UNIVERSIDADE FEDERAL FLUMINENSE

Algoritmo Tartaruga Na Modelagem 3D

Aldenir de Farias Barboza
Leonardo Loback Martinez

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:

- Regina Paula Leal Toledo
- Leonardo Cruz da Costa
- Aura Conci (orientador)

Departamento de Ciência da Computação
29 de julho de 2004

Agradecimentos

A Deus, que em Sua infinita bondade, deu-nos a força, a perseverança e a luz necessárias à conclusão deste trabalho, pois, com toda a certeza, sem isso não chegaríamos até aqui.

À Professora Aura Conci, que tanto nos honrou por aceitar o convite como orientadora e pela confiança e entusiasmo dedicados a este trabalho.

Aos professores Regina Paula Leal Toledo e Leonardo Cruz da Costa por aceitarem o convite em participar da banca examinadora.

A todos os Professores do Departamento de Ciência da Computação da Universidade Federal Fluminense, pelo apoio, profissionalismo, dedicação, competência na construção de diversos profissionais.

Aos funcionários da Universidade Federal Fluminense, por toda a ajuda prestada e colaboração.

Aos grandes amigos Cláudia, Daniela, Viviane, Keity e Miriam, que tanto nos apoiaram, nos ouviram e incentivaram, estando sempre dispostas a nos ceder um ombro amigo para dividir os bons e maus momentos da vida.

Aos nossos pais por estarem em todos os momentos ao nosso lado, nos apoiando e incentivando, registro de nossa eterna e mais sincera gratidão

Enfim, a todos aqueles que, direta ou indiretamente, contribuíram para que este trabalho se concretizasse.

RESUMO

Este trabalho tem como primeiro objetivo realizar um estudo sobre conceitos da utilização do algoritmo tartaruga para modelagem em duas e três dimensões.

Com um dos maiores empregos desse algoritmo é a geração de objetos fractais e L-Systems iniciamos por introduzir os elementos da geometria fractal e desses sistemas recursivos.

Embora o algoritmo tartaruga já seja muito conhecido e utilizado no desenho em duas dimensões seu emprego para geração de figuras e objetos tridimensionais não é uma simples adição de uma coordenada a mais. Os desenhos de objetos 3D em superfícies 2D precisam ser inicialmente projetados adequadamente nesta superfície. Se depois precisarem ser observados em diversas direções devem ser rotacionadas ou transladados adequadamente. Uma alternativa a movimentação do objeto é a alteração do posicionamento do observador ou da câmera que captura a cena. Uma forma adequada de se fazer isso é utilizar a API gráfica OpenGL. Ao se mover em 3D o autômato virtual modelado pelo algoritmo tartaruga introduz dificuldades adicionais. Essas dificuldades são principalmente devidas a rotações especiais que causam mais complexidade nos referenciais de posicionamento de objetos 3D.

No mundo tridimensional o que antes era uma simples rotação da tartaruga no plano deve ser pelo menos representado por rotações em torno de eixos principais da tartaruga ou giros em mais de um plano.

O desenvolvimento feito por nós utiliza a idéia de um sistema de coordenadas esféricas.

Apresentado esta modelagem como um software desenvolvido e projetado por nós, usa esses desenvolvimentos na modelagem de curvas e plantas bi e tridimensionais.

Índice

Figuras

1 – Introdução

Uma das maiores utilizações do autômato virtual descrito no algoritmo tartaruga é a geração de objetos fractais, ou seja, elementos descritos pela Geometria Fractal. Como essa geometria ainda não é disseminada iniciaremos nosso trabalho introduzindo neste primeiro capitulo seus conceitos.

1.1 - Geometria Euclidiana

Euclides (330 a 260 a.C.) é reconhecido como o matemático mais importante da Grécia clássica e até de todos os tempos por alguns autores.

O trabalho de Euclides é tão vasto que alguns historiadores não acreditam que seja obra de um só homem. Os trabalhos matemáticos que chegaram até nós foram inicialmente traduzidos para árabe, depois para o latim, e a partir destes dois idiomas para outras línguas européias.

Embora alguns conceitos já fossem conhecidos anteriormente à sua época, o que impossibilita uma análise completa da sua originalidade, pode-se considerar o seu trabalho genial, pois reuniu tudo o que então se conhecia, sistematizou dados intuitivos e substituiu imagens concretas por noções abstratas, possibilitando raciocinar até logicamente sem qualquer apoio intuitivo.

Euclides tem uma importância excepcional na história da matemática. Suas definições, axiomas ou postulados (conceitos e proposições admitidas sem demonstração que constituem os fundamentos especificamente geométricos e fixam a existência dos entes fundamentais: ponto, reta e plano)

e os teoremas não aparecem agrupados ao acaso, mas expostos numa ordem concreta.

Durante séculos, os objetos e os conceitos da Geometria Euclidiana foram à única ferramenta para descrever o mundo em que vivemos. O homem observou a Natureza concebendo conceitos de formas, figuras planas, corpos, volumes, retas e curvas. A Lua e o Sol foram representados como discos; o raio de luz deu a idéia de linha reta; as margens de algumas folhas e o arco-íris a idéia de curva; os troncos de algumas árvores e as montanhas deram idéia de formas muito variadas. Assim se for observado por um telescópio observa-se que a superfície do Sol e da Lua são muito irregulares que um plano e os detalhes das folhas e troncos ou relevo são dependentes da lente usada para examiná-lo.

Euclides fez uma sistematização que vem sendo usada e ensinada até nossos dias, mas a geometria evoluiu e a sua geometria não é suficiente para descrever toda a complexidade de todos as formas e do conhecimento matemático atual. Existe uma infinidade de fenômenos da natureza que graças à sua irregularidade não podem ser descritos pela geometria euclidiana.

1.2 - Geometria Fractal

> *Por que a geometria é freqüentemente descrita como "fria" e "seca"? Uma razão repousa em sua inabilidade de descrever a forma de uma nuvem, uma montanha, uma linha costeira ou uma árvore. Nuvens não são esferas, montanhas não são cones, linhas costeiras não são círculos, e uma casca de árvore não é lisa, tampouco um feixe de luz viaja em linha reta. (...) Eu afirmo que muitos dos padrões da Natureza são tão irregulares e fragmentados, que, se comparados com a geometria tradicional, exibem não somente um grau mais alto, mas um nível de complexidade completamente diferente.*
>
> *(Mandelbrot, 1991, p.1)*

Nos últimos trinta anos um novo tipo de geometria permitiu ao homem compreender e reconstruir muitas das imagens complexas da Natureza, estendendo-se desde aspectos macroscópicos das nuvens até aspectos microscópicos do sistema vascular humano.

Surge assim, uma nova forma de geometria denominada de Geometria Fractal, por Benoit Mandelbrot (matemático franco-polaco, nascido na Polônia, que sistematizou esta geometria no fim da década de setenta), quebrando o determinismo matemático e completando a Geometria Euclidiana, possibilitando trabalhar com as formas complexas e repetitivas da Natureza que nos rodeiam.

1.2.1 - O que é fractal ?

Segundo a definição de fractal de Mandelbrot:

"Fractais são formas igualmente complexas no detalhe e na forma global".

A palavra fractal foi criada por Mandelbrot para descrever um objeto geométrico que nunca perde a sua estrutura, qualquer que seja à distância, escala ou resolução da visualização como se uma fração do objeto

reproduzisse o todo. Esta denominação foi usada, pois se verificou que os seus objetos de estudos possuíam uma dimensão fracionária (de Hausdorff).

As dimensões fracionárias tornaram-se uma forma de quantificar os objetos dessa geometria até o momento imprecisa pelo seu alto grau de repetividade, irregularidade ou tortuosidade.

De um modo mais formal, um fractal é uma forma geométrica irregular ou fragmentada que pode ser subdividida em partes, e cada parte será (pelo menos aproximadamente) uma copia reduzida da forma toda. Os fractais são geralmente semelhantes entre si e independente da escala considerada. Muitos exemplos de fractal clássicos são encontrados na natureza, como a couve-flor e a samambaia que têm presentes as características de auto-semelhança. Se arrancarmos um pedaço da couve-flor, verificamos que este é igual à couve inteira, embora em menor dimensão como mostra na figura 1.1. O mesmo acontece com uma folha de samambaia ou um galho de uma árvore.

Figura 1.1 – Exemplo de fractal encontrado na Natureza, a couve-flor.

Tecnicamente, um fractal é um objeto que não perde as suas características de forma, à medida que é ampliado, mantendo a sua estrutura idêntica à original. O mesmo não acontece com muitas outras formas, como, por exemplo, com uma esfera ou uma circunferência, que parece perder a sua curvatura à medida que ampliamos uma das suas partes, reduzindo-se a um plano ou reta.

1.2.2 - Características de um Fractal

Visando uma melhor compreensão da geometria fractal em termos mais objetivos e concretos, iremos explorar agora alguns dos principais aspectos que a diferenciam da geometria euclidiana tradicional. Apesar de enorme variedade de formas e figuras que hoje são "catalogadas" como fractais, é possível enumerar três características essenciais que estão presentes em todos os fractais e que podem servir, pelo menos inicialmente, para caracterizar esse novo conjunto de formas geométricas. A primeira dessas características está relacionada à aparência dos fractais e, apesar de ser essencialmente intuitiva e visual, é importante para um reconhecimento inicial dessas figuras. A segunda característica se relaciona a um aspecto teórico dos fractais, importantíssimo do ponto de vista matemático e principal responsável pela grande ruptura que a geometria fractal causou na Matemática tradicional. A terceira e última se relaciona à maneira como os fractais são construídos, uma vez que, mesmo utilizando objetos matemáticos completamente distintos, teremos sempre um mesmo enfoque básico na construção de todos os tipos de fractais.

Visualmente, os fractais possuem como característica básica à **"auto-**

semelhança"*,* o que, em termos simples, significa dizer que pequenas partes da curva repetem a forma da curva como um todo, ou seja, se fizermos uma ampliação de uma região específica de um fractal (mesmo quando ampliados milhares de vezes), iremos encontrar uma réplica do fractal como um todo.

Essa característica, como dissemos, é principalmente visual e intuitiva, visto que a auto-semelhança exata é obviamente um conceito artificial, pois não é possível encontrar na Natureza objetos rigorosamente iguais a si mesmos. Formalmente, uma figura possui auto-semelhança exata se, para qualquer que seus pontos, existe uma vizinhança que contém uma parte da figura semelhante a toda a figura. Apenas em termos abstratos ou em formas teóricas matemáticas podemos conceber tal situação. O mesmo já não se pode dizer em relação à auto-semelhança aproximada, que não sendo também verdadeiramente real, pois estamos limitados pela escala, mas podemos encontrar boas aproximações em formas naturais.

