João Manuel dos Santos Martins

SISTEMAS DE LINDENMAYER

Modelação de Árvores com Recurso ao Maple

Departamento de Matemática Pura
Faculdade de Ciências da Universidade do Porto
Março de 2008

João Manuel dos Santos Martins

SISTEMAS DE LINDENMAYER

Modelação de Árvores com Recurso ao Maple

Tese submetida à Faculdade de Ciências da Universidade do Porto

para obtenção do grau de Mestre em Ensino da Matemática

Departamento de Matemática Pura

Faculdade de Ciências da Universidade do Porto

Março de 2008

José Carlos Santos

Porto, 3 de Julho de 2008

Resumo

O trabalho apresentado nesta dissertação teve por objectivo implementar a modelação (não realista) de árvores tridimensionais em linguagem *Maple*, tendo por base a teoria dos sistemas de Lindenmayer.

Criados em 1968 por Aristid Lindenmayer, os sistemas que mais tarde viriam a ter o seu nome são um exemplo de um mecanismo de reescrita de palavras. Através da aplicação em simultâneo das regras de substituição, este formalismo permite descrever o desenvolvimento de estruturas ramificadas. A palavra resultante do processo de substituição pode ser objecto de uma interpretação gráfica, permitindo assim a criação de imagens realistas ou meramente esquemáticas da estrutura modelada.

Os sistemas de Lindenmayer munidos com uma interpretação gráfica são uma ferramenta de modelação bastante versátil, no entanto, apresentam algumas limitações. Com o objectivo de colmatar algumas dessas limitações surgiram os sistemas de Lindenmayer paramétricos, como uma extensão do conceito original deste formalismo. Essa extensão consistiu na associação de parâmetros numéricos aos símbolos que representam as diferentes componentes dos organismos modelados.

O *Maple* faz parte de uma família já relativamente vasta de ambientes científicos designados por sistemas de Álgebra computacional. Recorrendo a esta linguagem criaram-se diversos procedimentos que permitiram mostrar o potencial prático dos diversos tipos de sistemas de Lindenmayer abordados nesta dissertação e, em especial, dos paramétricos, os quais

possibilitaram fazer a modelação de árvores tridimensionais.

Palavras-chave: mecanismos de reescrita, sistemas de Lindenmayer, interpretação gráfica de palavras, modelação, árvores, *Maple*.

Agradecimentos

O meu sincero e profundo agradecimento ao meu orientador, Professor Doutor Fernando Jorge Soares Moreira, pelo acompanhamento em todos os momentos da realização desta dissertação.

Agradeço também à minha família, à Angélica e a todos os meus amigos por todo o seu apoio e encorajamento.

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

1.1 Contextualização

A abordagem matemática a qualquer fenómeno natural requer sempre uma codificação e quantificação dos vários elementos usados na sua modelação. Neste trabalho apresentam-se mecanismos, chamados de reescrita, que tentam descrever as propriedades geométricas intrínsecas ao crescimento de diversas árvores. Esta técnica de reescrita consiste em definir objectos complexos fazendo substituições sucessivas de partes de um objecto inicial mais simples, usando para o efeito um conjunto de regras de substituição. Smith designou por "*database amplification*" esta capacidade de produzir objectos complexos a partir de pequenos conjuntos de dados [15].

Serão aqui essencialmente abordadas duas classes de mecanismos de reescrita. A primeira classe opera sobre objectos geométricos e permite gerar as figuras fractais clássicas. A segunda classe, mais detalhada neste trabalho, são os designados sistemas de Lindenmayer que operam sobre palavras de um conjunto predefinido de símbolos.

A geometria fractal, introduzida por Mandelbrot, é um ramo da Matemática que está a permitir descrever alguns fenómenos naturais complexos tais como a turbulência, a linha

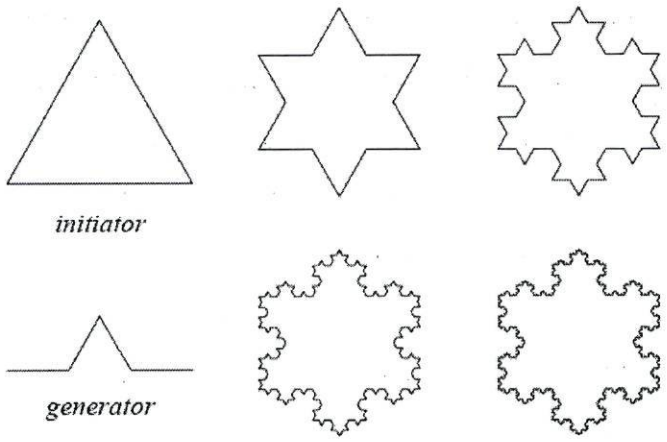

Figura 1.1: Construção do floco de neve de Koch [14, p. 2].

costeira de um país, as montanhas, as galáxias ou as nuvens [10]. Um exemplo clássico de uma figura fractal definida por um mecanismo de reescrita é o floco de neve de Koch. Segundo Mandelbrot [9], a construção deste fractal começa com duas formas, o iniciador ("*initiator*") e o gerador ("*generator*"). Depois, em cada iteração, substitui-se cada um dos segmentos de recta por uma cópia reduzida do gerador, a qual é disposta de modo a que as suas extremidades coincidam com as do segmento substituído (Figura 1.1). O floco de neve de Koch é o limite para o qual tende esta construção. Mesmo utilizando os instrumentos de desenho mais sofisticados não é possível obter muitos termos deste tipo de sucessões de imagens. Pelo contrário, utilizando um computador, este processo de gerar novas imagens pode continuar quase indefinidamente, obtendo-se figuras com pormenores invisíveis a olho nu mas que podem ser visualizados graças à sua crescente capacidade de ampliação. A evolução dos computadores desempenha assim um papel fundamental no desenvolvimento deste ramo da Matemática. A seguinte frase proferida por Mandelbrot no encerramento do congresso *Fractal '90*, em Lisboa, permite ilustrar na perfeição esta situação: «*Hoje, qualquer pessoa pode fazer em minutos, com um PC, o que há 11 anos me custou sangue, suor e lágrimas...*» [7].

Os mecanismos de reescrita mais estudados e melhor compreendidos são aqueles que operam com cadeias de caracteres, designadas aqui por palavras. Segundo Prusinkiewicz

e Lindenmayer [14, p. 2–3], a primeira definição formal deste tipo de mecanismos foi apresentada por Thue no início do século passado, no entanto, somente no final dos anos 50, com o trabalho de Chomsky sobre gramáticas formais, é que se gerou um grande interesse pela reescrita de palavras. Os anos 60 foram um período de enorme fascínio pela sintaxe, pelas gramáticas e pelas suas aplicações à Ciência de Computadores. Este entusiasmo deu origem a linguagens clássicas de programação como o ALGOL-60 (designação proveniente da expressão "*ALGOrithmic Language*"). Foi neste contexto que, em 1968, o biólogo Aristid Lindenmayer introduziu um novo tipo de mecanismo de reescrita de palavras, posteriormente designado por sistema de Lindenmayer (L-system). A diferença principal entre as gramáticas de Chomsky e este formalismo reside no processo de aplicação das regras de substituição. Nos L-systems as regras são aplicadas em paralelo, ocorrendo em cada iteração uma troca simultânea de todos os símbolos da palavra. Pelo contrário, nas gramáticas de Chomsky as regras são aplicadas sequencialmente, substituindo-se um caracter de cada vez [13]. É esta diferença que reflecte o interesse dos L-systems para a Biologia, uma vez que as divisões celulares ocorrem ao mesmo tempo em todas as partes de um ser vivo [16]. Os L-systems foram originalmente criados para modelar o desenvolvimento de organismos multicelulares simples (organismos filamentosos) [8]. Porém, a sua utilização foi mais tarde alargada a plantas maiores, com uma estrutura de ramificação complexa [5]. Para além da sua adequação para modelar objectos naturais também têm sido utilizados noutras áreas, que incluem a geração de fractais ou a composição de partituras musicais [13].

As situações modeladas por L-systems são intrinsecamente dinâmicas, o que significa que a forma apresentada por um organismo é vista como sendo o resultado final de um processo de desenvolvimento [19]. Por sistema dinâmico entende-se um ambiente físico juntamente com um conjunto de regras que determinam como esse ambiente evolui ao longo do tempo. Quando o ambiente físico é uma cadeia de símbolos formais, estes sistemas podem ser designados por sistemas dinâmicos simbólicos. Os L-systems são assim um exemplo deste tipo de sistemas dinâmicos [21].

Nos L-systems, os organismos são considerados como sendo uma congregação de partes distintas, designadas por módulos. Cada um desses módulos é representado por um símbolo de um alfabeto, e são usados símbolos diferentes para módulos de diferentes tipos ou em diferentes estados. Para além disto, pode juntar-se ao símbolo um ou mais parâmetros numéricos que, em conjunto, caracterizam o estado do módulo. Assim sendo, uma sequência destes módulos forma uma palavra que representa o organismo modelado num determinado momento do seu desenvolvimento [19].

1.2 Organização da Dissertação

A presente dissertação encontra-se organizada em cinco capítulos, ao longo dos quais se procurará expor a teoria dos sistemas de Lindenmayer e implementar a modelação (não realista) de árvores tridimensionais, recorrendo ao *Maple*.

No primeiro capítulo é feita uma breve contextualização dos sistemas de Lindenmayer e são dadas informações acerca da organização da dissertação.

Nos capítulos 2 e 3 é apresentada a fundamentação teórica associada aos sistemas de Lindenmayer. Esta fundamentação teórica foi baseada fundamentalmente no livro *The Algorithmic Beauty of Plants*, de Przemyslaw Prusinkiewicz e Aristid Lindenmayer, publicado em 1990. No segundo capítulo podem encontrar-se algumas definições que permitem descrever a classe mais simples deste formalismo e as respectivas operações. Também pode ser encontrada uma interpretação gráfica de palavras baseada na conhecida "*turtle geometry*" [1] e são ainda explicitados dois métodos de operar com este tipo de mecanismo de reescrita. O Capítulo 3 foi reservado para a modelação de árvores em abstracto baseada na evolução de sistemas de Lindenmayer. Neste capítulo é feita uma descrição matemática de estruturas com a forma de planta, e encontram-se alguns métodos e modelos que permitem gerar essas estruturas.

A compreensão da fundamentação teórica exposta nos dois capítulos anteriores conduziu

à modelação em concreto de vários objectos recorrendo à linguagem *Maple*. O Capítulo 4 inicia-se com algumas noções básicas desta linguagem, de modo a que os procedimentos usados na dissertação possam ser compreendidos mais facilmente. Esta breve exposição foi elaborada partindo do pressuposto que o leitor já está minimamente familiarizado com o *Maple*. O capítulo termina com a apresentação dos referidos procedimentos. Estes procedimentos permitiram criar a maioria das imagens presentes nesta dissertação, começando pelos fractais, passando pelas estruturas ramificadas bidimensionais com a forma de planta e terminando nas árvores tridimensionais.

Por fim, o Capítulo 5 apresenta a conclusão de todo o trabalho desenvolvido e algumas orientações para futura investigação.

Capítulo 2

Sistemas de Lindenmayer

Neste capítulo serão apresentadas algumas definições que permitem descrever a classe mais simples deste formalismo e as respectivas operações. Será também apresentada uma interpretação gráfica de palavras baseada na "*turtle geometry*" e explicitados dois métodos de operar com este tipo de mecanismo de reescrita. Por fim, será ainda abordada a extensão desta teoria à modelação em três dimensões.

