela principal da aplicação conforme a *figura 4.1*.

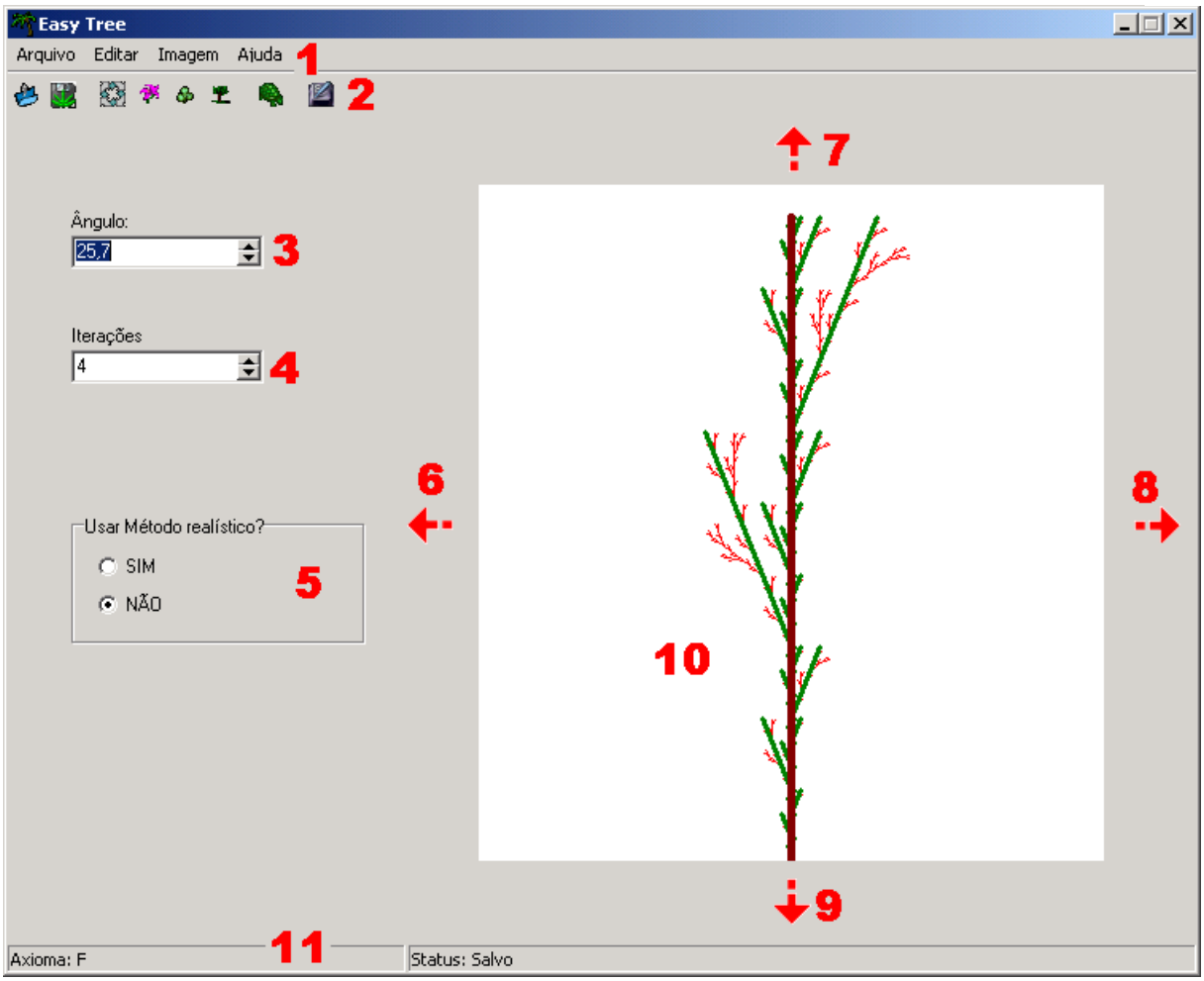

Figura 4.1 *Tela Principal do Easy Tree.*

1. *MENUS*: Através dos menus o usuário escolhe que tarefa a aplicação deve executar.
2. *BARRA DE FERRAMENTAS*: Agrupa atalhos para as principais tarefas do sistema.
3. *ÂNGULO PADRÃO*: O usuário pode alterar seu conteúdo utilizando o teclado seguido da tecla *enter* ou utilizando os botões laterais de incremento e decremento de 0,5º e com isso alterando a inclinação dos galhos.
4. *NÚMERO DE ITERAÇÕES*: O usuário pode alterar o número de execuções do L-System que é responsável pelo crescimento da planta. Podendo também utilizar o teclado seguido da tecla *enter* ou utilizando os botões laterais de incremento e decremento de 1 unidade.