A *tríade de Kock,* mostrada na figura 1.2, é um exemplo auto-semelhança exata. Este fractal recebeu o nome do criador, o matemático Von Kock.

Em cada geração do algoritmo recursivo que descreve a curva (mostrado na figura 1.2 a partir da segunda geração e até a oitava geração).

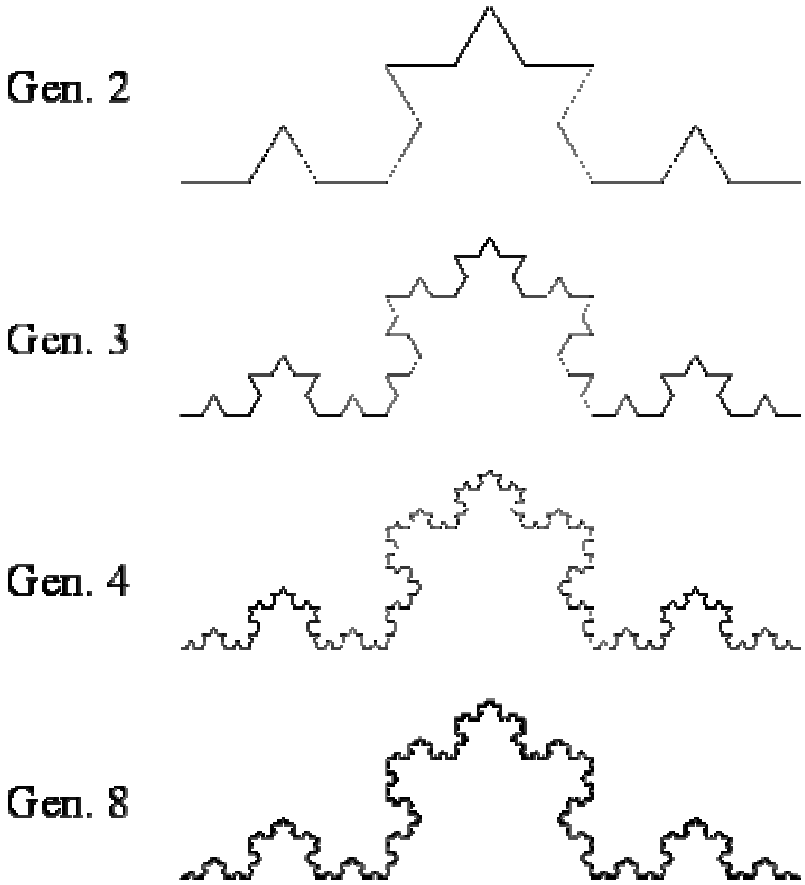


Figura 1.2 – Gerações da curva de Koch.

No caso de muitos tipos de árvores podemos encontrar uma certa semelhança entre as pequenas folhas que unidas constituem um pequeno ramo, que por sua vez combinado a outros ramos constitui os galhos, que conjuntamente com outros galhos maiores e assim sucessivamente irão gerar uma árvore. A árvore final não é muito diferente de cada ramo e das folhas iniciais.

No caso de árvores e outras formas da Natureza estamos perante a auto-semelhança aproximada. Esta forma de auto-semelhança pode ser observada na figura 1.3, onde um arbusto é gerado pela repetição de uma forma simples original.

Figura 1.3 – Geração de um arbusto - iteração 1, 2, 3, 4, 8 respectivamente.

A segunda característica dos fractais que iremos explorar é mais teórica e se refere à sua dimensão. Até então, a dimensão de uma figura era tradicionalmente representada por números inteiros. Na geometria euclidiana, pontos têm dimensão zero, retas têm dimensão um, planos têm dimensão dois e o espaço, dimensão três. As demais formas geométricas se encaixam perfeitamente em alguma dessas categorias. Porém, nos fractais criou-se à necessidade de uma nova abordagem, uma vez que as formas geradas não se encaixavam nas categorias euclidianas. Dessa forma, as figuras criadas pela geometria fractal acabaram por sugerir um outro tipo de análise, a "**dimensão fractal**", o que acabou fazendo com que o valor que irá expressar a dimensão de um objeto fractal seja, em geral, um valor não inteiro.

A dimensão de um fractal representa o grau de ocupação deste na reta, no plano ou no espaço, tendo a ver com sua distribuição e o seu grau de irregularidade das curvas dos conjuntos de pontos ou dos objetos fractais.

Objetos muito complexos, tais como a folhagem de uma árvore, o relevo de uma montanha ou a estrutura interna dos pulmões, podem realmente ser

estruturas tridimensionais. Nós podemos, portanto, pensar na irregularidade como um aumento na dimensão: uma curva irregular possui uma dimensão intermediária entre 1 e 2, enquanto que uma superfície irregular possui uma dimensão entre 2 e 3. Em ambos os casos, a dimensão é fracionária.

A terceira e última característica dos fractais se referem a sua construção, que sempre utiliza, de uma forma ou outra, algum tipo de processo iterativo, significando que, na construção de qualquer fractal, iremos repetir um determinado procedimento infinitamente, seja este procedimento um conjunto de cálculo algébrico ou uma determinada construção geométrica. A está característica, dá-se o nome de **complexidade infinita.**

Esta característica da construção dos fractais é, em grande parte, a responsável pelo grande fascínio que estas figuras provocam, pois na maioria dos casos, os fractais são construídos a partir de elementos extremamente simples, mas que apesar disso dão origem a figuras com extraordinária complexidade e riqueza de detalhes, bastando para isso aumentar o número de iterações a efetuar. Assim, qualquer que seja o número de ampliações de uma determinada figura fractal, nunca obteremos a “imagem final”, uma vez que ela poderá continuar a ser infinitamente ampliada e repetida.

1.3 - Geometria Euclidiana e Geometria Fractal

Durante muito tempo os objetos e os conceitos da geometria euclidiana foram considerados os que melhor descreviam o mundo que vivemos. A descoberta de uma geometria não-euclidiana introduziu novos objetos que representam certos fenômenos do Universo, tal como se passou com os fractais.

Para melhor se entender algumas características da Geometria Fractal, começaremos analisando resumidamente aquilo em que se opõe à Geometria Euclidiana.

A Geometria Euclidiana existe há mais de 2000 anos enquanto que a Geometria Fractal, com os seus princípios estabelecidos, tem pouco mais de 30 anos.

Um outro aspecto importante está relacionado com o fato da Geometria Fractal se adaptar bem à representação de objetos naturais. Assim, ao contrário dos objetos criados pelo homem que são caracterizados por possuírem linhas, ângulos retos, círculos perfeitos, etc. os objetos naturais estão repletos de irregularidades, assimetrias, repetições e “imperfeições”.

Enquanto que a Geometria Euclidiana se serve normalmente de fórmulas e equações para se exprimir, a Geometria Fractal, prefere os algoritmos recursivos e as fórmulas iterativas, pelo que está intimamente ligada à utilização dos computadores como ferramenta indispensável ao seu entendimento e representação.

A seguinte tabela faz a comparação entre a Geometria Euclidiana e a Geometria Fractal:

Geometria Euclidiana	Geometria Fractal
Tradicional (mais 2000 anos)	Moderna (aproximadamente 30 anos)
Baseada em tamanho ou escala definida	Sem tamanho ou escala específica
Apropriada a objetos feitos pelo Homem	Apropriada para formas naturais
Descrita por fórmulas e equações	Descrita por algoritmos recursivos
Dimensão no conjunto dos números inteiros	Dimensão no conjunto dos números fracionários

Neste capitulo fizemos uma rápida introdução ao conceito de geometria fractal. Muito mais pode ser escrito ou lido sobre o assunto na vasta bibliografia existente, mas acreditamos que essa rápida introdução já é suficiente para os objetivos deste trabalho.

2 - L-SYSTEM

Uma das formas mais eficientes de descrever o desenvolvimento de sistemas biológicos e fractais é através do formalismo denominado L-System, objeto deste capitulo de nossa monografia.

2.1 - L-System – Introdução

Sistemas Lindenmayer, mais considerados 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. Os *L-systems* são um tipo particular de sistema simbólico dinâmico que dá uma interpretação geométrica para a evolução dos sistemas. Mais recentemente, os L-systems têm tido muitas aplicações em computação gráfica. Duas das principais áreas incluem geração de fractais e modelagem realística de plantas.

L-system é uma linguagem para modelar o desenvolvimento celular. Células são representadas por símbolos e a subdivisão ou reprodução é modelada pela troca desses símbolos por *strings* de símbolos. Para melhor explicar, vejamos 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 ou multiplicação das células. No nascimento o organismo é uma única célula A ou B. Depois de uma subdivisão o organismo tem duas células

representadas pela string AB ou BA dependendo da célula inicial ser A ou B. Depois de duas subdivisões, cada célula subdividiu-se de acordo com as regras de subdivisão acima, o organismo tem quatro células dadas pela string ABBA ou BAAB. Depois de três subdivisões o organismo é representado pela string ABBABAAB (ou BAABABBA) e depois de quatro subdivisões o organismo tem 16 células representadas pela string ABBABAABBAABABBA.

A linguagem utilizada para gerar L-systems é axiomatizada e baseada em regras de reprodução, promovendo uma conexão entre o problema lógico e a solução visual. Ela usa a reescrita de símbolos como técnica básica, criando assim objetos complexos pela interação de um conjunto de regras ou produções para substituir partes de uma geometria inicial simples. A forma inicial é chamada de inicializador (ou axioma) e as regras que dão instruções de substituições são chamadas de geradores ou produções.

A idéia central para L-systems é a noção de reescrita (substituição), onde o objetivo básico é definir objetos complexos por sucessivas substituições de partes simples do objeto usando um conjunto de regras ou produções de reescrita. Uma reescrita pode ser feita recursivamente a partir do estágio anterior.

Os sistemas de reescrita mais estudados e entendidos operam em *strings* de caracter. O trabalho de Chomsky em linguagens formais (1957) gerou um grande interesse em sistemas de reescrita. Subseqüentemente, surgiu um período de fascinação com sintaxe, gramáticas e suas aplicações em ciência da computação, dando inicio ao campo das linguagens formais.

O trabalho de Aristid Lindenmayer introduziu um novo tipo de mecanismo de reescrita de strings chamado L-system. A diferença essencial

entre as gramáticas de Chomsky e L-systems se dá no método de aplicação das produções. Nas gramáticas de Chomsky as produções são aplicadas seqüencialmente, enquanto que nos L-systems elas são aplicadas em paralelo, substituindo simultaneamente todas as letras ou saindo diversas formas de substituição de uma dada palavra. Esta diferença reflete a motivação biológica dos L-systems. As produções pretendem simular as divisões e multiplicações celulares em organismos multicelulares, em que muitas divisões podem ocorrer ao mesmo tempo.