2.1 D0L-systems

Os "D0L-systems" são a classe mais simples de L-systems, que inclui aqueles que são determinísticos e livres de contexto. O "D" significa que são determinísticos e o "0" está relacionado com o facto de serem livres de contexto, resultando assim prefixo "D0". Antes de prosseguir é conveniente esclarecer o significado destes dois novos conceitos. Um L-system diz-se determinístico quando existe exactamente uma regra de substituição para cada símbolo do alfabeto. Por sua vez, é considerado livre de contexto quando as regras são aplicadas sem que se tenha em consideração o contexto no qual os caracteres surgem, isto é, as regras têm apenas em atenção o símbolo ao qual são aplicadas, ignorando os

símbolos adjacentes.

Vão-se agora apresentar algumas definições que permitem descrever os D0L-systems e as suas operações. Considere-se que V denota um conjunto finito de símbolos formais, usualmente constituído por letras mas também por outros caracteres, designado por alfabeto. Por V^* denota-se o conjunto de todas as palavras construídas por justaposição de símbolos de V, incluindo a palavra vazia, e por V^+ representa-se o conjunto de todas as palavras não vazias em V. Um D0L-system é então um terno ordenado $G = \langle V, w, P \rangle$, onde

- V diz-se o alfabeto do sistema,
- $w \in V^+$ designa-se por axioma,
- $P \subset V \times V^*$ é um conjunto finito dito de regras de substituição.

O axioma é a palavra inicial a partir da qual todo o sistema evolui. As regras $(a, \chi) \in P$ representam-se por $a \rightarrow \chi$, onde a letra a e a palavra χ se designam por antecessor e sucessor, respectivamente. Atendendo à definição apresentada para o conjunto de regras de substituição, pode concluir-se que o antecessor é formado por um único elemento do alfabeto do sistema, enquanto que o sucessor pode ser uma palavra com zero, um ou mais símbolos desse mesmo alfabeto [19]. Todas as letras do alfabeto têm uma única regra que as transforma numa palavra de V^*, o que significa que se não existir qualquer regra especificamente definida para um dado elemento $a \in V$ considera-se que a regra identidade $a \rightarrow a$ pertence ao conjunto das regras de substituição P. Neste último caso, essa letra diz-se uma constante do L-system. Numa única iteração, cada letra de uma palavra é substituída pelo respectivo sucessor, de acordo com as regras de substituição definidas. O desenvolvimento é simulado através de uma sequência destas iterações, começando o processo de reescrita no axioma w. Prusinkiewicz e Lindenmayer [14, p. 4] definem uma sucessão de desenvolvimento de G como sendo uma sucessão $(g_n), n = 0, 1, 2, 3, \ldots$, onde cada geração g_n resulta da geração anterior, g_{n-1}, por aplicação em simultâneo das regras de substituição a cada símbolo de g_{n-1}. A geração inicial g_0 corresponde ao axioma w.

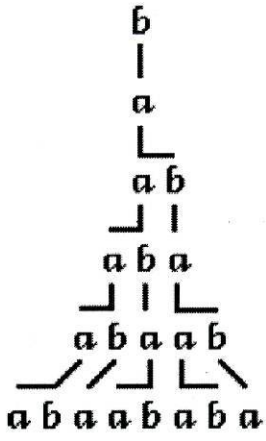

Figura 2.1: Algumas gerações de um D0L-system [14, p. 4].

Definem também que uma palavra $\nu = \chi_1\chi_2 \ldots \chi_m \in V^*$ deriva directamente de uma outra palavra $\mu = a_1a_2 \ldots a_m$ formada por elementos de V, e escreve-se $\mu \Rightarrow \nu$, se e só se $a_i \rightarrow \chi_i$ para todo o $i = 1, 2, \ldots, m$. Tendo em conta as definições anteriores pode ainda dizer-se que uma determinada palavra ν é gerada por G numa evolução de ordem n se existir uma sucessão de desenvolvimento $g_0, g_1, \ldots, g_n$ tal que $g_0 = w, g_n = \nu$ e $g_0 \Rightarrow g_1 \Rightarrow \ldots \Rightarrow g_n$.

O exemplo seguinte permite ilustrar de um modo muito simples como funcionam os D0L-systems. Considerem-se as palavras formadas pelas letras a e b, letras essas que podem surgir numerosas vezes na mesma palavra. Considere-se também que a cada uma das letras está associada uma regra de substituição. A regra $a \rightarrow ab$ significa que a letra a será substituída pela palavra ab, e a regra $b \rightarrow a$ significa que a letra b será substituída pela palavra a. Considere-se ainda que a palavra formada pela letra b é o axioma. O processo de reescrita começa com a substituição do axioma pela palavra a, através da regra $b \rightarrow a$. Na segunda iteração, a palavra a é transformada em ab, através da regra $a \rightarrow ab$. Na iteração seguinte, cada uma das letras da palavra ab é substituída simultaneamente usando as regras de substituição definidas. Assim sendo, após esta iteração resulta a palavra aba. De um modo análogo, esta palavra originará $abaab$, que por sua vez produzirá a palavra $abaababa$, e assim sucessivamente (Figura 2.1). Este exemplo é chamado de "*Fibonacci L-system*". A designação resulta da sua relação com a conhecida sucessão de Fibonacci, cujos termos

correspondem ao comprimento de cada uma das palavras (número de símbolos de cada palavra) produzidas nas primeiras iterações deste L-system [21]. Esta sucessão numérica que surge em diversos fenómenos naturais foi apresentada por Leonardo de Pisa, mais conhecido por Fibonacci, no mais famoso dos problemas incluídos no seu livro *Liber Abaci* (1202): «*Quantos casais de coelhos serão produzidos num ano, começando com um só casal, se, em cada mês, cada casal gerar um novo casal que se torna produtivo a partir do segundo mês?*». Este problema pressupõe que não ocorrem mortes [3]. Para melhor se perceber a relação supracitada, considere-se que as letras a e b são os dois estados possíveis que um casal de coelhos pode tomar. Após cada iteração, um casal imaturo b torna-se num casal adulto a, ficando apto a dar origem a um novo casal imaturo em cada uma das iterações seguintes. Este sistema dinâmico simbólico traduz deste modo um tipo muito simples de dinâmica populacional [21].

Dada a natureza discreta dos L-systems, é de notar que o crescimento contínuo dos organismos entre iterações não é possível ser representado recorrendo a esta teoria.

2.2 Interpretação Gráfica de Palavras

Como já foi referido, os L-systems foram concebidos como uma teoria matemática para modelar o desenvolvimento de organismos multicelulares simples. Numa fase inicial, os aspectos geométricos estavam para além do alcance desta teoria, o que significa que os L-systems não possuíam o detalhe necessário para a modelação de plantas com uma estrutura complexa. Posteriormente, com vista a torná-los numa ferramenta de modelação de plantas mais versátil, foram sendo propostas várias interpretações geométricas. Nos próximos parágrafos ir-se-á explicitar resumidamente essa evolução.

Inicialmente, as situações modeladas com L-systems eram ilustradas listando as sucessivas palavras geradas ao longo do processo iterativo. Para se obter uma visualização adicional era necessária a interpretação humana das letras como sendo módulos diferentes, através

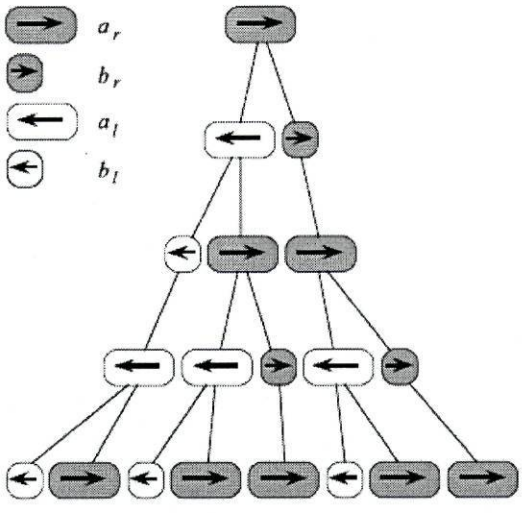

Figura 2.2: Desenvolvimento de um filamento da *Anabaena catenula* [14, p. 5].

da imaginação ou através de desenhos manuais [6, p. 29]. Na Figura 2.2 pode ver-se um exemplo simples desta abordagem aplicado à criação de uma imagem esquemática da *Anabaena catenula*. As letras a e b, que representam o estado das células (o seu tamanho e a sua prontidão para se dividirem), são representadas graficamente como rectângulos curtos ou longos com os cantos arredondados. Os índices l ("*left*") e r ("*rigth*") indicam a polaridade das células, especificando a posição onde as células filhas vão ser criadas. A situação representada na Figura 2.2 é modelada pelo seguinte L-system:

$$V = \{a_r,\ a_l,\ b_r,\ b_l\}$$

$$w : a_r$$

$$p_1 : a_r \rightarrow a_l b_r$$

$$p_2 : a_l \rightarrow b_l a_r$$

$$p_3 : b_r \rightarrow a_r$$

$$p_4 : b_l \rightarrow a_l$$

As estruturas geradas constituem cadeias unidimensionais de rectângulos, reproduzindo assim a sequência de símbolos das palavras correspondentes. Contudo, não existe nada no modelo apresentado que force a que os filamentos sejam rectilíneos, pelo que se pode fazer uma representação curvilínea dos mesmos, aproximando-a mais das imagens reais destes

organismos microscópicos [6, p. 29].

De acordo com Prusinkiewicz e Lindenmayer [14, p. 6], foram publicados em 1974, por Frijters e Lindenmayer e por Hogeweg e Hesper, os primeiros resultados no sentido de apresentar uma interpretação gráfica mais sofisticada que possibilitasse a modelação de plantas com uma estrutura complexa. Em 1979 foi apresentada, por Szilard e Quinton, uma nova abordagem à interpretação dos L-systems. Nessa proposta, cada uma das letras do alfabeto do L-system tem atribuída uma interpretação que serve de comando para uma "*plotter pen*". Eles mostraram que curvas complexas hoje conhecidas como fractais podiam ser geradas através de D0L-systems extremamente simples. Estes resultados constituíram a base de outros estudos em diferentes sentidos. Siromoney and Subramanian especificaram L-systems que geram algumas das curvas de preenchimento do espaço clássicas. Dekking investigou as propriedades de curvas geradas por L-systems, principalmente o problema de determinar a dimensão fractal (Hausdorff) do conjunto limite. Hanan [6, p. 30] refere que Prusinkiewicz, ao constatar que a abordagem feita por Szilard e Quinton era semelhante à "*LOGO turtle geometry*" (o termo *LOGO* foi escolhido como referência à sua significação grega: pensamento, raciocínio, discurso), incorporou os comandos geométricos directamente nos L-systems. No conceito original de *LOGO turtle geometry*, as imagens correspondem ao "rasto" de uma "tartaruga gráfica" que se desloca sobre uma superfície bidimensional, à medida que vai obedecendo aos comandos do utilizador. Isto acabou por ser o sistema ideal para dar uma interpretação geométrica à dinâmica dos L-systems.

Segundo Prusinkiewicz *et al* [18], quando se utiliza um L-system podem considerar-se duas fases distintas: a fase de geração e a fase de interpretação. Na fase de geração as regras de substituição são aplicadas numa sequência de iterações, formando uma palavra final designada por "*L-string*". Depois, na fase de interpretação, a L-string é convertida numa representação geométrica. Ao longo desta dissertação será usada uma interpretação gráfica baseada na *turtle geometry*.

Após uma L-string ser gerada passa-se, como já foi referido, à fase de interpretação. A

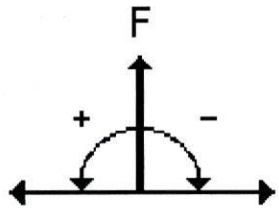

Figura 2.3: Interpretação gráfica dos símbolos F, $+$ e $-$ ($\delta = 90°$) [14, p. 7].