5. *FERRAMENTA DE REALIADADE*: Com esta função o usuário pode fazer com que a planta gerada seja mais próxima da realidade. Cada vez que é desenhada, o sistema adiciona um ângulo randomicamente ao ângulo padrão.
6. *SETA PARA ESQUERDA*: O Usuário pode visualizar a parte lateral esquerda da planta desenhada caso esteja oculta.
7. *SETA PARA CIMA*: O Usuário pode visualizar a parte superior da planta desenhada caso esteja oculta.
8. *SETA PARA DIREITA*: O Usuário pode visualizar a parte lateral direita da planta desenhada caso esteja oculta.
9. *SETA PARA BAIXO*: O Usuário pode visualizar a parte inferior da planta desenhada caso esteja oculta.
10. *ÁREA DE DESENHO*: Através dessa área o usuário visualiza a planta desenhada.
11. *BARRA DE STATUS*: A aplicação informa o axioma da planta corrente e também exibe se a planta já foi salva.

4.3.2 COMPONENTES DA TELA EDITOR DE L-SYSTEM

Podemos dividir a tela de editor de L-System conforme a *figura 4.2*.

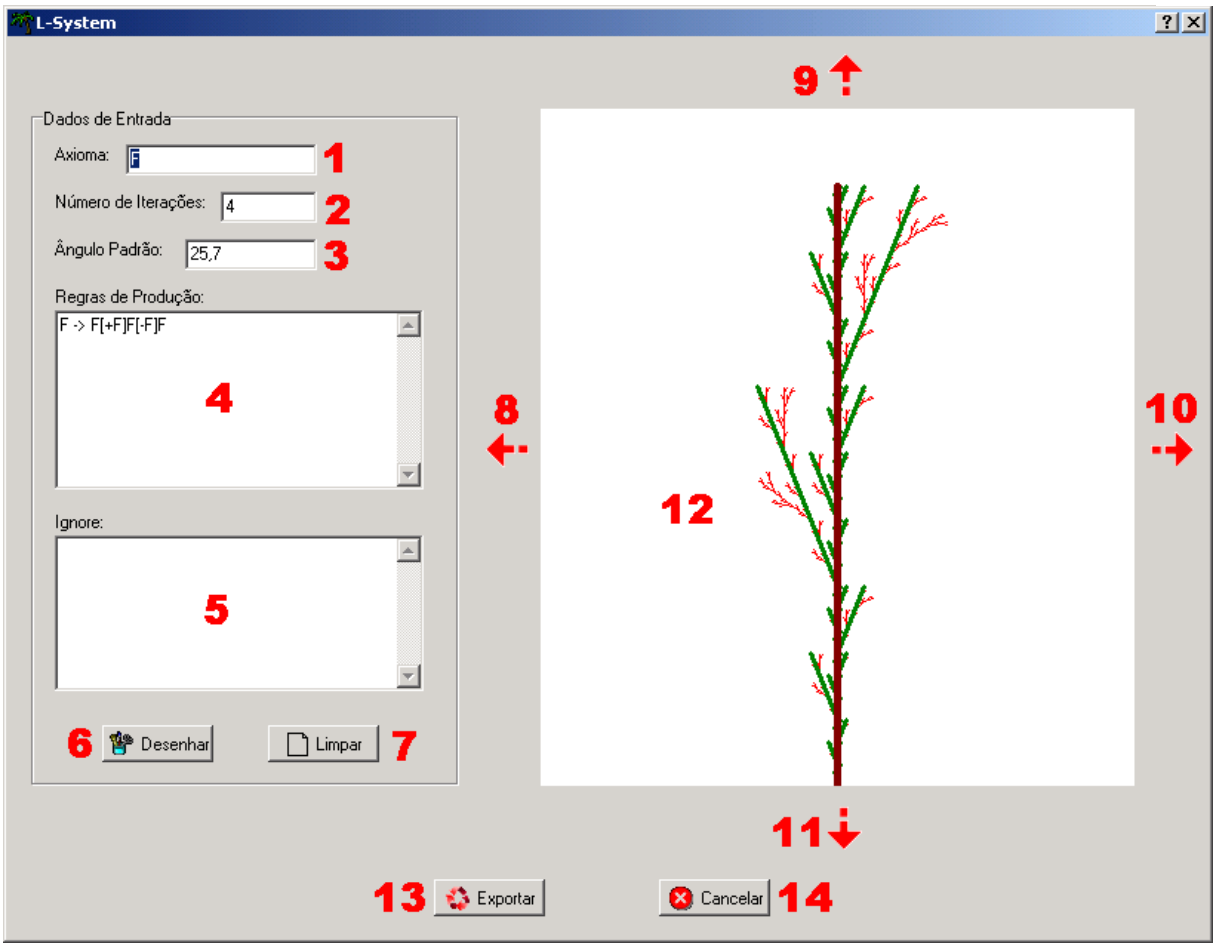

Figura 4.2 *Tela do editor de L-System.*

1. *AXIOMA*: Reservado para o axioma ou inicializador do L-System.
2. *NÚMERO DE ITERAÇÕES*: Reservado para o número de iterações do L-System que ocasionam o crescimento da planta.
3. *ÂNGULO PADRÃO*: Reservado para o ângulo de inclinação dos galhos.
4. *REGRAS*: O usuário digita todas as regras do L-System uma abaixo da outra seguindo o seguinte padrão:

Único espaço aceito é a separação do conteúdo da regra para o "->"