Os L-systems têm a propriedade única de gerar as produções em paralelo. Isto significa que qualquer problema que dependa de permutações para sua solução pode ser calculado simultaneamente ao invés de seqüencialmente. Por exemplo, tomando-se uma linha reta e dividindo-a em duas, dobra o número de segmentos. Sucessivas subdivisões redobram o número de segmentos.

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

Muitos organismos vivos se desenvolvem de tal maneira que produzem modelos fractais. Exemplos incluem até 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 ser aproximadamente resumido como um conjunto de regras simples.

2.1.1 - Formalizando os L-Systems

Assim os L-systems são conjuntos de regras e símbolos (usando a nomenclatura conhecidas como "gramáticas formais") que modelam processos de crescimento. Um simples L-system L é uma gramática como esta,

L = <V, w, P>

que contém três elementos: o alfabeto usado, V, as palavras, w e as produções P.

1. ALFABETO

O alfabeto é um conjunto finito V de símbolos formais, normalmente letras a, b, c, etc., ou possivelmente outros caracteres. Eles podem ser de dois tipos:

- VARIÁVEIS: são símbolos que denotam elementos que podem ser substituídos.
- CONSTANTES: são símbolos que denotam elementos que permanecem fixos, i.e., não podem ser substituídos.(são substituídos por ele mesmos a->a)

2. PRODUÇÕES

As regras de "syntax", ou as produções definem como as variáveis devem ser trocadas por constantes ou outras variáveis. Uma produção (ou regra de substituição) é um mapeamento de um símbolo para uma palavra x onde x pertence a V^* (conjunto de todas as palavras de V). Em nossa notação, P é o conjunto de produções ou regras.

3. AXIOMA

São expressões que definem como o sistema começa. O axioma (também chamado de inicializador) é uma string de símbolos do alfabeto usado V. Assim se V={a,b,c}, alguns exemplos de palavras possíveis são aabca, caab, b, bbc, etc. O comprimento |w| de uma palavra w é o número de símbolos na palavra.

São permitidos também mapeamentos de uma letra para a palavra vazia, ou para ela mesma. Se um símbolo não tem uma produção explicitamente definida, nós assumimos que ele é mapeado para ele mesmo por default. Neste caso o símbolo é uma constante do L-system.

2.2 - Gráficos Tartaruga (*Turtles Graphics*)

Além de representar os sistemas fractais ou biológicos é necessário muitas vezes reproduzir visualmente. Uma das formas mais eficientes de ser feito é através de idéia de autômatos virtuais reproduzidos nos Gráficos Tartarugas.

2.2.1 - A linguagem de programação LOGO e Gráficos Tartaruga

LOGO é uma linguagem de programação completamente definida por um conjunto de ações elementares e objetos, como números e listas, e uma sintaxe cujo objetivo é combinar ações e manipular objetos.

A estrutura central de controle da linguagem LOGO é a recursividade. Assim, a partir dos comandos fornecidos aos “Gráficos Tartarugas” (*turtle graphics*), torna-se possível a utilização de estruturas recursivas para gerar gráficos de alguns fenômenos geométricos, como os fractais, por exemplo. A figura é gerada a partir de um conjunto de traços deixados por uma “tartaruga robô” durante sua jornada pela tela, a qual é definida por uma seqüência organizada de ações que irá constituir um programa escrito e rodando em um sistema.

A linguagem de programação LOGO e a idéia da “tartaruga virtual” desenvolvida nos ambientes que a utilizam, foram inicialmente introduzidas por Seymour Papert, na década de 1960, com o objetivo de apresentar a Matemática e os computadores às crianças na escola primária. Porém, com o desenvolvimento da informática, da inteligência artificial e das pesquisas referentes aos ambientes informáticos de aprendizagem, esta idéia inicial se desenvolveu tremendamente e, atualmente, existem diversos softwares que se

utilizam da linguagem LOGO e da Geometria gerada pela “tartaruga virtual”, direcionados para diferentes propósitos e níveis de ensino. Assim sendo, apesar de seus objetivos iniciais, atualmente seria incorreto considerar o LOGO como uma linguagem direcionada a crianças do ensino primário.

Segundo Harold & DiSessa, a tradição de chamar as criaturas que aparecem nos ambientes baseados no LOGO de tartarugas começou com Grey Walter, um neurofisiologista que, no começo da década de 1960, realizou experimentos na Inglaterra com pequenas criaturas robóticas que ele chamou de *“tortoises”*. Isto inspirou as primeiras tartarugas (*turtles*) projetadas, pequenos robôs controlados por computador que se moviam no chão em resposta aos comandos FORWARD (para frente) e RIGHT (para a direita).

Conforme já dito, LOGO é uma linguagem de programação e como tal, se baseia em um conjunto de comandos e uma sintaxe que irá determinar as ações realizadas na tela do computador, ou seja, seu domínio é baseado em figuras que são desenhadas por uma tartaruga que se move na tela de acordo com um conjunto de comandos específicos da sintaxe de programação.

Os movimentos da tartaruga são controlados por um conjunto de comandos simples (chamados “primitivos”) que determinam ações especificas e previamente conhecidas pela tartaruga. Os principais comandos primitivos permitem que a tartaruga “ande” para frente (F ou forward) e para trás (B ou back), assim como “vire” para a direita (R ou right turn) e esquerda (L ou left turn). A tartaruga possui virtualmente uma caneta amarrada em sua cauda que, ao se mover, pode deixar uma linha por onde passa, se a caneta estiver “abaixada” ou se mover sem deixar a linha se “elevada” (neste caso “f” ande para frente ou “b” para trás).

Os comandos primitivos relativos aos movimentos da tartaruga devem ser seguidos por um valor numérico ou parâmetro que especifique o número de passos dado pela tartaruga, quando ela é mandada para frente ou para trás, e quantos graus ela deve virar à direita ou esquerda. Desta forma é disponibilizado um padrão de medida baseado na quantidade de passos dados pela tartaruga e um sistema de orientação baseado na direção apontada pela cabeça da tartaruga. Com essas primitivas, é possível fundamentar as ações da tartaruga tanto a partir de seus movimentos relativos quanto a partir de um sistema de coordenadas cartesianas ou polares.

Para o sistema de coordenadas cartesianas, convenciona-se o centro da tela como origem e o número de passos da tartaruga como escala de medida, utilizando, portanto, um sistema de coordenadas baseadas nas dimensões da tela em passos.

No sistema polar, utiliza-se igualmente o número de passos como medida de distância e o ângulo, em graus, da orientação da tartaruga como direção. Convencionou-se o sentido horário como padrão de orientação e a direção vertical, para cima como ângulo de 0º.

Por exemplo, pode-se formar um desenho de “um degrau” usando os comandos (R 90º F 100, L 90º F 100, R 90º F 100) como mostra a figura 2.1

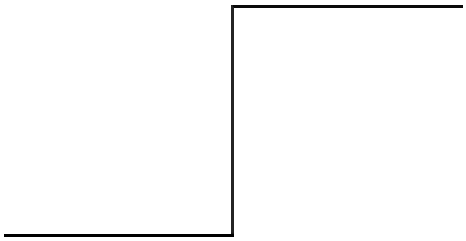


Figura 2.1 – Movimento simples da tartaruga

Um fato que é muito comum nas construções com a linguagem de programação LOGO é a presença de um primitivo especifico para facilitar a iteração de comandos na programação, o comando “repeat”.

Outra característica fundamental da linguagem LOGO é a possibilidade de se “criar” novas ferramentas a partir de objetos inicialmente fornecidos pelo ambiente. No LOGO, essa característica é disponibilizada pelos “procedimentos” de programação e pelo uso de “parâmetros”.

O uso de “procedimentos” de programação no LOGO cria então uma importante possibilidade de produções complexas, ou seja, a partir desse recurso é possível estender o vocabulário do ambiente com a criação de novos comandos que utilizam combinações de comandos primitivos. Dessa forma, um procedimento irá “ensinar” a tartaruga a executar tarefas mais complexas a partir da combinação de comandos primitivos que ela já “conhece”.

Existe a possibilidade de se inserir um conjunto de “parâmetros” e randomicidade num procedimento do LOGO no sentido de atribuir um maior grau de generalidade ao procedimento. É também possível criar

procedimentos que só ocorram sob dadas condições ou que tenham associados a sua ocorrência aleatoriamente e parâmetros randômicos.

2.2.2 - Algoritmos Recursivos

Um algoritmo recursivo é aquele que atinge o objetivo pretendido invocando a si próprio (direta ou indiretamente).

Uma função que se chama, tem que estar prevista alguma condição para a qual essas sucessivas invocações terminem, ou então o processo irá se tornar infinito. Esta condição está tipicamente associada ao caso mais simples que a função pode processar, e ao qual corresponde à última chamada da função. É após esta última execução que a função devolve um primeiro valor, que é passado á penúltima chamada que a função fez a si mesma. Por sua vez, esta já consegue terminar as operações pendentes e também ela devolve um valor para a antepenúltima chamada, e assim por diante até a primeira chamada da função, que finalmente termina de executar e devolve o resultado e o controle ao programa principal.

Um exemplo de função recursiva é a função fatorial que é definida por:

- $n ! = n \cdot (n - 1) \cdot (n - 2) \cdots 1$

ou

$$n ! = \begin{cases} n \cdot (n-1)! & if\ n > 1 \\ 1 & if\ n = 1 \end{cases}$$

Um outro exemplo de função recursiva que gera a imagem fractal conhecida como curva de Kock já comentado no capitulo anterior e mostrada na figura 1.2 é definida pelos passos expressos a seguir.

$$K_n = \begin{cases} K_{n-1}LK_{n-1}RRK_{n-1}LK_{n-1} & if\ n > 0 \\ F & if\ n = 0 \end{cases}$$

Nós podemos observar que o que fizemos apenas foi escrever um procedimento pequeno (que descreve a curva de Koch) cuja saída é um programa em uma outra linguagem (os comandos da tartaruga que descrevem K_n, com n = 3). Este fractal pode ser descrito na forma de uma curva fechada como mostrado na figura 2.2. Nesta forma a fractal é denominado “floco de neve” de Von Kock.