L-string é então analisada sequencialmente, da esquerda para a direita, sendo os símbolos interpretados como comandos que vão dirigir a "tartaruga" [16]. Esta "tartaruga" é representada pelo seu estado, o qual é definido pelo terno ordenado (x, y, α). As coordenadas cartesianas (x, y) representam a posição da "tartaruga" no plano, enquanto que o sentido para onde a "tartaruga" está virada é representado por $\alpha \in \mathbb{R}$, correspondente ao ângulo formado com o semi-eixo positivo Ox. Definindo à priori o avanço $d \in \mathbb{R}^+$ e o incremento angular $\delta \in \mathbb{R}^+$, então a "tartaruga" responde aos comandos representados pelos seguintes símbolos (Figura 2.3):

F : Avançar na direcção actual um passo de comprimento d. O estado da "tartaruga" muda para (x', y', α), onde $x' = x + d.\cos\alpha$ e $y' = y + d.\sin\alpha$. É desenhado um segmento de recta entre os pontos (x, y) e (x', y');

f : Avançar na direcção actual um passo de comprimento d. O estado da "tartaruga" muda para (x', y', α), onde $x' = x + d.\cos\alpha$ e $y' = y + d.\sin\alpha$. Não é desenhado qualquer segmento entre os pontos (x, y) e (x', y');

$+$: Fazer uma rotação de δ graus no sentido positivo (anti-horário). O estado da "tartaruga" muda para $(x, y, \alpha + \delta)$;

$-$: Fazer uma rotação de δ graus no sentido negativo (horário). O estado da "tartaruga" muda para $(x, y, \alpha - \delta)$.

Dada uma L-string ν, o estado inicial (x_0, y_0, α_0) e os parâmetros fixos d e δ, a interpretação gráfica de ν é a figura (conjunto de segmentos de recta) correspondente ao rasto deixado

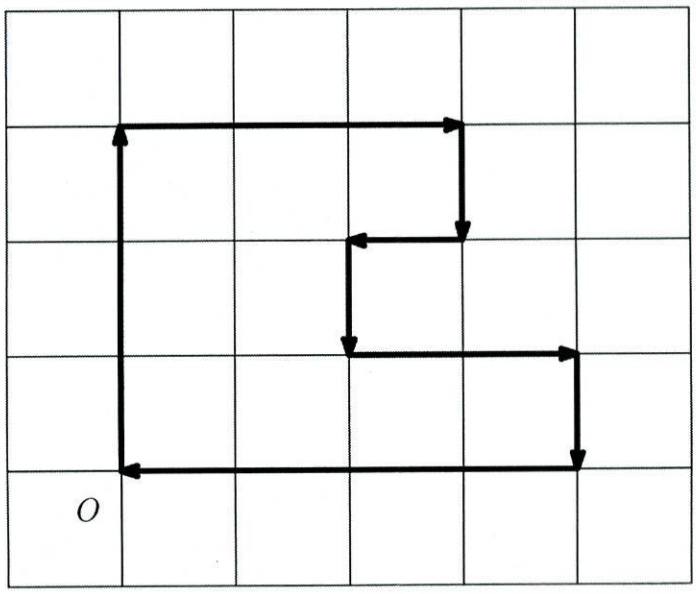

Figura 2.4: Interpretação gráfica da L-string "$FFF-FFF-F-F+F+FF-F-FFFF$".

pelos movimentos da "tartaruga" como resposta às instruções codificadas na palavra ν. Na Figura 2.4 pode ver-se a interpretação de uma palavra, onde $O = (x_0, y_0)$ e $\alpha = 90°$ definem o estado inicial da "tartaruga", $d = 1$ e $\delta = 90°$.

Vai-se agora formalizar a noção de interpretação gráfica de uma palavra de V^* como um subconjunto do plano. Sendo uma imagem no plano um conjunto de pontos desse plano, então uma função $I : V^* \to \mathcal{P}(\mathbb{R}^2)$, que transforma o conjunto das palavras formadas por justaposição de letras do alfabeto V, incluindo a palavra vazia, no conjunto das imagens no plano, designa-se por "função de interpretação gráfica". Considere-se o estado inicial (x_0, y_0, α_0) e os parâmetros fixos d e δ, e seja $\nu = a_1 a_2 \ldots a_m \in V^*$ uma palavra a interpretar. Para a palavra de comprimento zero (palavra vazia) define-se que $I(\nu)$ é o ponto de coordenadas (x_0, y_0) e que a "tartaruga" mantém o seu estado inicial. Vai-se seguidamente supor que para palavras ν de comprimento m está definida correctamente a imagem $I(\nu)$ de uma função de interpretação gráfica e o estado final da "tartaruga" $s(\nu) = (x_\nu, y_\nu, \alpha_\nu)$. Seja $\nu' = a_1 a_2 \ldots a_m a_{m+1} = \nu a_{m+1} \in V^*$ uma palavra de comprimento

$m+1$. Então, se

- $a_{m+1} = F$, define-se: $s(\nu') = (x_{\nu'}, y_{\nu'}, \alpha_{\nu'})$, onde $\alpha_{\nu'} = \alpha_\nu$ e $(x_{\nu'}, y_{\nu'}) = (x_\nu, y_\nu) + d.(\cos(\alpha_{\nu'}), \sin(\alpha_{\nu'}))$; Neste caso, $I(\nu')$ é igual a $I(\nu)$ reunido com o segmento de recta que une (x_ν, y_ν) a $(x_{\nu'}, y_{\nu'})$.

- $a_{m+1} = f$, define-se: $s(\nu') = (x_{\nu'}, y_{\nu'}, \alpha_{\nu'})$, onde $\alpha_{\nu'} = \alpha_\nu$ e $(x_{\nu'}, y_{\nu'}) = (x_\nu, y_\nu) + d.(\cos(\alpha_{\nu'}), \sin(\alpha_{\nu'}))$; Neste caso, $I(\nu') = I(\nu)$.

- $a_{m+1} = +$, define-se: $s(\nu') = (x_{\nu'}, y_{\nu'}, \alpha_{\nu'})$, onde $\alpha_{\nu'} = \alpha_\nu + \delta$, $x_{\nu'} = x_\nu$ e $y_{\nu'} = y_\nu$; Neste caso, $I(\nu') = I(\nu)$.

- $a_{m+1} = -$, define-se: $s(\nu') = (x_{\nu'}, y_{\nu'}, \alpha_{\nu'})$, onde $\alpha_{\nu'} = \alpha_\nu - \delta$, $x_{\nu'} = x_\nu$ e $y_{\nu'} = y_\nu$; Neste caso, $I(\nu') = I(\nu)$.

- Se a_{m+1} não for igual a algum dos símbolos anteriores então $I(\nu') = I(\nu)$ e $s(\nu') = s(\nu)$. Ou seja, a "tartaruga" ignora o símbolo e conserva o seu estado.

Dando o estado inicial, o avanço d e o incremento angular δ, tem-se deste modo definida indutivamente uma função de interpretação gráfica para todas as palavras de comprimento finito pertencentes a V^*, o que possibilita interpretar as palavras geradas por um qualquer L-system.

Nas secções seguintes podem encontrar-se mais alguns exemplos de figuras geradas por D0L-systems tendo por base a interpretação gráfica de palavras acabada de apresentar.

2.2.1 Cantor L-system

Um dos exemplos mais simples corresponde à representação geométrica do conjunto de Cantor (Figura 2.5). O L-system que permite gerar este conjunto tem as seguintes com-

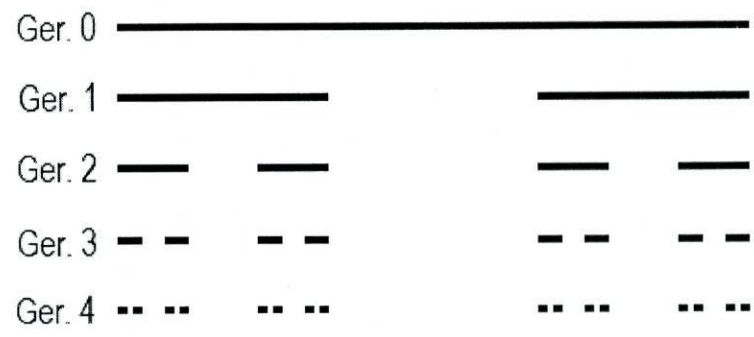

Figura 2.5: Algumas gerações do Cantor L-system [21].

ponentes:

$$V = \{F,\ f\}$$
$$w : F$$
$$p_1 : F \rightarrow FfF$$
$$p_2 : f \rightarrow fff$$

Considere-se que o estado inicial é definido por $(x_0, y_0, \alpha_0) = (0, 0, 0°)$ e que $d = 1$. Neste caso particular, o incremento angular é irrelevante uma vez que no axioma e nos sucessores não intervêm símbolos que forcem uma actualização do sentido da "tartaruga". Pode constatar-se que a representação geométrica do axioma é um segmento de recta horizontal e que, em cada iteração, se vai removendo o terço do meio de cada um dos segmentos. Note-se que houve a necessidade de acrescentar uma segunda regra com o objectivo de manter as distâncias apropriadas entre as várias componentes da figura [21]. Note-se ainda que, à excepção do axioma, todas as gerações representadas foram alvo de um redimensionamento, uma vez que as posições inicial e final se mantêm ao longo das diversas gerações apesar de cada segmento (visível ou não) ter comprimento 1. O mesmo efeito poderia ser alcançado se se permitisse que o avanço da "tartaruga" decrescesse, de uma geração para a seguinte, segundo uma progressão geométrica de razão 1/3.

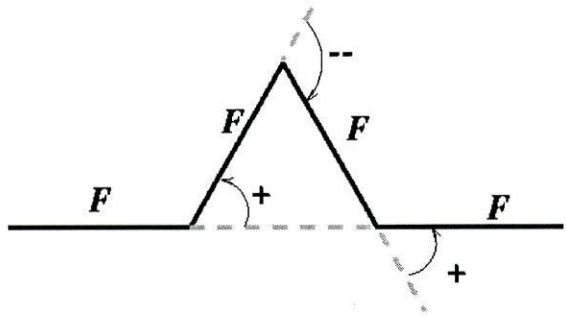

Figura 2.6: Interpretação gráfica do sucessor da regra de substituição que permite gerar a curva de Koch [21].

2.2.2 Floco de Neve de Koch

As componentes do L-system que permite gerar a curva de Koch são as seguintes:

$$V = \{F,\ +,\ -\}$$
$$w : F$$
$$p : F \rightarrow F + F - -F + F$$

Considere-se que o estado inicial é dado por $(x_0, y_0, \alpha_0) = (0, 0, 0°)$ e que $d = 1$ e $\delta = 60°$. Geometricamente, a regra definida significa que um segmento de recta F é substituído pela disposição de quatro segmentos de recta que se pode observar na Figura 2.6. Se se ampliar para o triplo o segmento F, pode descrever-se esta regra dizendo que o terço do meio deste novo segmento de recta ampliado é substituído pelos outros dois lados de um triângulo equilátero construído sobre essa porção do segmento. A Figura 2.6 coincide com a geração 1 e na Figura 2.7 podem ver-se mais algumas gerações da curva de Koch.

O floco de neve de Koch (Figura 2.8) obtém-se de um modo análogo à curva de Koch, residindo apenas no axioma a diferença entre ambos. A construção do floco de neve de Koch inicia-se com um triângulo equilátero ($w : F - -F - -F$), em vez de começar com um segmento de recta.