Regras livres de contexto:

```
F -> F[+F]F
```

Regras sensíveis ao contexto:

```
a<F -> FF
F>b -> F[-F]
a<F>b -> FF[+FF]
```

Regras utilizando probabilidade:

```
F1 -> F[+F]F
F2 -> F[-F] F
F3 -> FF
Fn -> …, onde n > 0
```

5. *IGNORE*: O usuário digita todos os símbolos a serem ignorados nas regras sensíveis ao contexto. Cada símbolo deve ocupar uma linha.
6. *BOTÃO DESENHA*: Se todas as informações foram preenchidas corretamente a aplicação gera a planta representada pelo L-System digitado.
7. *BOTÃO LIMPA*: Limpa todos os campos de dados e a figura desenhada.
8. *SETA PARA ESQUERDA*: O Usuário pode visualizar a parte lateral esquerda da planta desenhada caso esteja oculta.
9. *SETA PARA CIMA*: O Usuário pode visualizar a parte superior da planta desenhada caso esteja oculta.
10. *SETA PARA DIREITA*: O Usuário pode visualizar a parte lateral direita da planta desenhada caso esteja oculta.
11. *SETA PARA BAIXO*: O Usuário pode visualizar a parte inferior da planta desenhada caso esteja oculta.
12. *ÁREA DE DESENHO*: Através dessa área o usuário visualiza a planta desenhada.
13. *BOTÃO EXPORTAR*: Transfere a planta desenhada para a tela principal, possibilitando desta maneira ao usuário salvar posteriormente na galeria.
14. *BOTÃO CANCELAR*: Volta para a tela principal sem transferir a planta desenhada.

4.3.3 COMPONENTES DA GALERIA DE PLANTAS

Podemos dividir a tela de Galeria de Plantas conforme a *figura 4.3*.