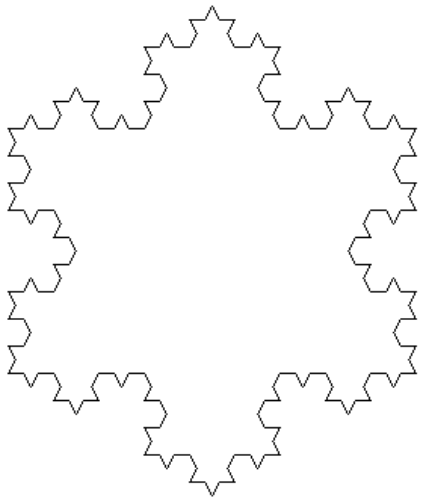


Figura 2.2 – Tríade de Kock na forma de curva fechada na iteração número 3

Outro exemplo de função recursiva que gera uma imagem fractal é a denominada fractal *tree*, ou árvore fractal, que na forma básica pode ser vista na figura 2.3.

Para gerar este fractal que chamamos de T_n, nós começamos com um tronco, e adicionamos dois ganhos no alto. Então, na extremidade de cada galho, nós adicionaremos também mais dois galhos. Como no exemplo anterior da curva de Kock, K_n nós podemos usar recursão, observando que T_n contém

duas cópias pequenas de si mesmo T_{n-1} que formam os membros (galhos) esquerdos e direitos da árvore. Assim, nós podemos definir T_n através de uma função recursiva.

Ou seja, se T_0 = FB, e nós podemos descrever T_1 como o “FS LFBR RFB LGB”. Do trecho “LFBR” sai o galho e os retornos esquerdos, e o do trecho “RFBL” sai o galho direito e o retorno. Para começar T_2, nós substituímos cada cópia do FB em T_1 com uma versão escalada de todo o T_1, após ter girado um ângulo para a esquerda ou direita. A fórmula de recorrência do fractal árvore é:

- $T_n = \begin{cases} FSLT_{n-1}RRT_{n-1}LGB & if\ n > 0 \\ FB & if\ n = 0 \end{cases}$

Nesta fórmula foram introduzidas mais duas primitivas. S (smaller) diminui o tamanho do galho e G (greater) aumenta o tamanho do galho.

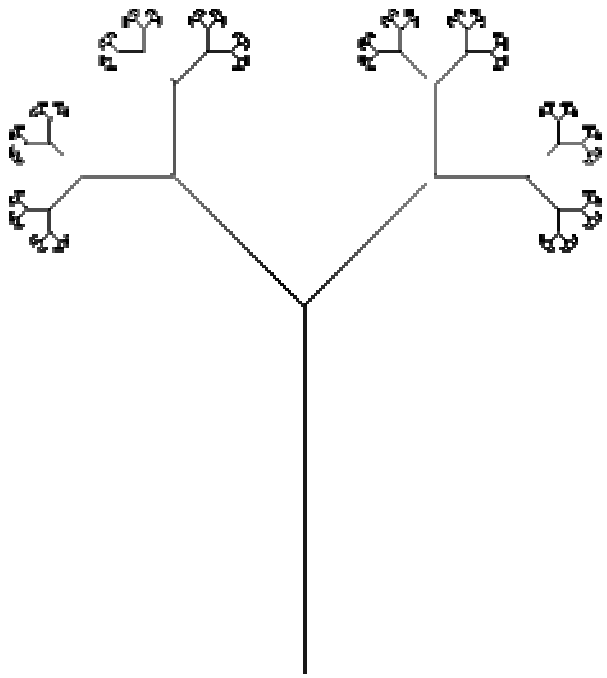


Figura 2.3 – Árvore fractal simples sem mudança de ângulo e tamanho nos galhos.

Para gerar a figura da fractal árvore, mostrada na figura 2.3 usamos os comandos da tartaruga que descrevem T_n, com n = 8 e usamos também o ângulo fixo de $\pi/4$ (45º), a variação do comprimento e da largura da primitiva ande para frente, F, ou ande para trás, B, com outros valores para esses parâmetros os resultados que obteríamos seria completamente diferentes, como pode-se observar na figura 2.4.

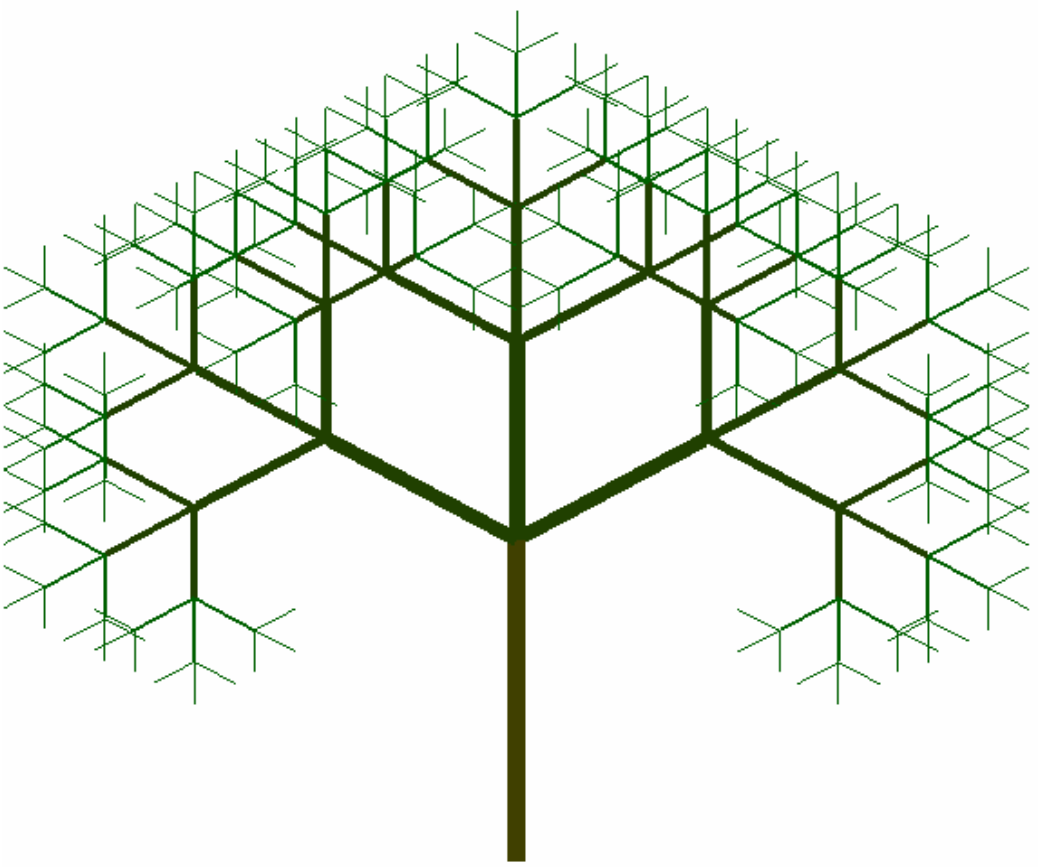


Figura 2.4 – Fractal árvore usando ângulo $\pi/3$ (60º). Com a primitiva variando a cor e a espessura, onde n = 7.

Neste capitulo, foi apresentado o algoritmo tartaruga, embora este já seja muito conhecido e utilizado, fizemos uma introdução de como ele funciona em 2D, será apresentado no capitulo 4, o funcionamento em 3D.

3 – OpenGL

Neste capitulo descreveremos os elementos básicos do OpenGL. Esta foi à forma para viabilizar os desenhos tridimensionais com os gráficos tartaruga que são o objetivo final deste trabalho.

3.1 - Conceitos Iniciais

OpenGL é uma interface de *software* para dispositivos de *hardware* (placas controladoras de vídeo). É uma biblioteca gráfica de modelagem e exibição tridimensional, bastante rápida e portável para vários sistemas operacionais. Seus recursos permitem ao usuário criar objetos gráficos 3D com qualidade próxima à de um *raytracer (técnica de renderização de uma cena que calcula a imagem desta cena simulando a forma como os raios de luz que percorrem o seu caminho no mundo real)*, de modo mais rápido que este último, além de facilitar aspectos avançados de animação, tratamento de imagens e texturas.

A biblioteca OpenGL (*Open Graphics Library*) foi introduzida em 1992 pela Silicon Graphics, no intuito de conceber uma API (Interface para Programas de Aplicação) gráfica independente de dispositivos de exibição. Com isto, seria estabelecida uma ponte entre o processo de modelagem geométrica de objetos, situadas em um nível de abstração mais elevado, e as rotinas de exibição e de processamento de imagens implementadas em dispositivos (hardware) e sistemas operacionais específicos. A função utilizada pelo OpenGL para desenhar um ponto na tela, por exemplo, possui o mesmo nome e parâmetros em todos os sistemas operacionais nos quais OpenGL foi

implementado, e produz praticamente o mesmo efeito de exibição em cada um destes sistemas.

Diante das funcionalidades providas pelo OpenGL, tal biblioteca tem se tornado um padrão amplamente utilizado na indústria de desenvolvimento de aplicações. Este fato tem sido impulsionado também pela facilidade de aprendizado, pela estabilidade das rotinas e pelos resultados visuais consistentes para qualquer sistema de exibição concordante com este padrão. Diversos jogos, aplicações científicas e comerciais tem utilizado OpenGL como ferramenta para a apresentação dos recursos visuais, principalmente com a adoção deste padrão por parte dos fabricantes de placas de vídeo destinadas aos consumidores domésticos.

O OpenGL pode ser usado em muitas linguagem de programação. Todas as rotinas do OpenGL são implementadas em C.

Entre os recursos gráficos disponíveis pelo OpenGL, podem ser destacados os seguintes:

- Modos de desenho de pontos;
- Ajuste de largura de linhas;
- Aplicação de transparência ;
- Ativação/desativação de tratamento para melhor o efeito de serrilhamento (*aliasing*);
- Mapeamento de superfícies com textura;
- Seleção de janela de desenho;
- Manipulação de fontes e tipos de iluminação e sombreamento;
- Transformação de sistemas de coordenadas.

- Transformações em perspectiva
- Combinação de imagens (*blending*)

As implementações do OpenGL geralmente provêem bibliotecas auxiliares, tais como a GLUT (OpenGL Utility library), utilizada para realizar tarefas comuns, tais como manipulação de matrizes, geração de superfícies e construção de objetos por composição.

3.2 - Utilização

Como uma API, OpenGL segue a convenção de chamada de funções da linguagem C. Isto significa que programas escritos em C podem facilmente chamar funções desta API, tanto porque estas foram escritas em C, como porque é fornecido um conjunto de funções C intermediárias que chamam funções escritas em *assembly* ou outra linguagem.