Estes exemplos permitem mostrar a relação existente entre as construções de Koch, apresentadas na Introdução, e os L-systems. O iniciador corresponde ao axioma e o gerador

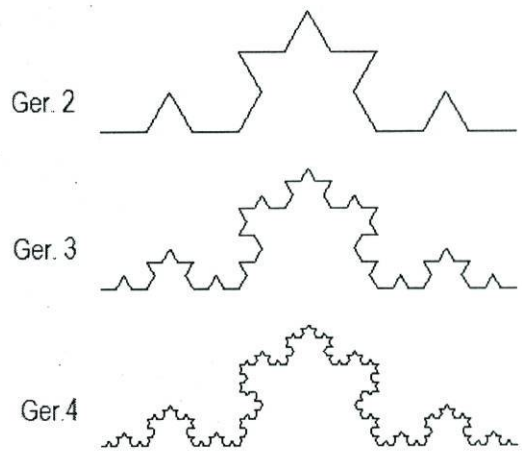

Figura 2.7: Algumas gerações da curva de Koch [21].

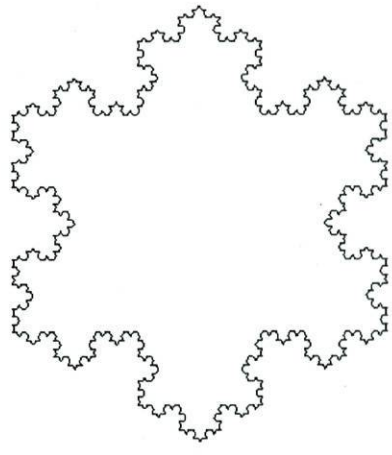

Figura 2.8: Geração quatro do floco de neve de Koch.

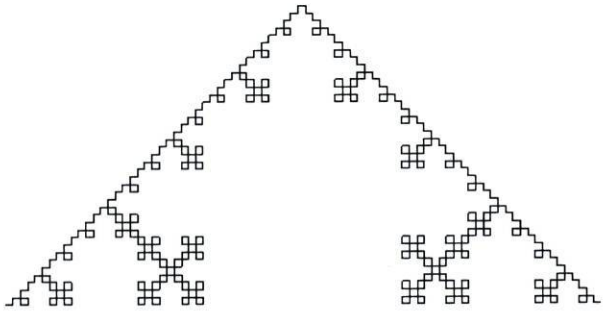

Figura 2.9: Modificação quadrática da curva de Koch.

corresponde ao sucessor da regra de substituição.

2.2.3 Outros Exemplos de Curvas de Koch

Nesta secção são apresentados mais alguns exemplos de curvas de Koch geradas por L-systems. Em todos eles é utilizado o mesmo estado inicial que nos exemplos da secção anterior e os parâmetros fixos são $d = 1$ e $\delta = 90°$.

A Figura 2.9 é o resultado de uma evolução de ordem 4 do L-system cujo axioma é F e que possui a regra $F \rightarrow F + F - F - F + F$. Por sua vez, a Figura 2.10 resulta de uma evolução de ordem 2 do L-system com axioma $F + F + F + F$ e regras

$$p_1 : F \rightarrow F + f - FF + F + FF + Ff + FF - f + FF - F - FF - Ff - FFF$$
$$p_2 : f \rightarrow ffffff$$

Os L-systems constituem uma ferramenta adequada para desenvolver novas curvas de Koch, uma vez que é extremamente fácil fazer ligeiras modificações nas regras de substituição e verificar o resultado obtido. Por exemplo, começando com uma evolução de ordem 3 do L-system definido pelo axioma $F-F-F-F$ e pela regra $F \rightarrow F-F+F+FF-F-F+F$ (Figura 2.11) podem observar-se, nas figuras seguintes (Figuras 2.12 a 2.15), as variações resultantes da introdução, da eliminação ou da substituição de alguns símbolos. Note-se que a última destas figuras resultou de uma evolução de ordem 4.

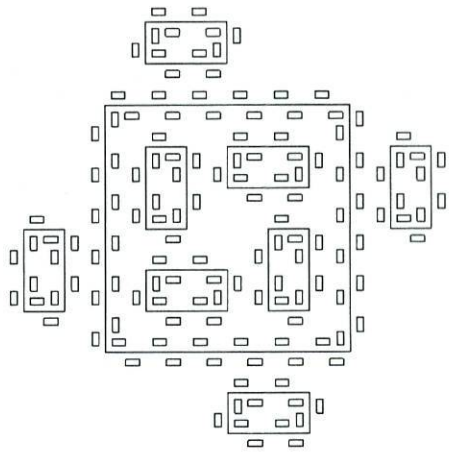

Figura 2.10: Combinação de ilhas e lagos.

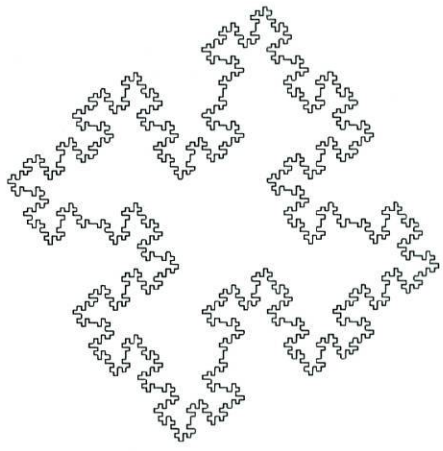

Figura 2.11: Ilha de Koch quadrática.

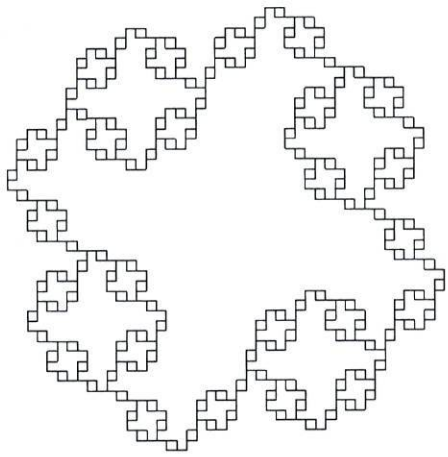

Figura 2.12: Imagem gerada por um L-system com a regra $F \rightarrow FF-F-F-F-F-F+F$.

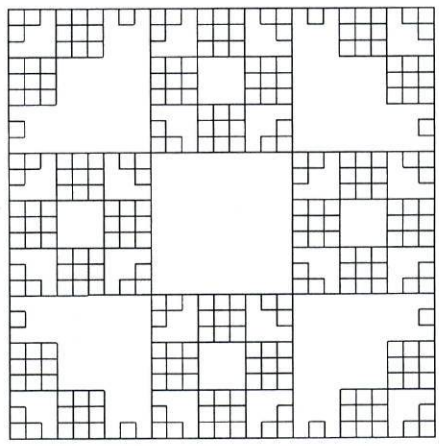

Figura 2.13: Imagem gerada por um L-system com a regra $F \rightarrow FF-F-F-F-FF$.

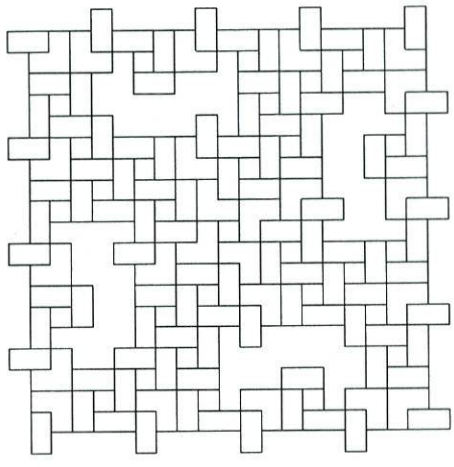

Figura 2.14: Imagem gerada por um L-system com a regra $F \rightarrow FF - F + F - F - FF$.

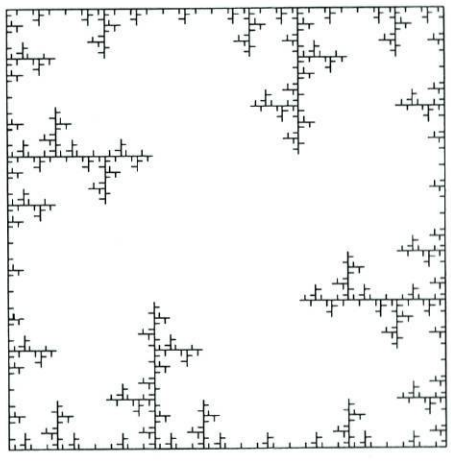

Figura 2.15: Imagem gerada por um L-system com a regra $F \rightarrow FF - F - -F - F$.

2.3 Modos de Operar com D0L-systems

A modificação das regras de substituição permite perceber um pouco a relação existente entre os L-systems e as figuras por eles geradas. Contudo, esta abordagem torna-se insuficiente, uma vez que os L-systems são frequentemente usados para modelar processos de desenvolvimento, sendo necessário que capturem a essência desses processos. A construção de um L-system que modele um processo de desenvolvimento designa-se por problema de inferência. Existem dois modos distintos de operar com D0L-systems usando a interpretação gráfica de palavras apresentada. Num desses modos as regras de substituição operam com arestas, enquanto que no outro operam com vértices. A explicitação destes modos será feita usando curvas abstractas, embora possam também ser aplicados a estruturas ramificadas encontradas nas plantas [14, p. 11].

2.3.1 Reescrita de Arestas

Nesta abordagem, que é vista como uma extensão das construções de Koch, podem ser considerados dois tipos de arestas, "esquerda" (F_l) e "direita" (F_r). Os símbolos F_l e F_r representam arestas que resultam da resposta da "tartaruga" ao comando "avançar na direcção actual um passo de comprimento d". Na Figura 2.16 pode ver-se a geração 4 de uma curva construída com base nestes dois tipos de arestas. As componentes do L-system que gerou essa curva são as seguintes:

$$
\begin{aligned}
&V : \{F_l,\ F_r,\ +,\ -\} \\
&w : F_r \\
&p_1 : F_l \rightarrow F_r + F_l + F_r \\
&p_2 : F_r \rightarrow F_l - F_r - F_l \\
&(\delta = 60°)
\end{aligned}
$$

Segundo Prusinkiewicz e Lindenmayer [14, p. 12], McKenna apresentou um algoritmo que permite construir curvas de preenchimento do espaço usando a reescrita de arestas.

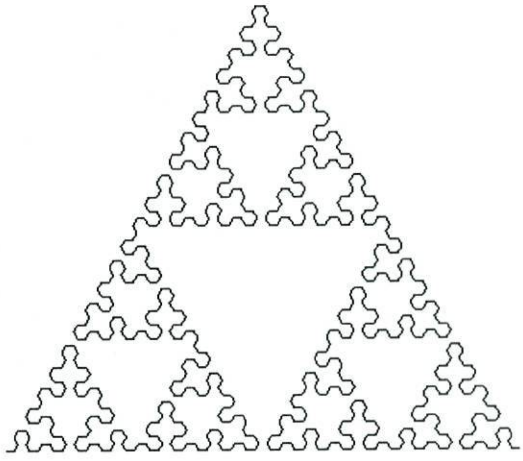

Figura 2.16: Curva gerada usando a reescrita de arestas: *Sierpinski gasket* [14, p. 11].