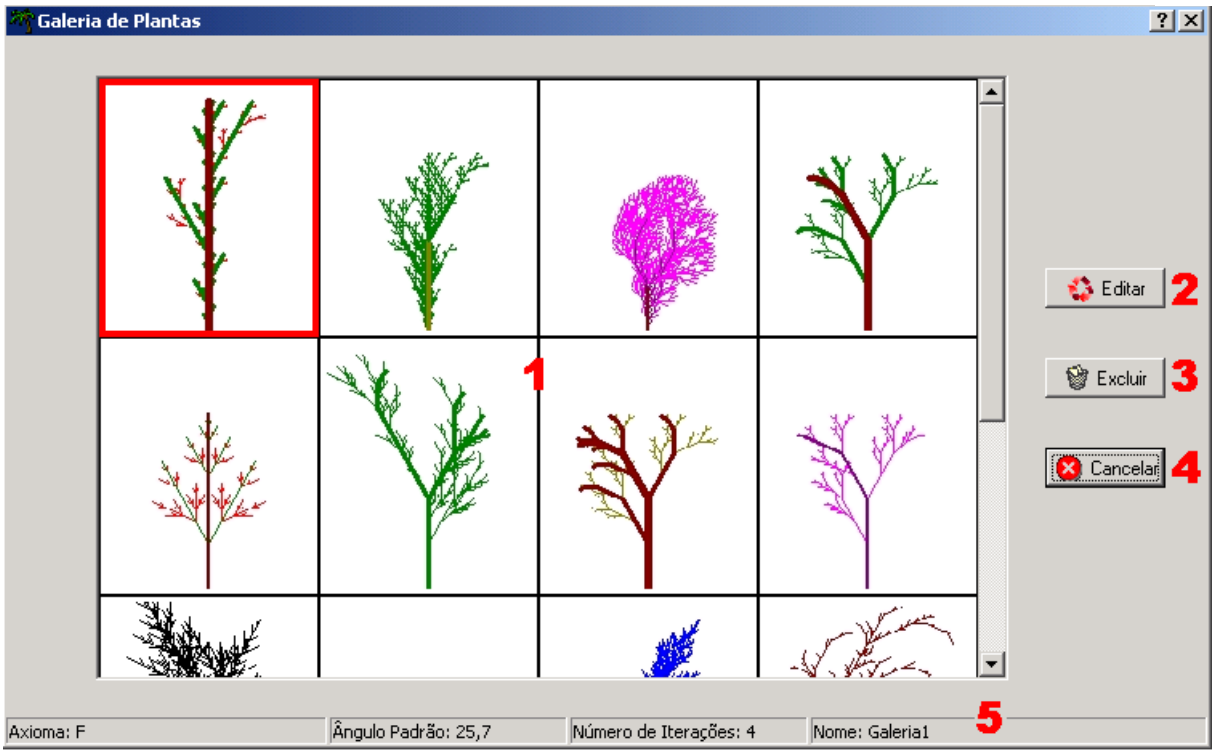

Figura 4.3 *Tela da Galeria de Plantas.*

1. *PLANTAS*: Contém uma prévia de todas as plantas salvas pelo sistema. Um clique sobre uma planta a seleciona. Um clique duplo executa a mesma tarefa do botão editar (abre a planta na a tela inicial).
2. *BOTÃO EDITAR*: Abre planta selecionada na tela principal.
3. *BOTÃO EXCLUIR*: Caso a planta não seja exemplos de demonstração do Easy Tree, o usuário pode excluir a planta selecionada.
4. *BOTÃO CANCELAR*: Fecha Galeria e volta a tela principal.
5. *BARRA DE STATUS*: A aplicação informa o axioma, ângulo padrão, número de iterações e nome da planta corrente.

4.3.4 COMPONENTES DA TELA SALVAR PLANTA

Podemos dividir a tela Salvar Planta conforme a *figura 4.4*.