Apesar de OpenGL ser uma biblioteca de programação "padrão", existem muitas implementações desta biblioteca, por exemplo, para *Windows* e para *Linux*. Também existem implementações para os compiladores do *Visual C++*, *Borland C++*, *Dev-C++*, *Delphi* e *Visual Basic.*

3.3 - Linhas, Pontos e Polígonos

Com apenas algumas primitivas simples, tais como pontos, linhas e polígonos, é possível criar estruturas complexas. Em outras palavras, objetos e cenas criadas com OpenGL consistem em simples primitivas gráficas que podem ser combinadas de várias maneiras. Portanto, OpenGL fornece ferramentas para desenhar pontos, linhas e polígonos, que são formados por um ou mais vértices. Neste caso, é necessário passar uma lista de vértices, o que pode ser feito entre duas chamadas de funções OpenGL:

glBegin()

glEnd()

O argumento passado para *glBegin()* determina qual objeto será desenhado. No exemplo fornecido, para desenhar três pontos pretos foi usada a seguinte seqüência de comandos:

```
glBegin(GL_POINTS);
        glColor3f(0.0f, 0.0f, 0.0f);
        glVertex2i(100, 50);
        glVertex2i(100, 130);
        glVertex2i(150, 130);
glEnd();
```

Para desenhar outras primitivas, basta trocar *GL_POINTS*, que exibe um ponto para cada chamada ao comando *glVertex*, por:

- GL_LINES: exibe uma linha a cada dois comandos *glVertex*;

- GL_LINE_STRIP: exibe uma seqüência de linhas conectando os pontos definidos por *glVertex*;
- GL_LINE_LOOP: exibe uma seqüência de linhas conectando os pontos definidos por *glVertex* e ao final liga o primeiro como último ponto;
- GL_POLYGON: exibe um polígono convexo preenchido, definido por uma seqüência de chamadas a *glVertex*;
- GL_TRIANGLES: exibe um triângulo preenchido a cada três pontos definidos por *glVertex*;
- GL_TRIANGLE_STRIP: exibe uma seqüência de triângulos baseados no trio de vértices v0, v1, v2, depois, v2, v1, v3, depois, v2, v3, v4 e assim por diante;
- GL_TRIANGLE_FAN: exibe uma seqüência de triângulos conectados baseados no trio de vértices v0, v1, v2, depois, v0, v2, v3, depois, v0, v3, v4 e assim por diante;
- GL_QUADS: exibe um quadrado preenchido conectando cada quatro pontos definidos por *glVertex*;
- GL_QUAD_STRIP: exibe uma seqüência de quadriláteros conectados a cada quatro vértices; primeiro v0, v1, v3, v2, depois, v2, v3, v5, v4, depois, v4, v5, v7, v6, e assim por diante.

3.4 - Transformações Geométricas

As transformações geométricas são usadas para manipular um modelo, isto é, através delas é possível mover, rotacionar ou alterar a escala de um objeto. A aparência final da cena ou do objeto depende muito da ordem na qual estas transformações são aplicadas.

A biblioteca gráfica OpenGL é capaz de executar transformações de translação, escala e rotação através de uma multiplicação de matrizes. A idéia central destas transformações em OpenGL é que elas podem ser combinadas em uma única matriz, de tal maneira que várias transformações geométricas possam ser aplicadas através de uma única operação.

Isto ocorre porque uma transformação geométrica em OpenGL é armazenada internamente em uma matriz. A cada transformação que é aplicada, esta matriz é alterada e usada para desenhar os objetos a partir daquele momento. A cada nova alteração é feita uma composição de matrizes. Para evitar este efeito "cumulativo", é necessário utilizar as funções *glPushMatrix()* e *glPopMatrix()*, que salvam e restauram, respectivamente, a matriz atual em uma pilha interna da OpenGL.

A **translação** é feita através da função *glTranslatef(Tx, Ty, Tz)*, que pode receber três números *float* como parâmetro. Se os números forem *double* a função será *glTranslated*. Neste caso, a matriz atual é multiplicada por uma matriz de translação baseada nos valores dados.

A **rotação** é feita através da função *glRotatef(Ângulo, x, y, z)*, que pode receber quatro números *float* como parâmetro. Se os números forem *double* a função será *glRotated*. Neste caso, a matriz atual é multiplicada por uma matriz de rotação de "Ângulo" graus ao redor do eixo definido pelo vetor "x,y,z" no sentido anti-horário.

A mudança de **escala** é feita através da função *glScalef(Ex, Ey, Ez)*, que pode receber três números *float* ou três parâmetros *double*. Neste caso, a matriz atual é multiplicada por uma matriz de escala baseada nos valores dados.

Neste capitulo, foram apresentados os pontos mais usados do OpenGL para implementar o sistema autômato virtual em 3D, gráfico tartaruga.

Muito mais pode ser escrito ou lido sobre o OpenGL em livros, artigos e sites, mas achamos que o que foi apresentado foi suficiente para mostrarmos o sistema.

4 – Projeto

Comandar um autômato para se mover no espaço é bem mais complexo que no plano. Mesmo que se suponha o autômato relacionado a um sistema de coordenadas cartesianas, onde “andar para frente” uma das 3 direções fixas, os comandos relacionados à rotação não são mais tão simples. Ao invés de uma única possibilidade de rotação teremos três possibilidades para nos gerar o ângulo de rotação. Este tratamento da tartaruga 3D é ponto básico do nosso trabalho.

4.1 – Objetivo

Após o estudo de algumas cadeiras na área de computação gráfica, em especial a cadeira de Geometria Fractal, onde foi desenvolvido um sistema capaz modelar estruturas em duas dimensões utilizando a geometria da tartaruga. Consideramos que muitas vezes, ao tentarmos representar uma figura espacial, em especial árvores no plano bidimensional, percebemos uma certa distorção em valores ou medidas (segmentos ou ângulo). Começamos então, a aprimorar e a implementar um sistema que utilizasse a geometria da tartaruga em três dimensões. Possibilitando assim, descrever objetos que se aproximem cada vez mais da realidade.

4.2 - Considerações iniciais

O sistema implementado é baseado na geometria da tartaruga onde foram inseridas novas primitivas que permitem a incorporação da característica 3D ao algoritmo da tartaruga, permitindo assim a movimentação através do espaço, pois a tartaruga move-se nos eixos (X,Y,Z).

Ao mesmo tempo em que conseguimos chegar mais próximo do “real” com esta nova implementação, deparamo-nos a um sistema de representação através de duas dimensões dos objetos, pelo uso de um plano representado pela tela do monitor como saída. Para solucionar esse problema foi usado a API OpenGL que trabalha internamente com projeções dos objetos em 3D no plano 2D que conserva a característica tridimensional.

Para os desenhos 3D usuais os principais primitivos devem permitir que a tartaruga “avance” (uma distância “d” para frente ou uma distância “–d” para trás), “avance sem desenhar” (uma distância “d” para frente ou uma distância “–d” para trás) e “gire” (um ângulo α em torno do eixo “Y” e um ângulo β em torno da normal ao plano Y formado pelo eixo Y e a projeção P’ do ponto P no plano “XZ”), para este foi usado coordenadas esféricas.

Assim o comando “**gira**”: passa a ser parametrizado com dois ângulos, sendo o primeiro o ângulo “XZ” (no plano ou em torno do eixo “Y”) e o segundo o ângulo “Y” (em torno da normal ao plano Y formado pelo eixo Y e a projeção P’ do ponto P no plano “XZ”).

```
Procedure Tartaruga.gira ( AngXY, AngY: Real );
    Begin
       anguloY:= anguloY + angY;
       anguloXZ:= anguloXZ + angXY;
    End;
```

O “anguloY” e “anguloXZ” são variáveis de instância da classe “tartaruga”. Sendo mais bem explicadas na figura a seguir 4.1:

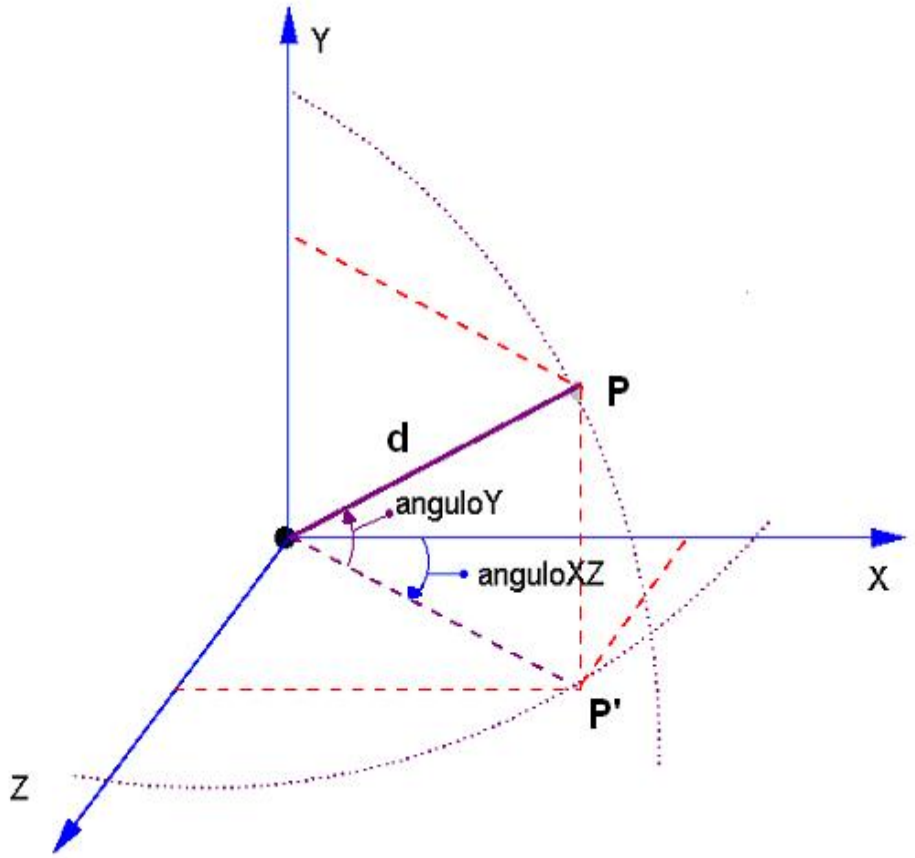


Figura 4.1 – As coordenadas do sistema.