Através de uma breve explicação deste algoritmo pretende-se ilustrar este modo de operar com L-systems. Na Figura 2.17a e 2.17b pode ver-se uma aresta F_l e uma aresta F_r, respectivamente, com $\alpha = 0°$. As arestas distinguem-se pela direcção da marca. Mais especificamente, se um viajante imaginário percorrer uma aresta F_l encontrará um quadrado vazio à sua esquerda. Por sua vez, se o viajante percorrer uma aresta F_r verá o quadrado à sua direita. A aresta F_l é substituída por uma figura que preenche aproximadamente o quadrado à sua esquerda (Figura 2.17c). De um modo análogo, a aresta F_r é substituída por uma figura que preenche aproximadamente o quadrado à sua direita (Figura 2.17d). As duas regras de substituição que descrevem a situação apresentada na Figura 2.17 são:

- $P_1 : F_l \rightarrow +F_rF_r - F_l - F_l + F_r + F_rF_l + F_r - F_lF_l - F_r - F_l + F_rF_lF_l - F_r - F_lF_r + F_r + F_l - F_l - F_r + F_r + F_lF_l$

- $P_2 : F_r \rightarrow F_rF_r - F_l - F_l + F_r + F_r - F_l - F_lF_r + F_l + F_rF_rF_l - F_r + F_l + F_rF_r + F_l - F_rF_l - F_l - F_r + F_r + F_lF_l$

Na iteração seguinte, cada um dos 25 pequenos quadrados associados às curvas representadas na Figura 2.17c ou 2.17d será preenchido por uma cópia reduzida dessas curvas. Na Figura 2.18 pode ver-se um exemplo desse preenchimento. Neste caso, o axioma é F_l,

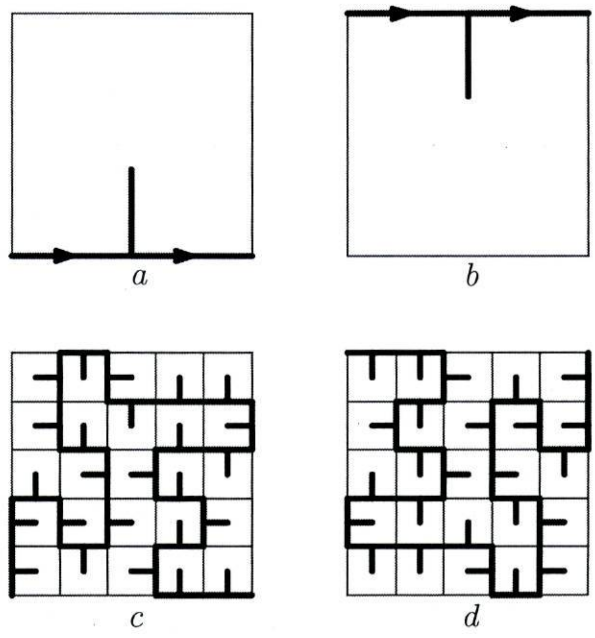

Figura 2.17: Construção da *E-curve* sobre uma grelha quadrada.

$\alpha = 0°$, $\delta = 90°$ e $d = 1$. Aplicando recursivamente este argumento, obtém-se uma curva que, no limite, irá preencher o quadrado.

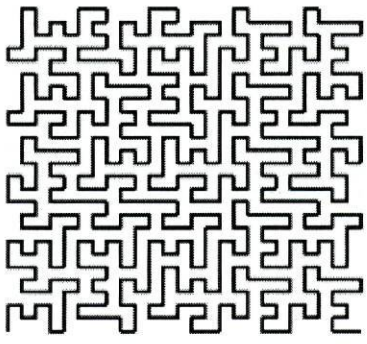

Figura 2.18: *E-curve* obtida usando a reescrita de arestas [14, p. 12].

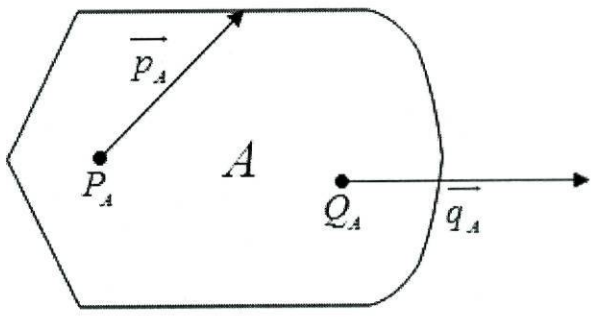

Figura 2.19: Descrição de uma subfigura A [14, p. 14].

2.3.2 Reescrita de Vértices

De acordo com Prusinkiewicz e Lindenmayer [14, p. 13], a ideia geral desta abordagem consiste na substituição dos vértices da figura obtida na iteração anterior por novas figuras. De modo a realizar esta tarefa, surge a necessidade de introduzir novos símbolos que representam subfiguras. Na Figura 2.19 está representada uma determinada subfigura A pertencente ao conjunto de todas as subfiguras $\mathcal{A}$. Cada uma delas apresenta:

- dois pontos de contacto, designados por *ponto de entrada* P_A e por *ponto de saída* Q_A;
- dois vectores de direcção, chamados de *vector de entrada* $\vec{p}_A$ e de *vector de saída* $\vec{q}_A$.

Durante a fase de interpretação, a "tartaruga" ao encontrar o símbolo A numa L-string ν incorpora a correspondente subfigura na imagem. No processo de incorporação, a subfigura A é sujeita a uma translação e a uma rotação de modo a alinhar o seu ponto de entrada P_A e a sua direcção $\vec{p}_A$ com a posição e orientação da "tartaruga" naquele momento. Após a incorporação da referida subfigura, a "tartaruga" irá ficar com a posição e orientação correspondentes ao ponto de saída Q_A e direcção $\vec{q}_A$.

A curva de Hilbert é outro exemplo de uma curva de preenchimento do espaço. Através deste exemplo pretende-se ilustrar o funcionamento de um L-system com reescrita de

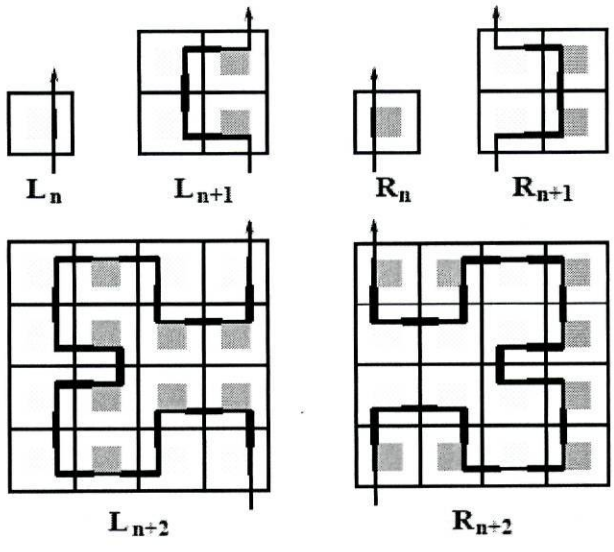

Figura 2.20: Construção recursiva da curva de Hilbert usando a reescrita de vértices [14, p. 15].

vértices. O processo de construção que é apresentado na Figura 2.20 pode ser generalizado para outras curvas da mesma classe. Nessa figura podem observar-se os pontos de contacto e as direcções das subfiguras L_n e R_n. Note-se que como $\vec{p}_A = \vec{q}_A$ então estes dois vectores estão representados graficamente por um único vector ($\uparrow$). Cada uma dessas subfiguras está limitada por um pequeno quadrado imaginário, designado por "*frame*", cujos lados se encontram todos à mesma distância dos lados de uma peça quadrada ("*tile*") que contém esse *frame*. Cada um dos *frames* limita então uma curva aberta cujas extremidades coincidem com os dois vértices de contacto desse *frame*. Na mesma figura também se podem observar as figuras L_{n+1} e R_{n+1}. Estas figuras estão construídas numa matriz de 2×2 peças quadradas. Como se pode constatar, unindo os vértices de contacto dos *frames* vizinhos através de segmentos de recta horizontais ou verticais (segmentos a grosso) pode ser construída uma linha contínua que passa por todas as peças. A construção de uma curva de preenchimento do espaço resulta da repetição recursiva deste padrão de ligações, considerando-se a matriz de $m \times m$ peças quadradas como sendo um "*macrotile*" que contém uma curva aberta inscrita num "*macroframe*". Uma matriz de $m \times m$ *macrotiles* é seguidamente formada e são ligadas as curvas contidas nos seus *macroframes*. Este processo

de construção prosseguirá recursivamente com $m \times m$ *macrotiles* numa iteração a darem origem a um único *macrotile* maior na iteração seguinte.

As figuras L_{n+1} e R_{n+1} da Figura 2.20 podem ser descritas pelas seguintes fórmulas recursivas:

$$L_{n+1} = +R_nF - L_nFL_n - FR_n+$$
$$R_{n+1} = -L_nF + R_nFR_n + FL_n-$$

Note-se que $\alpha = \delta = 90°$. Supondo que as curvas L_0 e R_0 são dadas, pode obter-se L_n ou R_n, para $n > 0$, gerando-as recursivamente. Por exemplo, a determinação de R_2 processa-se do seguinte modo:

$$\begin{aligned} R_2 &= -L_1F + R_1FR_1 + FL_1- \\ &= -\left(+R_0F - L_0FL_0 - FR_0+\right)F + \left(-L_0F + R_0FR_0 + FL_0-\right) \\ &\quad F\left(-L_0F + R_0FR_0 + FL_0-\right) + F\left(+R_0F - L_0FL_0 - FR_0+\right) - \end{aligned}$$

Atendendo a que todos os índices presentes na palavra obtida são iguais, estes podem ser omitidos, desde que a contagem global das iterações seja mantida. Assim sendo, este processo de gerar a figura R_n, para $n > 0$, pode ser considerado como um mecanismo de reescrita de palavras. Esta figura pode ser obtida numa evolução de ordem n do seguinte L-system:

$$V : \{F,\ L,\ R,\ +,\ -\}$$
$$w : R$$
$$p_1 : L \rightarrow +RF - LFL - FR+$$
$$p_2 : R \rightarrow -LF + RFR + FL-$$

Na Figura 2.21 pode visualizar-se R_4. O estado inicial considerado foi $(x_0, y_0, \alpha_0) = (0, 0, 90°)$. Considerou-se ainda $d = 1$ e $\delta = 90°$. Note-se que os símbolos F, $+$ e $-$ são constantes deste L-system, uma vez que são substituídos por eles próprios. Para definir completamente R_n, para $n > 0$, é ainda necessário especificar as subfiguras representadas pelos símbolos L e R. No caso particular das "curvas puras", de que a curva de Hilbert é um exemplo, estas subfiguras reduzem-se a um único ponto. Atendendo a este facto, podem ocorrer duas situações. Na primeira dessas situações, na fase de geração, os símbolos L e

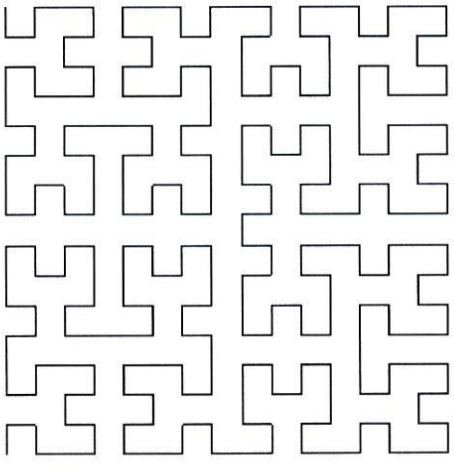

Figura 2.21: Curva gerada usando a reescrita de vértices: Curva de Hilbert.

R são substituídos pela palavra vazia no final do processo iterativo. Na outra situação, os símbolos L e R são mantidos na L-string final mas, durante a fase de interpretação, são ignorados pela "tartaruga". Qualquer uma das situações corresponde a um método que permite colapsar num ponto as subfiguras. Esta segunda abordagem à questão é consistente com a definição de interpretação gráfica apresentada, sendo por isso utilizada ao longo da presente dissertação.

2.3.3 Relação entre a Reescrita de Arestas e de Vértices

Existe uma relação entre as classes de curvas que podem ser geradas pelas técnicas de reescrita de arestas e de vértices, relação essa que será explicitada através do exemplo que se segue.