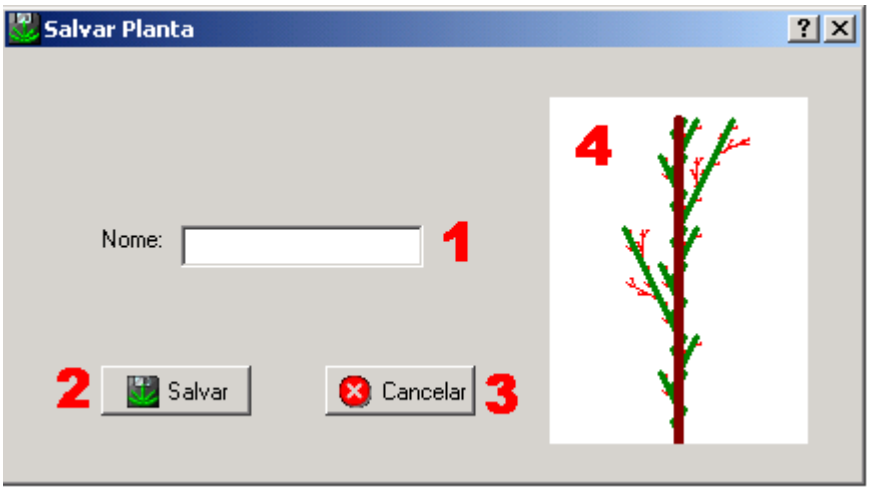

Figura 4.4 *Tela para Salvar Planta.*

1. *NOME*: Usuário digita um nome para a planta a ser armazenada na galeria.
2. *BOTÃO SALVAR*: Salva planta na galeria de Plantas.
3. *BOTÃO CANCELAR*: Fecha tela Salvar Planta e volta a tela principal.
4. *ÁREA DE DESENHO*: Através dessa área o usuário visualiza a planta a ser salva.

4.3.5 COMPONENTES DA TELA PARA GIRAR PLANTA

Podemos dividir a tela Girar Planta conforme *figura 4.5*.

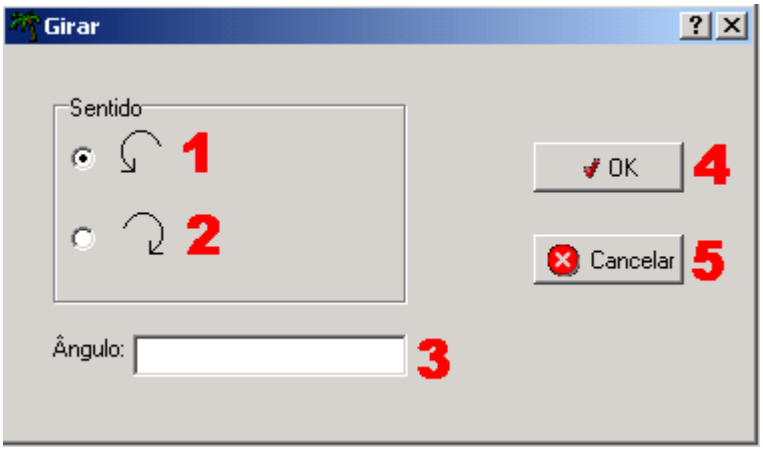

Figura 4.5 *Tela para Girar Planta.*

1. *SENTIDO ANTI-HORÁRIO*: Sistema gira planta no sentido anti-horário.
2. *SENTIDO HORÁRIO*: Sistema gira planta no sentido horário.
3. *ÂNGULO*: Ângulo em graus para a rotação.

4. *BOTÃO OK*: Sistema gira a planta de acordo com ângulo digitado no campo anterior no sentido escolhido.
5. *BOTÃO CANCELAR*: Fecha tela Girar Planta e volta a tela principal.

4.3.6 COMPONENTES DA TELA AJUSTE DE FLORES, FOLHA E GALHOS

Podemos dividir essas telas conforme a *figura 4.6*.

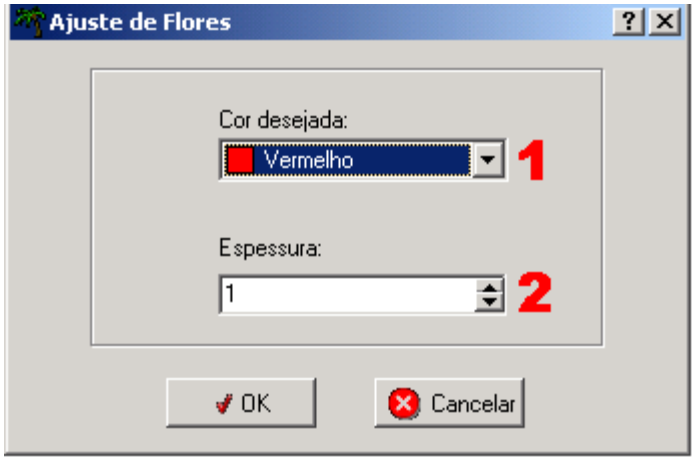

Figura 4.6 *Tela para ajuste de Flores.*

1. *COR*: Usuário altera a cor das flores, folhas e galhos respectivamente.
2. *ESPESSURA*: O usuário pode alterar a espessura das flores, folhas e galhos respectivamente. Podendo utilizar o teclado seguido da tecla *enter* ou utilizar os botões laterais de incremento e decremento de 1 unidade.