Se “d” for o tamanho atual de deslocamento da tartaruga, essas coordenadas usadas para a descrição da tartaruga 3D têm a seguinte relação com coordenadas cartesianas:

1. Y = d sen (anguloY);
2. Z = d cos (anguloY) * sen (anguloXZ);
3. X = d cos (anguloY) * cos (anguloXY).

O comando “**avanca**”: deve ser parametrizado com a distância “d”, sendo esta um valor positivo ou negativo, que indica a distância que será percorrida pela tartaruga. Considerando as direções X,Y,Z, esta função pode ser descrita como:

```
Procedure Tartaruga.avanca(distancia: Real);
   Begin
      X:= X + ( distancia * cos( anguloY * pi/180) * cos( anguloXZ * pi/180) );
     Y:= Y + ( distancia * sin( anguloY * pi/180) );
     Z:= Z + ( distancia * cos( anguloY * pi/180) * sin( anguloXZ * pi/180) );

     glVertex3f(xOld,yOld,zOld);
     glVertex3f(x,y,z);

     xOld:= X;
     yOld:= Y;
    zOld:= Z;
End;
```

O comando “**avancaSemDesenhar**”: deve ser também parametrizado com a distância “d”, sendo esta positiva ou negativa, indicando a distância que será percorrida pela tartaruga, sem que a mesma desenhe na tela.

```
Procedure Tartaruga.avancaSemDesenhar(distancia: Real);
   Begin
      X:= X + ( distancia * cos( anguloY * pi/180) * cos( anguloXZ * pi/180) );
     Y:= Y + ( distancia * sin( anguloY * pi/180) );
     Z:= Z + ( distancia * cos( anguloY * pi/180) * sin( anguloXZ * pi/180) );
     xOld:= X;
     yOld:= Y;
    zOld:= Z;
End;
```

Para o melhor entendimento, de como será controlado o movimento 3D por estes comandos iremos explicar o funcionamento por meio de alguns exemplos:

- Exemplo 1 – Para produzir um ato de avançar sobre o eixo X, para isso é usado as seguintes linhas:
    1. gira (0,0);
    2. avança (1).

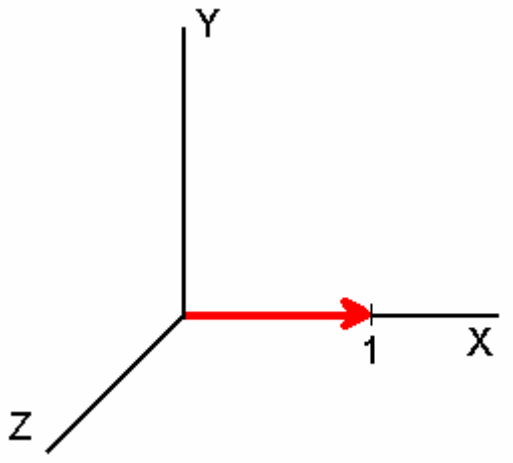


Figura 4.2 – Avança sobre o eixo X

.

- Exemplo 2 – Para produzir o ato de avançar sobre o eixo Y, para isso é usado as seguintes linhas:
    1. gira (0,90);
    2. avança (1).

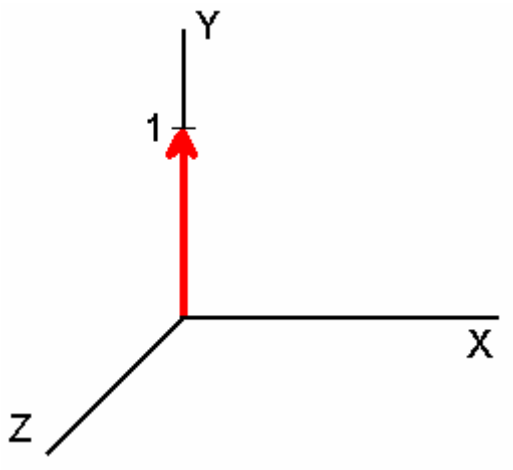


Figura 4.3 – Avança sobre o eixo Y.

- Exemplo 3 – Para produzir um efeito de avançar sobre o eixo Z, usamos as seguintes linhas:
    1. gira (90,0);
    2. avança (1).

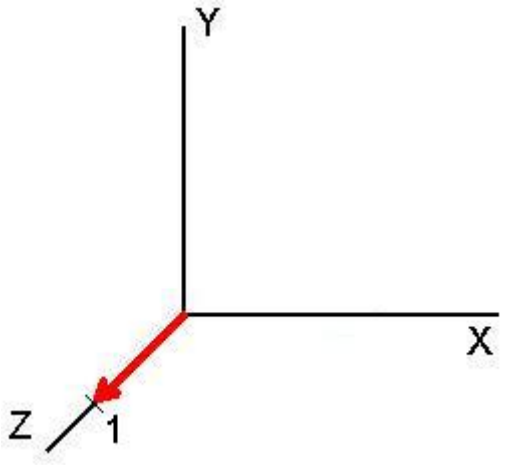


Figura 4.4 – Avança sobre o eixo Z.

A partir dessas primitivas simples fomos capazes de produzir imagens mais complexas como a *figura 4.5* mostra.

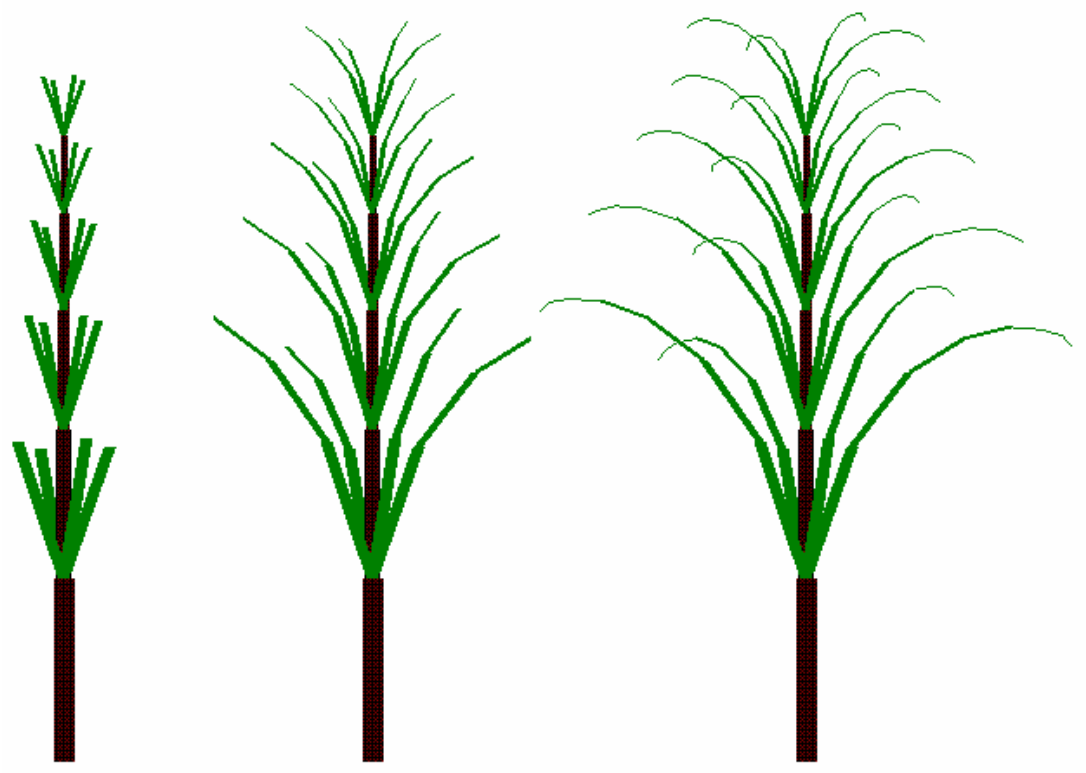


Figura 4.5 – Primeira árvore do sistema usando o algoritmo da tartaruga, com as respectivas iterações 2,4,8.

```
procedure TArvore.Arvore1;
   Se o NumIteracao não é máximo então
        Incrementa (NumIteracao);
        Tart.Gira( 0, -AnguloYGalho );
        Tart.Avanca( S );
        Arvore1 (S,NumIteracao,R,G,B);
        Tart.Anda_Sem_Desenhar ( -S );
        Tart.Gira( 0, AnguloYGalho );
   Fim Se
Fim
procedure TArvore.DrawArvore1;
   Faça 5 vezes
        Tart.Avanca(S*0.8);
        Tart.setAnguloXZ(15);
        Arvore1( S, 1, R, G, B );
        Tart.setAnguloXZ(105);
        Arvore1( S, 1, R, G, B );
        Tart.setAnguloXZ(195);
        Arvore1( S, 1, R, G, B );
        Tart.setAnguloXZ(285);
        Arvore1( S, 1, R, G, B );
    Fim Faça
Fim
```

Este pseudocódigo é apenas ilustrativo, mostrando apenas a idéia para a construção da planta da *figura 4.5, não levando em consideração o tamanho e nem a cor dos galhos.*

4.3 - O Sistema Desenvolvido

A escolha do Delphi como ambiente de programação foi, principalmente motivada, pela nossa experiência prévia e a verificação de que o ambiente escolhido permitia a construção de fractais. Além disso, o Delphi é uma linguagem que tem suporte a orientação objeto e também permite a utilização de funções da biblioteca OpenGL (gl.pas e glu.pas), o que nos permitiu a criação das classes TTartaruga e TArvore, facilitando assim o reconhecimento de erros, futuras manutenções e alterações na estrutura do código.

O Sistema oferece a possibilidade da criação de algumas árvores pré-definidas, permitindo pequenas alterações, como causar uma rotação, mudança de escala (zoom in e zoom out), probabilidade na criação de galhos, mudança de cor e número de iterações. As principais funções do sistema implementado são discutidas a seguir na figura 4.6:

1. **MÓDULO** – Através desse menu o usuário escolhe em qual módulo será trabalhado; três dimensões, onde são apresentadas plantas em 3D, duas dimensões onde são apresentadas plantas em 2D e diversos, onde são apresentadas figuras fractais diversos em 2D;
2. **MENU** – Através desse menu o usuário escolhe a planta ou figura fractal que deseja executar. Sempre que é escolhida uma nova figura são

setados os parâmetros padrões, para que a figura não perca suas características do desenho inicial;

3. **ZOOM** – Liga ou desliga o Zoom, ou seja, tem o efeito de Zoom in e Zoom out que com o auxilio do OpenGL muda a escala do objeto;
4. **RGB** – Define as cores iniciais do objeto fractal. A partir dessas cores iniciais a cada nova iteração o R (red) e o B (blue) são decrementados (tendem a zero) e o G (green) e incrementado (chegando a 255).
5. **Quantidade de Iterações** – Quantidade de vezes que a função geradora é executada recursivamente. Mínimo de duas iterações e o máximo recomendado de dez iterações;
6. **Probabilidade de criação do Galho** – Probabilidade para a criação de cada galho;
7. **Ângulo** – Esta opção é o ângulo delta que será somado ao anguloY (visto na figura 4.1);
8. **Painel** – Local onde é exibido o objeto fractal.

Rotação – A rotação da figura definida iterativamente pelo usuário ao clicar e arrastar o mouse, tanto com o botão esquerdo, quanto o botão direito.