Considere-se, por exemplo, o L-system que permite gerar a "curva dragão" através do método da reescrita de arestas ($\delta = 90°$, $\alpha = 90°$):

$$w : F_l$$
$$p_1 : F_l \rightarrow F_l + F_r +$$
$$p_2 : F_r \rightarrow -F_l - F_r$$

Admita-se temporariamente que o antecessor de uma regra pode ser formado por mais do

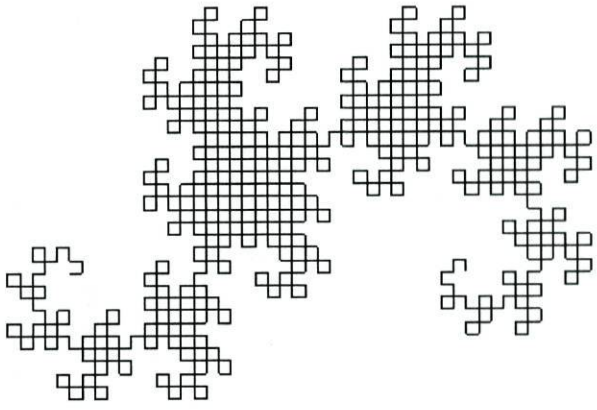

Figura 2.22: Geração 10 da curva dragão obtida usando a reescrita de vértices.

que um símbolo. Tem-se assim que o L-system anterior pode ser apresentado da seguinte forma:

$$w : Fl$$
$$p_1 : Fl \rightarrow Fl + rF+$$
$$p_2 : rF \rightarrow -Fl - rF$$

onde os símbolos l e r não são interpretados pela "tartaruga". Analisando as duas regras, pode constatar-se que na primeira o símbolo l é substituído pela palavra $l + rF+$, não sofrendo qualquer alteração o símbolo F inicial. Por sua vez, na segunda regra, verifica-se que o símbolo r é substituído pela palavra $-Fl - r$, ficando inalterado o símbolo F final. Pode então concluir-se que o L-system anterior pode ser convertido num outro, do seguinte modo:

$$w : Fl$$
$$p_1 : l \rightarrow l + rF+$$
$$p_2 : r \rightarrow -Fl - r$$

Esta última versão consiste num L-system com reescrita de vértices que permite gerar a curva dragão (Figura 2.22).

A imagem obtida por ambos os métodos será obviamente igual, pelo que, na prática, se escolhe aquele que seja mais conveniente. Nenhum dos métodos fornece um modo automático e geral de construir L-sytems que permitam modelar as mais variadas situações,

no entanto, a compreensão das diferenças existentes entre estas duas abordagens possibilita uma melhor percepção do modo de funcionamento dos L-systems, auxiliando assim a tarefa de modelar.

2.4 Modelação Tridimensional

Até este ponto apenas se considerou a modelação em duas dimensões, no entanto, a interpretação gráfica de palavras pode ser extendida de maneira natural a três dimensões. Para se interpretar uma L-string em três dimensões é necessário introduzir algumas alterações relativamente ao que já foi apresentado para o caso bidimensional. Tratando-se agora de uma "tartaruga" que se desloca no espaço, o seu estado é definido pela sua posição, representada em coordenadas cartesianas por (x, y, z), e pela sua orientação. A orientação da "tartaruga" é representada por um referencial ortonormado formado por três vectores, $\vec{H}$, $\vec{L}$ e $\vec{U}$, que indicam, respectivamente, o sentido para onde está virada (*"turtle's Heading"*), a sua esquerda (*"turtle's Left"*) e o sentido "cima" (*"turtle's Up"*). Estes três vectores satisfazem a equação $\vec{H} \times \vec{L} = \vec{U}$. Usando esta notação, as rotações da "tartaruga" são expressas pela fórmula

$$\left[\begin{array}{ccc} \vec{H}' & \vec{L}' & \vec{U}' \end{array} \right] = \left[\begin{array}{ccc} \vec{H} & \vec{L} & \vec{U} \end{array} \right] R$$

onde R é uma matriz de rotação 3×3. Esta matriz, que permite efectuar rotações de amplitude α, varia consoante o vector em torno do qual a rotação é feita [14, p. 19]. Assim sendo, as rotações de amplitude α em torno dos vectores $\vec{H}$, $\vec{L}$ e $\vec{U}$ são representadas pelas seguintes matrizes:

$$R_U(\alpha) = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

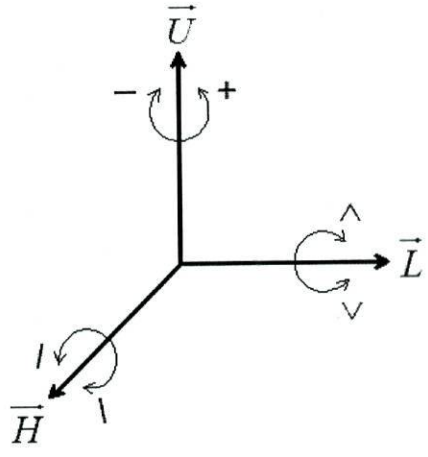

Figura 2.23: Comandos da "tartaruga" em três dimensões.

$$R_L(\alpha) = \begin{bmatrix} \cos\alpha & 0 & \sin\alpha \\ 0 & 1 & 0 \\ -\sin\alpha & 0 & \cos\alpha \end{bmatrix}$$

$$R_H(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha \\ 0 & -\sin\alpha & \cos\alpha \end{bmatrix}$$

Durante a fase de interpretação, à medida que a "tartaruga" analisa a L-string da esquerda para a direita, os quatro vectores que caracterizam o seu estado vão sendo modificados de acordo com as instruções codificadas nessa palavra [18]. Para mover a "tartaruga" no espaço são obviamente necessários mais comandos do que aqueles que foram apresentados para a mover no plano. Dando o avanço d e o incremento angular δ, então os seguintes símbolos permitem controlar o estado da "tartaruga" (Figura 2.23):

F : Avançar na direcção actual um passo de comprimento d. A posição da "tartaruga" muda para (x', y', z'), onde $x' = x + d.\vec{H}_x$, $y' = y + d.\vec{H}_y$ e $z' = z + d.\vec{H}_z$. É desenhado um segmento de recta entre os pontos (x, y, z) e (x', y', z');

f : Avançar na direcção actual um passo de comprimento d. A posição da "tartaruga"

muda para (x', y', z'), onde $x' = x + d.\vec{H}_x$, $y' = y + d.\vec{H}_y$ e $z' = z + d.\vec{H}_z$. Não é desenhado qualquer segmento entre os pontos (x, y, z) e (x', y', z');

$+$: Virar para a esquerda δ graus, usando a matriz de rotação $R_U(\delta)$;

$-$: Virar para a direita δ graus, usando a matriz de rotação $R_U(-\delta)$;

$\vee$: Virar para baixo δ graus, usando a matriz de rotação $R_L(\delta)$;

$\wedge$: Virar para cima δ graus, usando a matriz de rotação $R_L(-\delta)$;

$\backslash$: Rolar para a esquerda δ graus, usando a matriz de rotação $R_H(\delta)$;

$/$: Rolar para a direita δ graus, usando a matriz de rotação $R_H(-\delta)$;

$|$: Inverter o sentido, usando a matriz de rotação $R_U(180°)$.

Observe-se que os dois primeiros comandos apenas alteram o vector posição da "tartaruga" enquanto que os restantes apenas modificam a sua orientação.

A formalização da noção de interpretação gráfica de uma palavra de V^* é análoga ao caso bidimensional, pelo que se dispensa a sua apresentação.

Na Figura 2.24 pode observar-se uma imagem produzida após uma evolução de ordem 3 do L-system com axioma A, $\delta = 90°$ e com as seguintes regras de substituição:

$$p_1 : A \rightarrow B - F + CFC + F - D \vee F \wedge D - F + \vee \vee CFC + F + B//$$
$$p_2 : B \rightarrow A \vee F \wedge CFB \wedge F \wedge D \wedge \wedge - F - D \wedge |F \wedge B|FC \wedge F \wedge A//$$
$$p_3 : C \rightarrow |D \wedge |F \wedge B - F + C \wedge F \wedge A \vee \vee FA \vee F \wedge C + F + B \wedge F \wedge D//$$
$$p_4 : D \rightarrow |CFB - F + B|FA \vee F \wedge A \vee \vee FB - F + B|FC//$$

Note-se que a "tartaruga" estava inicialmente posicionada na origem e que a sua orientação inicial era dada pelos vectores $\vec{H} = (1,\ 0,\ 0)$, $\vec{L} = (0,\ 1,\ 0)$ e $\vec{U} = (0,\ 0,\ 1)$. Esta imagem, que consiste na extensão da curva de Hilbert em três dimensões, foi obtida através da reescrita de vértices, usando-se "macrocubos" em vez de "*macrotiles*". Na Figura

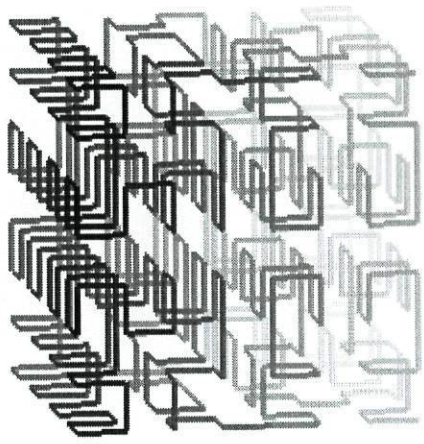

Figura 2.24: Curva de Hilbert em três dimensões.

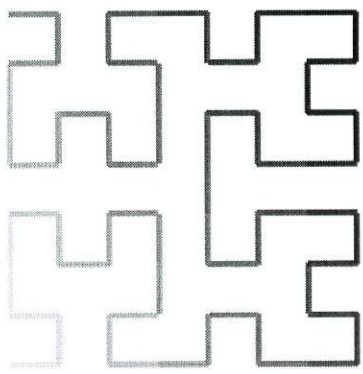

Figura 2.25: Uma perspectiva da curva de Hilbert em três dimensões.

2.25 pode ver-se uma perspectiva diferente da imagem produzida pelo L-system anterior. Comparando-a com a Figura 2.21, pode constatar-se a existência de uma relação evidente entre ambas. A Figura 2.25 poderia ser o resultado do processo de gerar R_3.

Capítulo 3

Modelação de Árvores

Os L-systems apresentados até este ponto apenas permitem modelar o desenvolvimento de organismos com uma estrutura linear, ou seja, o resultado da interpretação de uma qualquer L-string é uma única linha formada por vários segmentos de recta (visíveis ou não). Contudo, o mundo natural está repleto de organismos com uma estrutura ramificada, entre os quais se encontram as mais diversas árvores. Portanto, para simular o seu desenvolvimento é necessário que exista uma descrição matemática de estruturas com a forma de planta, para além de métodos e modelos para gerar essas estruturas. Essa descrição e alguns desses métodos e modelos podem ser encontrados ao longo deste capítulo.

3.1 L-systems Ramificados

Para modelar o desenvolvimento de estruturas ramificadas pode ser usado um mecanismo de reescrita que opere sobre árvores axiais. Por árvore axial entende-se uma árvore (grafo) cujas arestas são etiquetadas e têm um sentido definido, formando caminhos que se iniciam no vértice inicial, designado por raíz ou base, e que findam nos vértices terminais. Numa árvore axial, de cada um dos seus vértices parte, no máximo, um segmento em linha recta,

sendo os restantes (caso existam) designados por segmentos laterais. Uma sequência de segmentos é chamada de eixo se

- o primeiro segmento da sequência parte da raíz da árvore ou é um segmento lateral de um qualquer vértice,
- cada um dos segmentos seguintes é um segmento em linha recta, e
- o último segmento não é seguido por qualquer outro segmento em linha recta.