4.3.7 FUNÇÕES DE SISTEMA

1. *Abrir Galeria*: Essa opção permite ao usuário recuperar ou excluir plantas salvas anteriormente ou recuperar plantas de demonstração do Easy Tree.

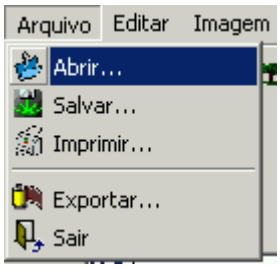

Figura 4.7 *Abrir galeria pelo menu.*

Figura 4.8 *Abrir galeria pela barra de ferramentas.*

2. *Salvar Planta*: Essa opção permite ao usuário salvar na galeria a planta corrente. Para melhor organização é possível nomear a planta a ser salva.

Figura 4.9 *Salvar Planta pelo menu.*

Figura 4.10 *Salvar Planta pela barra de ferramentas.*

3. *Imprimir Planta*: Essa opção permite ao usuário escolher uma impressora e imprimir planta corrente.

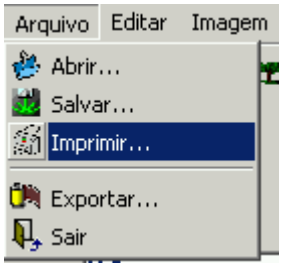

Figura 4.11 *Imprimir Planta.*

4. *Exportar Planta*: Essa opção permite ao usuário salvar em formato JPG, GIF ou BMP a planta corrente.

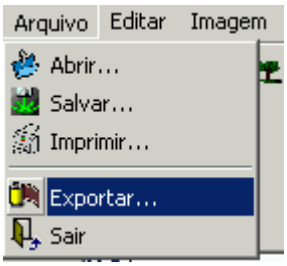

Figura 4.12 *Exportar Planta.*

5. *Sair*: Essa opção encerra a aplicação.

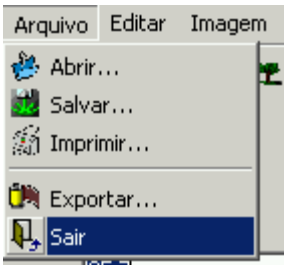

Figura 4.13 *Sair do Easy Tree.*

6. *Desfazer*: Essa opção permite ao usuário desfazer a última ação executada.

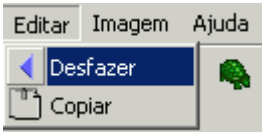

Figura 4.14 *Desfazer.*

7. *Copiar*: Essa opção permite ao usuário copiar para área de transferência a planta corrente.

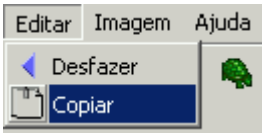

Figura 4.15 *Copiar.*

8. *Refresh*: Essa opção redesenha a planta corrente. Caso a opção de modo realístico seja usada ou hajam regras de probabilidade, cada vez será desenhada uma nova planta.

Figura 4.16 *Refresh pelo menu.*

Figura 4.17 *Refresh pela barra de ferramentas.*

9. *Girar*: Essa opção permite ao usuário girar de um determinado ângulo e sentido a planta corrente.

Figura 4.18 *Girar pelo menu.*

Figura 4.19 *Girar pela barra de ferramentas.*

10. *Ajustar Flores*: Essa opção permite ao usuário personalizar cor e espessura das flores (ramos mais finos).

Figura 4.20 *Ajustar flores pelo menu.*

Figura 4.21 *Ajustar flores pela barra de ferramentas.*

11. *Ajustar Folhas*: Essa opção permite ao usuário personalizar cor e espessura das folhas (ramos de espessura intermediária).

Figura 4.22 *Ajustar folhas pelo menu.*

Figura 4.23 *Ajustar folhas pela barra de ferramentas.*

12. *Ajustar Galhos*: Essa opção permite ao usuário personalizar cor e espessura dos galhos (ramos mais grossos).