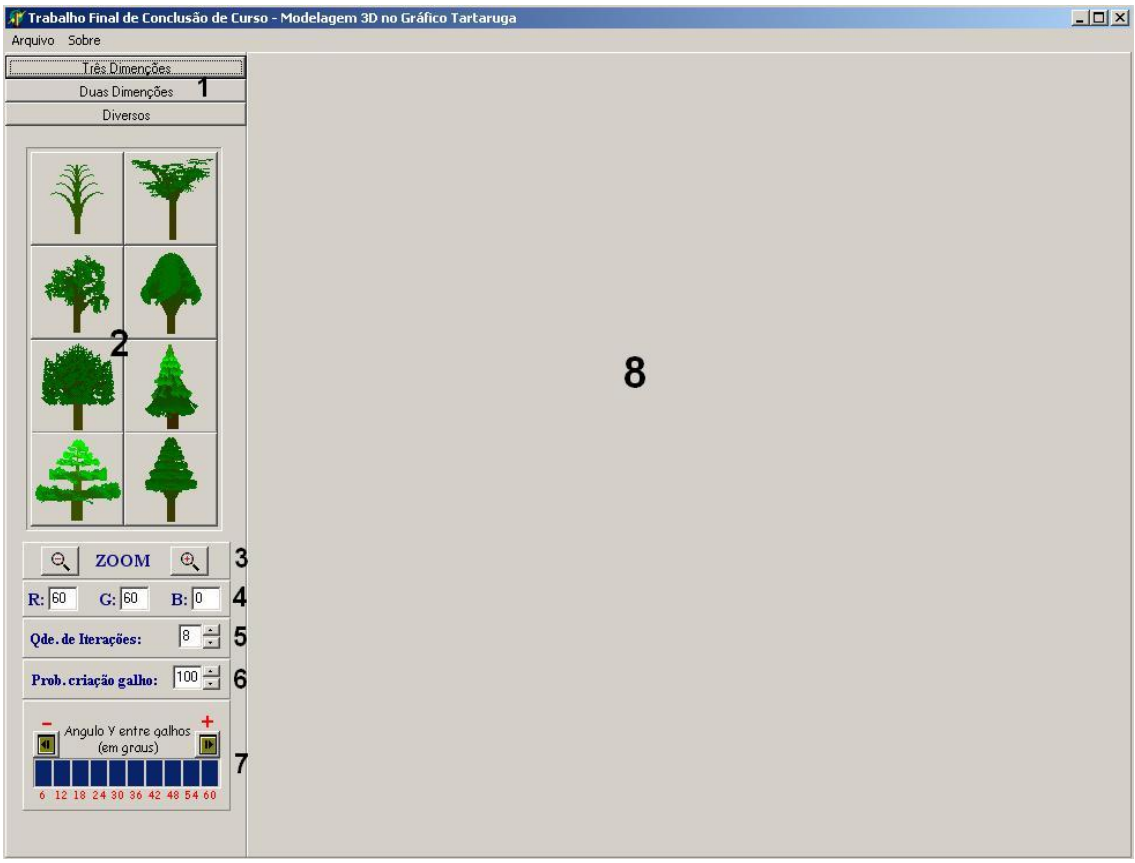


Figura 4.6 – Tela Principal do Sistema que implementa o Gráfico Tartaruga em 3D.

4.4 – Diagrama de Classe do Projeto

Este diagrama descreve os tipos de classes do sistema e os vários tipos de relacionamentos estáticos que existe entre elas.

A classe UArvore (classe árvore 3D) possui uma classe UTartaruga que executa ações simples conhecidas. A classe UArvore2D possui uma classe UTartaruga que executa ações simples conhecidas, mas não trabalhando com um dos ângulos (anguloY ou anguloXZ) zerado, para planificar as árvores fractais.

A classe GL e a classe GLU são classes que implementam funções do OpenGL (Open Source), onde são usadas algumas funções do OpenGL pela UTartaruga.

Obs1.: A classe principal onde se encontram os eventos não está definida pelo diagrama abaixo, esta classe usa as classes GL e GLU para executar algumas transformações geométricas implementadas nestas classes.

Obs2.: Alguns tipos de dados das funções ou variáveis foram convertidos para tipos de dados equivalentes do Java, pois não existem alguns tipos de dados do Delphi na ferramenta usada.

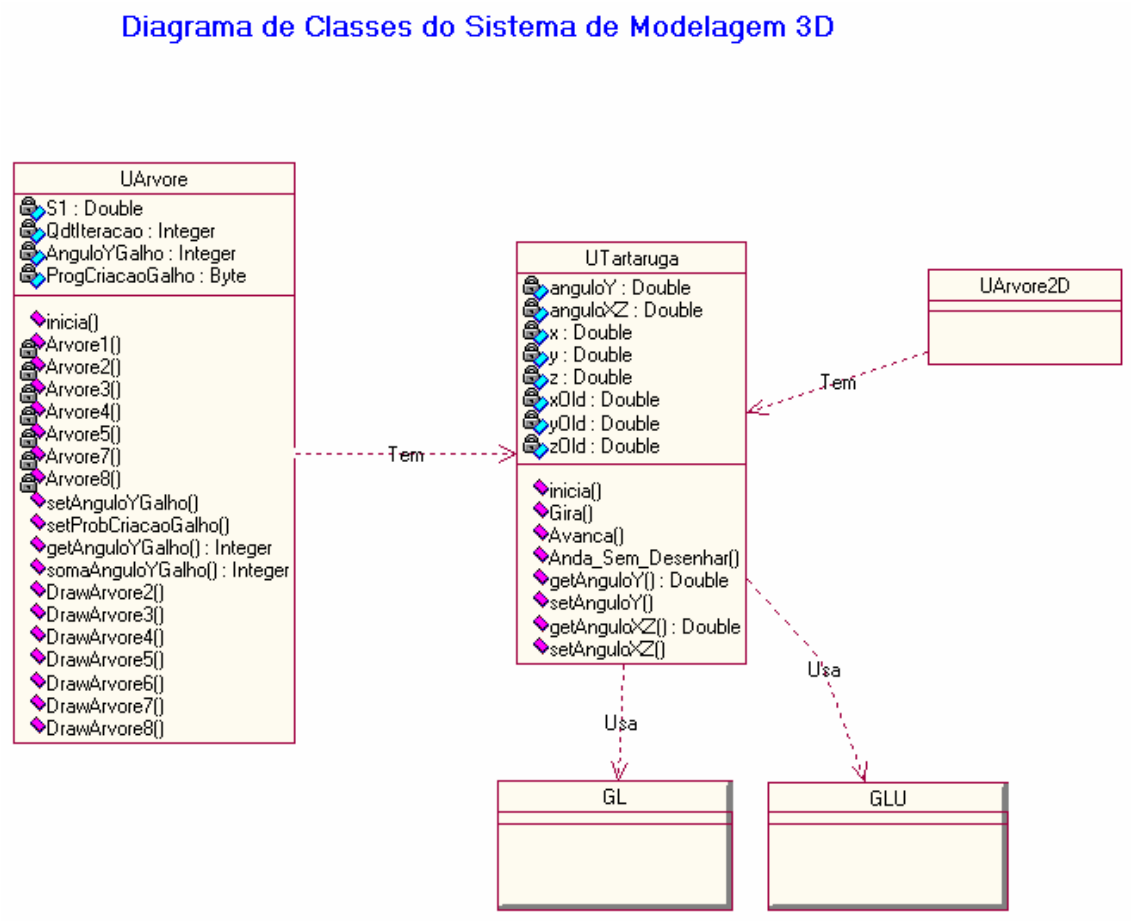


Figura 4.7 – Diagrama de Classes.

4.5 – Resultados

Neste item, apresentaremos algumas das imagens fractais criados pelo sistema.

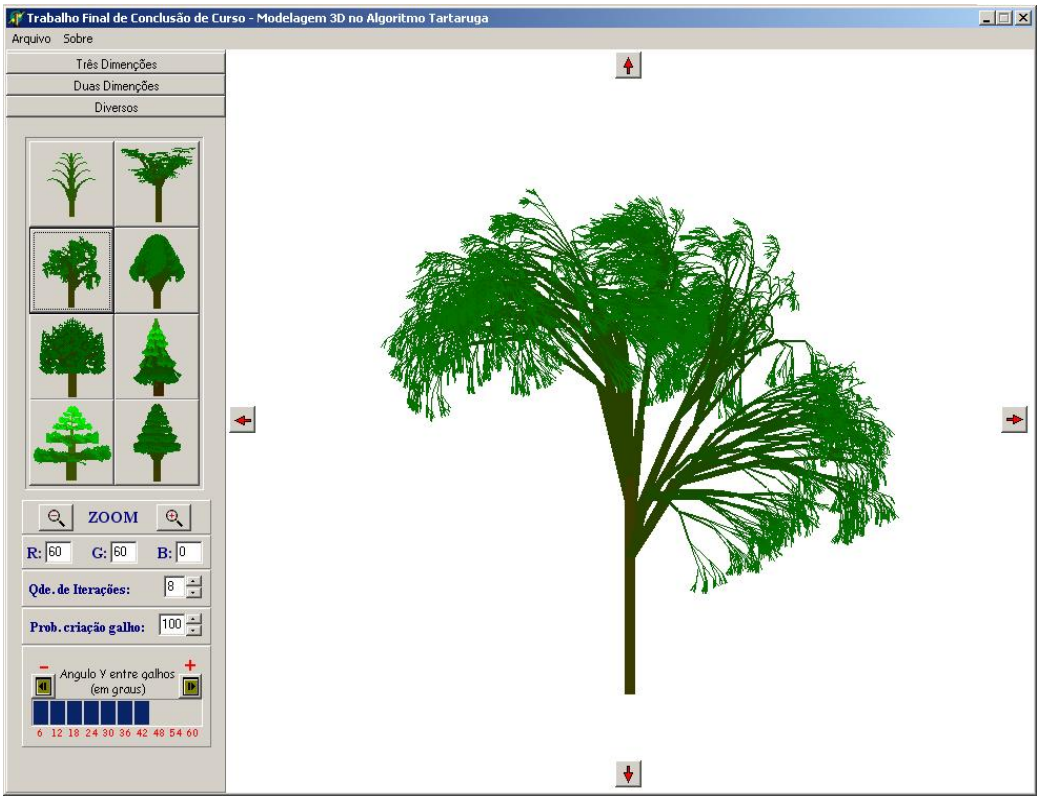


Figura 4.8 – Fractal criada pelo sistema no módulo de três dimensões.