Um ramo é então formado por um eixo e por todos os segmentos que se originam a partir dele. Um ramo pode, por sua vez, ser considerado como uma (sub)árvore axial. Os eixos e os ramos caracterizam-se pela sua ordem. Um eixo que se origine na raíz da árvore tem ordem zero, enquanto que um eixo que se origine a partir de um segmento lateral de um eixo de ordem n tem ordem $n + 1$. Por sua vez, a ordem atribuída a um ramo é igual à ordem do seu eixo principal. Note-se que as arestas podem ser interpretadas como sendo segmentos de ramos de árvores abstractas e que, num caminho, um segmento seguido por pelo menos mais um segmento é designado por "*internode*", enquanto que um segmento terminal é chamado de "*apex*".

Tal como se pode observar na Figura 3.1, uma regra de substituição define que uma aresta (antecessor) é substituída por uma árvore axial (sucessor). No processo de substituição, o vértice inicial da aresta antecessora ("*Start*") corresponde à raíz da árvore axial sucessora ("*Base*") e o vértice final ("*End*") é identificado com o seu cimo ("*Top*"). Na Figura 3.2 pode ver-se a aplicação da regra representada na figura anterior a uma determinada aresta S de uma árvore axial.

Após a explicitação destes conceitos, apresentam-se seguidamente algumas definições que permitem descrever um "*tree 0L-system*" (T0L-system) e as suas operações. Um T0L-system G é um terno ordenado $G = \langle V, w, P \rangle$, onde

- V é um conjunto de etiquetas de arestas,

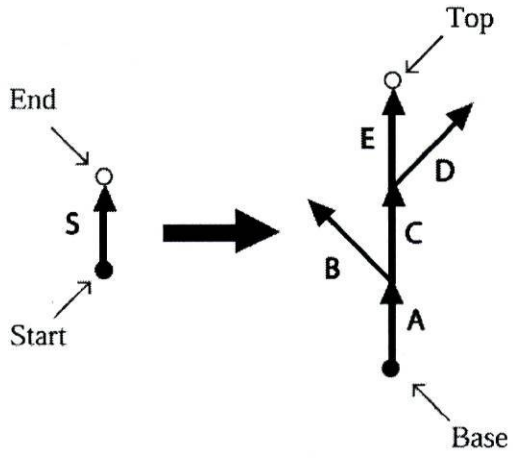

Figura 3.1: Interpretação geométrica de uma regra de substituição [14, p. 23].

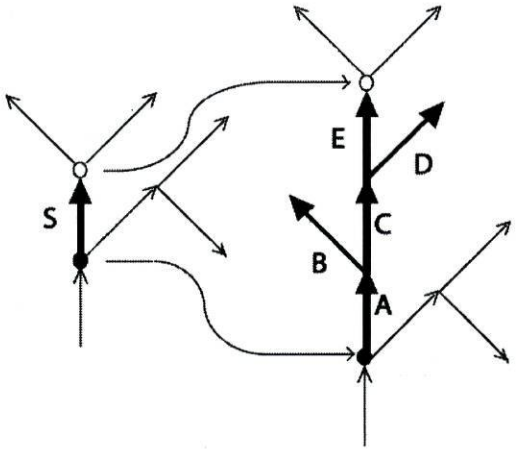

Figura 3.2: Aplicação de uma regra de substituição à aresta S de uma árvore [14, p. 23].

- w é uma árvore inicial com etiquetas de V,
- P é um conjunto finito dito de regras de substituição.

Dado um T0L-system G, diz-se que uma árvore axial T_2 deriva directamente de uma outra árvore T_1, e escreve-se $T_1 \Rightarrow T_2$, se T_2 se obtém de T_1 por substituição simultânea de todas as arestas de T_1, de acordo com o conjunto de regras P. Diz-se ainda que uma determinada árvore T é gerada por G numa evolução de ordem n se existir uma sucessão de desenvolvimento $T_0, T_1, \ldots, T_n$ tal que $T_0 = w, T_n = T$ e $T_0 \Rightarrow T_1 \Rightarrow \ldots \Rightarrow T_n$ [14, p. 23].

As definições anteriores embora descrevam os T0L-systems não especificam um método de representar árvores axiais. De acordo com Hanan [6, p. 22], Lindenmayer apresentou um modo de o fazer na segunda parte do seu artigo publicado em 1968. Nesse artigo, Lindenmayer introduziu o conceito de palavra com colchetes. Este conceito materializou-se através da adição dos símbolos '[' e ']' ao alfabeto, criando assim os L-systems ramificados (BL-systems). Note-se que o termo BL-system resulta da expressão original "*bracketed L-system*". Estes dois novos símbolos são usados para delimitar um ramo, ficando entre eles todos os elementos que o constituem. Tal como acontece numa expressão numérica, também numa palavra correctamente formada, os colchetes surgem em pares correspondentes. O colchete esquerdo indica o vértice onde se vai formar o novo ramo, enquanto que o correspondente colchete direito vai terminar essa ramificação. Mais especificamente, a notação '$\ldots A\,[C]\,B\ldots$' significa que B é o segmento em linha recta que se segue a A num determinado eixo '$\ldots AB\ldots$', e que C é um segmento lateral que se originou no vértice final de A. Os colchetes podem estar encaixados uns dentro dos outros, representando assim os ramos com ordens mais elevadas.

O processo de se obter uma imagem a partir de um D0L-system ramificado é o mesmo que no caso de um D0L-system sem colchetes, no entanto, é necessário expor duas breves considerações: neste novo tipo de L-systems existe uma memória que tem a forma de um empilhamento, figurando no topo o último elemento guardado; durante a fase de

geração, os colchetes são considerados como constantes do sistema, sendo, na fase seguinte, interpretados pela "tartaruga" do seguinte modo:

[: Guardar na memória o estado actual. Para além do estado actual, a informação guardada pode conter outros atributos, tais como a espessura da linha a ser desenhada;

] : Retirar da memória o último estado que aí foi guardado e torná-lo no estado actual. Não é desenhado qualquer segmento de recta, embora, de um modo geral, a posição da "tartaruga" seja alterada.

Vai-se seguidamente completar a formalização da noção de interpretação gráfica de uma palavra de V^* apresentada na Secção 2.2, de modo a ficarem contemplados os novos símbolos. Para o caso dos BL-systems é ainda necessário definir a "função memória de ramificação" $m(\nu) = (m_1, m_2, \ldots, m_k) \in (\mathbb{R}^2 \times \mathbb{R})^k$, $k \leq l$, onde $\nu = a_1 a_2 \ldots a_l \in V^*$ é a palavra a interpretar. Note-se que cada m_i corresponde a um estado da "tartaruga". Considere-se também que apenas os colchetes fazem modificar a função memória de ramificação e que esta não contém qualquer elemento no início da fase de interpretação. Seja $\nu' = a_1 a_2 \ldots a_l a_{l+1} = \nu a_{l+1} \in V^*$ uma palavra de comprimento $l+1$. Por hipótese, tem-se já definido $I(\nu)$, $s(\nu)$ e $m(\nu)$ para esta palavra. Então, se

- $a_{l+1} = [$, define-se: $s(\nu') = s(\nu)$; $m(\nu') = (m_1, m_2, \ldots, m_k, m_{k+1})$, onde $m_{k+1} = s(\nu')$; Neste caso, $I(\nu') = I(\nu)$.
- $a_{l+1} =]$, define-se: $s(\nu') = m_k$; $m(\nu') = (m_1, m_2, \ldots, m_{k-1})$; Neste caso, $I(\nu') = I(\nu)$.

Segundo Prusinkiewicz *et al* [17], cada árvore axial pode ser descrita por uma palavra com colchetes correctamente formada, e cada palavra com colchetes correctamente formada descreve uma árvore axial. Assim sendo, na Figura 3.3 pode observar-se um exemplo de uma árvore axial que é descrita pela palavra $F[+F][-F[-F]F]F[+F][-F]$.

Alguns exemplos de estruturas bidimensionais com a forma de planta, geradas por D0L-systems ramificados, podem ser vistos nas Figuras 3.4 a 3.9. Os três primeiros exem-

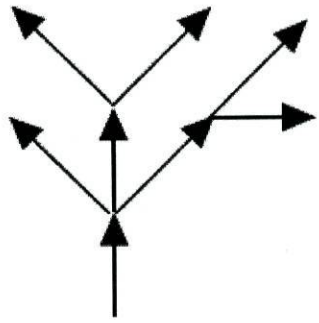

Figura 3.3: Representação gráfica de uma palavra com colchetes [14, p. 24].

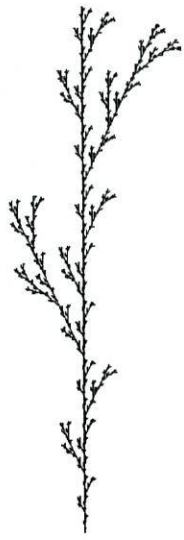

Figura 3.4: Estrutura gerada por um BL-system com $w : F$ e $P : F \rightarrow F[+F]F[-F]F$ (evolução de ordem 5 com $\delta = 25.7°$).

plos foram obtidos pela técnica de reescrita de arestas, enquanto que os três últimos foram obtidos por reescrita de vértices. Foi usado como estado inicial o terno ordenado $(x_0, y_0, \alpha_0) = (0, 0, 90°)$ e considerou-se ainda $d = 1$. Um BL-system também pode ser usado para modelar estruturas tridimensionais.

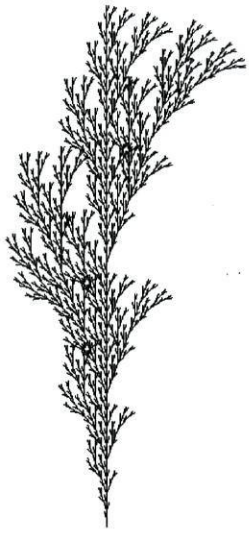

Figura 3.5: Estrutura gerada por um BL-system com $w : F$ e $P : F \rightarrow F[+F]F[-F][F]$ (evolução de ordem 5 com $\delta = 20°$).

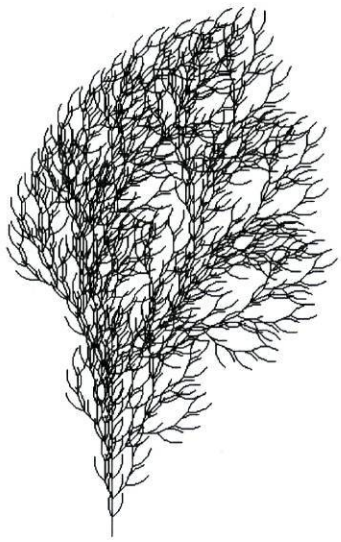

Figura 3.6: Estrutura gerada por um BL-system com $w : F$ e $P : F \rightarrow FF - [-F + F + F] + [+F - F - F]$ (evolução de ordem 4 com $\delta = 22.5°$).

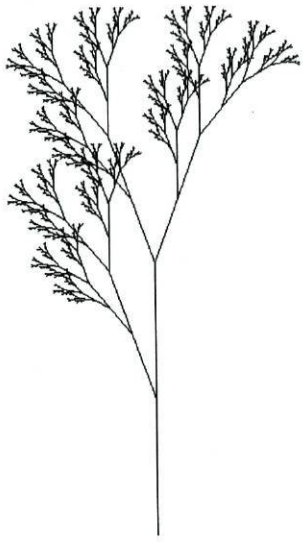

Figura 3.7: Estrutura gerada por um BL-system com $w : X$ e $P_1 : X \rightarrow F[+X]F[-X]+X$, $P_2 : F \rightarrow FF$ (evolução de ordem 7 com $\delta = 20°$).