Figura 4.24 *Ajustar galhos pelo menu.*

Figura 4.25 *Ajustar galhos pela barra de ferramentas.*

13. *L-System*: Essa opção permite ao usuário abrir o editor de L-System.

Figura 4.26 *Abrir Tela de Editor de L-System pelo menu.*

Figura 4.27 *Abrir Tela de Editor de L-System pela barra de ferramentas.*

5 CONCLUSÃO

Este trabalho tenta unir diversas áreas e conceitos da Computação. Principalmente conceitos da Engenharia de *Software* e Computação Gráfica. Focando sempre as expectativas do usuário final do sistema projetado, a Engenharia de *software* aborda tópicos que possibilitam a implementação de um único sistema capaz de atender uma vasta e heterogênea comunidade de usuários. Já a Computação gráfica, tenta de forma visual (sendo mais agradável e mais fácil de ser compreendida) mostrar soluções de problemas matemáticos complexos, além de permitir a construção das mais belas e complexas imagens com regras simples e com conceitos que podem ser aplicados em diversas áreas como História e Medicina, por exemplo.

Além de todos os benefícios trazidos pela descoberta e pesquisa na abordagem de L-System, podemos concluir que nenhuma área de pesquisa é independente; cada resultado obtido está intimamente ligado com algum outro resultado obtido anteriormente, qualquer que seja a época e área de pesquisa. Podemos exemplificar nossa conclusão com o próprio histórico de L-System. Estudos do biólogo Aristid Lindenmayer refletiram e motivaram a utilização de seus resultados em Computação Gráfica.

Podemos verificar que a abordagem de L-Systems simplifica o entendimento e a criação de uma coleção de figuras complexas. Sem estes conceitos, essas figuras poderiam ser esquecidas por sua grande dificuldade.

6 OBRAS CITADAS

1 PRESSMAN, Roger S. *Engenharia de Software*. 3 ed. São Paulo: Makron Books, 1995. 987 p.

2 SHNEIDERMAN, Ben. *Designing the user interface:* Strategies for Effective Human-Computer Interaction. Massachusetts: Reading, 1998. 256 p.

7 OBRAS CONSULTADAS

PRUSINKIEWICZ, Przemyslaw; LINDENMAYER, Aristid. *The algorithmic beaty of plants*. New York: Springer, 1990. 228 p.

PRESSMAN, Roger S. *Engenharia de Software*. 3 ed. São Paulo: Makron Books, 1995. 987 p.

SHNEIDERMAN, Ben. *Designing the user interface:* Strategies for Effective Human-Computer Interaction. Massachusetts: Reading, 1998. 256 p.

HECKEL, Paul. *Software Amigável:* Técnicas de Projeto de *Software* para uma Melhor Interface com o usuário. Rio de Janeiro: Editora Campus, 1993. 347 p.

INSTITUTO MILITAR DE ENGENHARIA. Disponível em: <http://www.des.ime.eb.br/~madeira/compgraph/lsystem/lsystem.htm>. Acesso em: 14 dezembro 2003.

UNIVERSIDADE DE LISBOA. Faculdade de ciências. Disponível em: <http://www.educ.fc.ul.pt/icm/icm2000/icm24/>. Acesso em: 21 Dezembro 2003.

ESCOLA PROF. CHRISTINO CABRAL. Disponível em: <http://www.proarte.pro.br/MATEM%C3%81TICA.htm>. Acesso em: 21 dezembro 2003.

DAVID J. WRIGHT. *Dynamical Systems and Fractals Lecture Notes*. Disponível em: <http://www.math.okstate.edu/mathdept/dynamics/lecnotes/lecnotes.html>. Acesso em: 21 dezembro 2003.

CANONGIA, Ligia. Holofractal, a arte no futuro. *O Globo*, Rio de Janeiro, 22 novembro 1988, Segundo Caderno. Disponível em: <http://www.ekac.org/canongia.html>. Acesso em: 21 dezembro 2003.

AVATAR SPACE DESIGN. Disponível em: <http://www.avatar.com.au/courses/Lsystems/L-SystemTypes.html>. Acesso em: 24 fevereiro 2004.