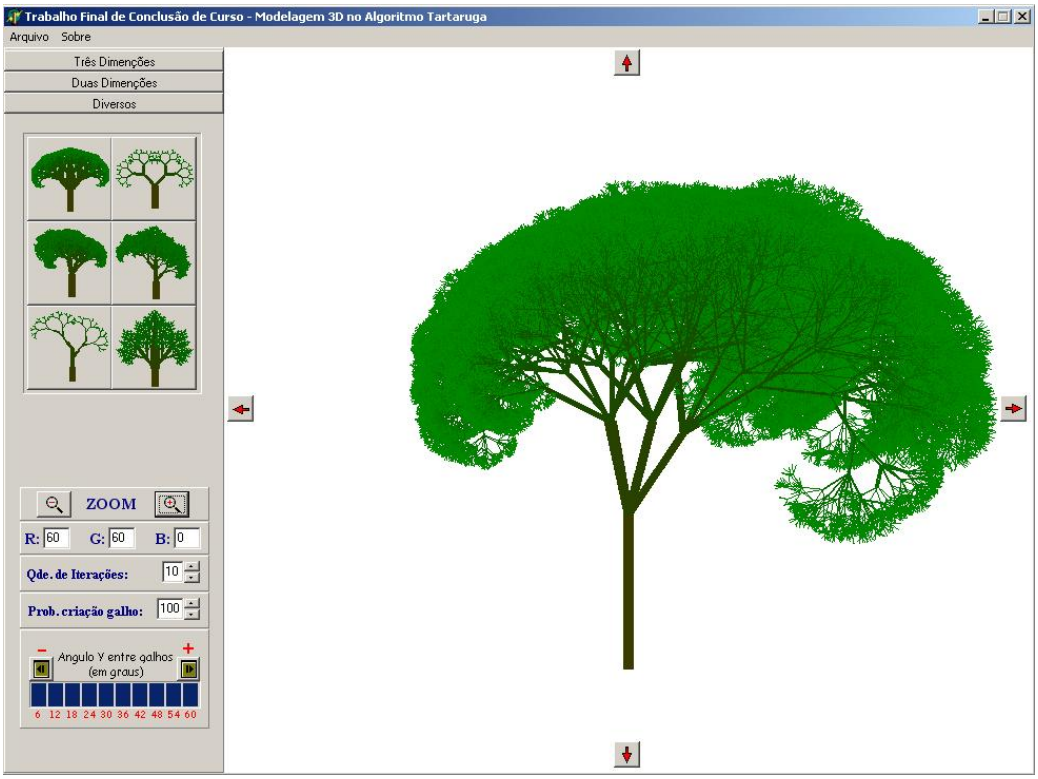


Figura 4.9 – Fractal criada pelo sistema no módulo de duas dimensões.

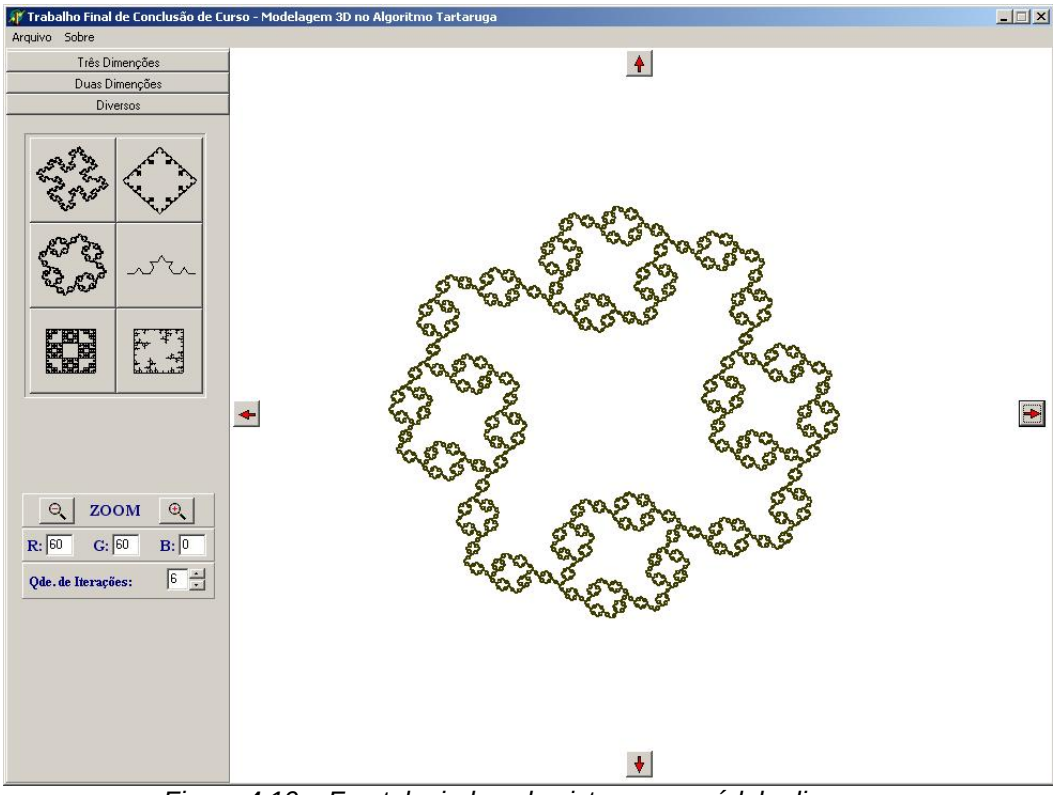


Figura 4.10 – Fractal criada pelo sistema no módulo diversos.

5 – Conclusão

A principal contribuição feita neste trabalho, consistiu a criação de um material contendo tópicos para o desenvolvimento da modelagem 3D dos Gráficos Tartaruga e também um sistema que implementa esse gráfico no desenho de plantas e algumas figuras fractais em 2D,

O sistema desenvolvido permite o desenho de fractais 2D e 3D, apresentando uma interface simples e intuitiva, onde a forma das árvores e de outras figuras fractais a serem geradas podem ser escolhidas em um menu gráfico pelo usuário..

Um dos principais desafios encontrados foi criar algoritmos que se aproximavam mais das plantas reais.

Com relação a trabalhos futuros, há muito a ser explorado e melhorado, como uso de threads, a utilização de texturas, modelos de iluminação e criação de outros algoritmos que gerem modelos mais realísticos.

6 - Referências Bibliográficas

ABELSON, H., diSESSA, A. A. (1981) *Turtle Geometry: The Computer as a Medium for Exploring Mathematics*. Cambridge: MIT Press.

AGUIAR, T.- *UML NO DESENVOLVIMENTO DE SISTEMAS DE INFORMAÇÃO.* Departamento de Ciência da Computação. UFF

AZEVEDO, E.; CONCI, A. – *Computação Gráfica Teoria e Pratica*, Campos, 2003.

CONCI, A – *Curso de Fractais e Mapeamentos* – em <http://www.ic.uff.br/~aconci/FRACTAIS.html>

ESCOLA SECUNDARIA DE PONTE DE SOR – *Geometria Fractal e a Natureza* – Disponível em <http://www.terravista.pt/ancora/2765/indfrac.htm> Acesso em: 18 de maio de 2004.

HILL, Francis S. *Computer Graphics Using OpenGL*. 2nd ed. Upper Saddle River, New Jersey: Prentice-Hall, Inc., 2000. 922 p.

JÚNIOR, A. M. B. – *Introdução á computação gráfica com OpenGL* – em: <http://www.dca.ufrn.br/~ambj/ele435/opengl/index.html>

MANDELBROT, B. B. – Objectos *Fractais: Forma, Acaso e Dimensão*, Tradução a partir da 3ª Edição francesa de Carlos Fiolhais, Lisboa: Gradiva, 1991.

PRUSINKEIWICZ, P., LINDENMAYER, A., and HANAN, J., "Developmental Models of *Herbaceous Plants for Computer Imagery Purposes,"* SIGGRPAH'88, pp. 141-150.

PAPERT, S.; SILVERMAN, B. – *MicroWords, v.2.05, Logo Computer Systems* Inc., Cambridge, 2000, MS Windows, software educativo (geometria da tartaruga) .

UNIVERSIDADE DE COIMBRA – *Artigo sobre Geometria Fractal, fundamentos e ensino da Álgebra*. Faculdade de Ciências e Tecnologia

VALENTE L. – *Artigo sobre OpenGL* - em: < http://www.ic.uff.br/~lvalente/>.

WRIGHT, Richard S. Jr.; SWEET, Michael. *OpenGL SuperBible*. 2nd ed. Indianapolis, Indiana: Waite Group Press, 2000. 696 p.

____ - *A turtle in fractal garden* – Disponível em <http://www.math.sunysb.edu/~scott/Book331/turtle_in_fractal.html> Acesso em: 17 de maio de 2004.

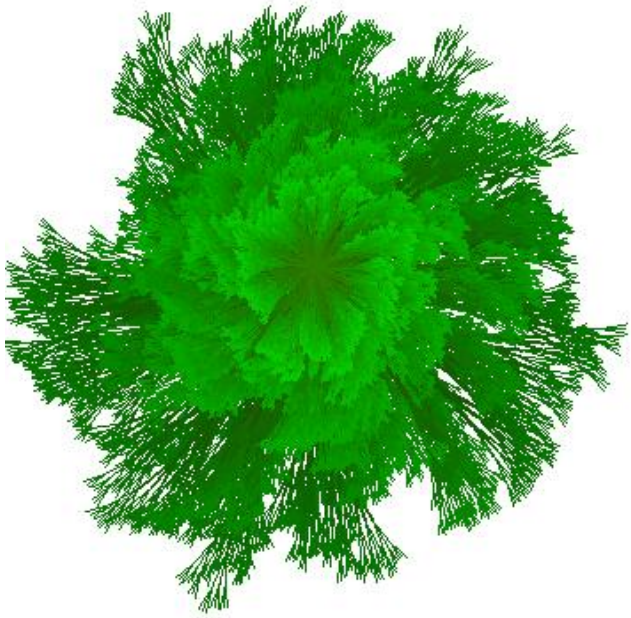
____ - *Página da Ciência pura e Aplicada* – Disponível em <http://tatooine.fortunecity.com/stephenson/51/matematica/geofractal.html> Matemática Moderna, Acesso em: 18 de maio de 2004.

7 – Anexo

Neste anexo, apresentaremos algumas figuras fractais 3D implementados no sistema.

7.1 Figura – Árvore Tridimensional onde existe uma randomicidade nos ângulos entre os galhos, tamanho dos galhos, tonalidade de cores.

7.2 Figura – Árvore Tridimensional chamada por nós de árvore de Natal.



7.3 Figura – Árvore de Natal vista de cima.