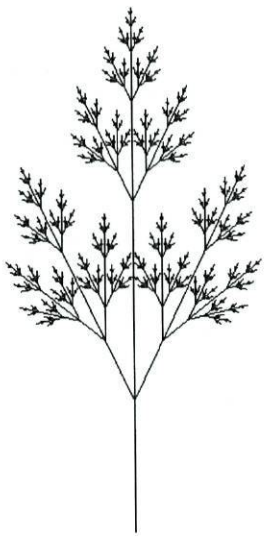

Figura 3.8: Estrutura gerada por um BL-system com $w : X$ e $P_1 : X \rightarrow F[+X][-X]FX$, $P_2 : F \rightarrow FF$ (evolução de ordem 7 com $\delta = 25.7°$).

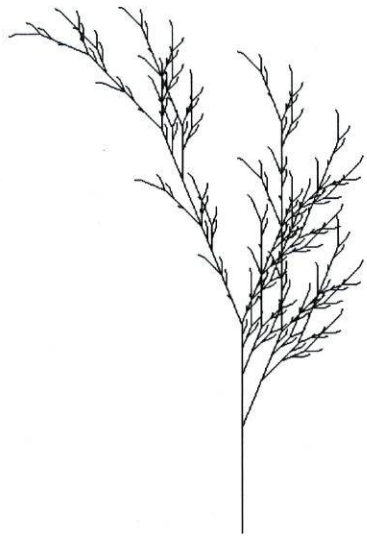

Figura 3.9: Estrutura gerada por um BL-system com $w : X$ e $P_1 : X \rightarrow F - [[X] + X] + F[+FX] - X$, $P_2 : F \rightarrow FF$ (evolução de ordem 5 com $\delta = 22.5°$).

3.2 L-systems Estocásticos

As imagens apresentadas até este momento foram geradas a partir de L-systems determinísticos. Como cada caracter tem apenas uma regra de substituição atribuída, todas as imagens obtidas através de uma evolução de ordem n de um mesmo L-system são exactamente iguais. No caso das plantas, pode então dizer-se que estes L-systems descrevem organismos individuais e não espécies. Por vezes esta situação pode não ser desejada, isto é, pode pretender-se a introdução de alguma variabilidade de planta para planta. Um exemplo de uma dessas situações é a criação de uma imagem realista de um campo de flores da mesma espécie. Por um lado, se fosse usado apenas um L-system para gerar todas as plantas, a imagem obtida pareceria muito pouco natural devido a uma notória regularidade artificial. Por outro lado, se para introduzir a desejada variabilidade fosse criado um L-system diferente para modelar cada uma das plantas, então estar-se-ia perante uma tarefa extremamente fastidiosa. Contudo, esta tarefa pode facilmente ser realizada com êxito através da utilização de um L-system estocástico (SL-system). Note-se que o termo SL-system resulta da expressão original "*stochastic L-system*". Este tipo de L-systems permite

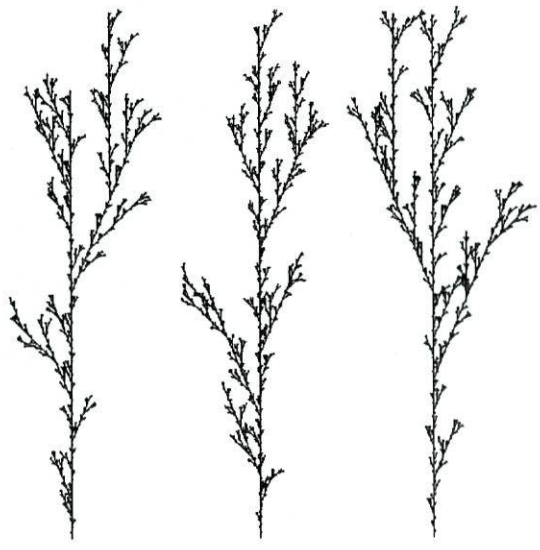

Figura 3.10: Estruturas ramificadas obtidas através de um S0L-system.

introduzir variabilidade de indivíduo para indivíduo através da aplicação estocástica das regras de substituição, preservando, no entanto, os aspectos gerais característicos da espécie modelada.

Um S0L-system é um quadruplo ordenado $G_\pi = \langle V, w, P, \pi \rangle$, onde

- V é um alfabeto,
- w é o axioma,
- P é um conjunto dito de regras de substituição,
- $\pi : P \rightarrow (0, 1]$ é uma função distribuição de probabilidade.

Este tipo de L-systems fornece regras de substituição alternativas com o mesmo antecessor mas com sucessores diferentes, tendo cada uma delas uma probabilidade atribuída através da função π. Em termos de notação, essa probabilidade surge sobre a seta que separa o antecessor do sucessor. Segundo Prusinkiewicz e Lindenmayer [14, p. 28], numa iteração em G_π, a cada ocorrência de uma determinada letra a numa palavra μ, a probabilidade de aplicar uma regra p, com antecessor a, é igual a $\pi(p)$. Tendo isto em consideração,

pode concluir-se que, numa iteração, em várias ocorrências de uma mesma letra podem ser aplicadas diferentes regras de substituição. Como seria de esperar, para cada letra $a \in V$, a soma das probabilidades de todas as regras com antecessor a é igual a 1. A probabilidade a atribuir a cada regra é determinada a partir da recolha e da análise estatística de dados de diversos espécimes [6, p. 20–21].

Na Figura 3.10 podem ver-se três exemplos de plantas geradas numa evolução de ordem 6 do S0L-system com as seguintes componentes:

$$
\begin{aligned}
&V : \{F, +, -, [,]\} \\
&w : F \\
&p_1 : F \overset{0.33}{\rightarrow} F[+F]F[-F]F \\
&p_2 : F \overset{0.33}{\rightarrow} F[+F]F \\
&p_3 : F \overset{0.34}{\rightarrow} F[-F]F
\end{aligned}
$$

Foi usado como estado inicial o terno ordenado $(x_0, y_0, \alpha_0) = (0, 0, 90°)$ e considerou-se ainda $d = 1$ e $\delta = 22.5°$. Tal como era esperado, de cada vez que o L-system anterior foi aplicado obtiveram-se imagens que representam diferentes exemplares de uma mesma espécie.

3.3 L-systems Paramétricos

Os L-systems munidos com a interpretação gráfica anteriormente apresentada são uma ferramenta de modelação bastante versátil. Contudo, embora permitam gerar, como foi visto até este ponto, uma vasta variedade de objectos, desde fractais até estruturas ramificadas com a forma de planta, a natureza discreta deste formalismo condiciona o seu poder de modelação. Uma das maiores limitações consiste na necessidade de todas as linhas desenhadas pela "tartaruga" serem múltiplos inteiros de um segmento unitário. Isto tem como consequência o uso de uma grande quantidade de comandos F, especialmente no caso da modelação de estruturas que apresentam segmentos com tamanhos muito variados.

Esta restrição faz ainda com que, por exemplo, um triângulo rectângulo isósceles não possa ser desenhado com exactidão, uma vez que a razão entre as medidas do comprimento da sua hipotenusa e de qualquer um dos seus catetos é um número irracional ($\frac{\sqrt{2c^2}}{c} = \frac{c\sqrt{2}}{c} = \sqrt{2}$). A limitação supracitada também se aplica aos ângulos. Por exemplo, se uma estrutura modelada apresentar ângulos com diversas amplitudes, é necessário definir o incremento angular δ como sendo um divisor comum a todos os ângulos presentes nessa estrutura. A natureza discreta dos L-systems faz assim com que, por vezes, a beleza matemática conferida aos L-systems pela sua simplicidade seja destruída [14, p. 40].

Com o objectivo de resolver este tipo de problemas, Lindenmayer foi o primeiro a propor a associação entre os símbolos já existentes nos L-systems e um número arbitrário de parâmetros numéricos. Esses parâmetros podem ser usados durante a fase de interpretação para representar, por exemplo, aspectos geométricos, tais como o comprimento ou o diâmetro dos segmentos, ou relações entre componentes, tais como a amplitude dos ângulos de ramificação [17]. Através da palavra $F(1) + (90)F(1) + (135)F(\sqrt{2})$ é agora possível desenhar com exactidão um triângulo rectângulo isósceles com um número de comandos muito reduzido. Esta nova ferramenta vem assim resolver os problemas apresentados no início desta secção.

Nas duas secções seguintes ir-se-á apresentar a definição de L-systems paramétricos (PL-systems) e extender a interpretação gráfica ao caso das palavras paramétricas.

3.3.1 D0L-systems Paramétricos

Um PL-system opera sobre palavras paramétricas, que são cadeias de módulos formados por símbolos de um alfabeto V aos quais estão associados parâmetros pertencentes ao conjunto dos números reais. Um módulos formado pela letra $A \in V$ e pelos parâmetros $a_1, a_2, \ldots, a_j \in \mathbb{R}$ denota-se por $A(a_1, a_2, \ldots, a_j)$. Cada módulo pertence ao conjunto $M = V \times \mathbb{R}^*$, onde $\mathbb{R}^*$ é o conjunto de todas as sequências finitas de parâmetros. O conjunto de todas as palavras formadas por módulos e o conjunto de todas as palavras não

vazias formadas por módulos representam-se por $M^* = (V \times \mathbb{R}^*)^*$ e $M^+ = (V \times \mathbb{R}^*)^+$, respectivamente.

No formalismo dos PL-systems distinguem-se dois tipos de parâmetros que estão associados entre si: os parâmetros actuais e os parâmetros formais. Os parâmetros actuais surgem nos módulos que compõem as L-strings, enquanto que os parâmetros formais são representados por variáveis que surgem na especificação das regras de substituição. A associação de uma letra com uma sequência de parâmetros formais é chamada de módulo formal, e uma sequência desses módulos é designada por palavra paramétrica formal. Se Σ é um conjunto de parâmetros formais, então $C(\Sigma)$ denota uma expressão lógica com parâmetros de Σ e $E(\Sigma)$ uma expressão aritmética com parâmetros do mesmo conjunto. Ambos os tipos de expressões consistem em constantes numéricas e parâmetros formais, combinados com operadores aritméticos, de exponenciação, relacionais, lógicos e parêntesis curvos. As expressões podem ainda incluir funções matemáticas, tais como funções trigonométricas, exponenciais, logarítmicas, entre muitas outras [6, p. 45]. As regras usuais para construir expressões sintacticamente correctas e de precedência das operações são mantidas. As expressões relacionais e as expressões lógicas tomam o valor zero quando são falsas e o valor 1 quando são verdadeiras. Considera-se ainda que a afirmação lógica especificada pela palavra vazia assume sempre o valor 1. Os conjuntos de todas as expressões lógicas e aritméticas correctamente construídas com parâmetros de Σ denotam-se por $\mathcal{C}(\Sigma)$ e $\mathcal{E}(\Sigma)$, respectivamente [14, p. 41].

Um 0L-system paramétrico é definido como sendo um quadruplo ordenado $G = \langle V, \Sigma, w, P \rangle$, onde

- V é um conjunto não vazio de símbolos designado por alfabeto do sistema,
- Σ é o conjunto dos parâmetros formais,
- $w \in (V \times \mathbb{R}^*)^+$ é uma palavra paramétrica não vazia chamada de axioma,
- $P \subset (V \times \Sigma^*) \times \mathcal{C}(\Sigma) \times (V \times \mathcal{E}(\Sigma))^*$ é um conjunto finito dito de regras de substituição.

As regras de substituição de um PL-system livre de contexto têm assim a forma

$$antecessor : cond \rightarrow sucessor$$

onde o antecessor é um módulo formal individual, o sucessor é uma palavra paramétrica formal, e a condição (*cond*) é uma expressão lógica. Note-se que uma regra de substituição pode não ter associado qualquer parâmetro formal nem expressão lógica. Para uma dada regra, é ainda assumido que um determinado parâmetro formal não pode surgir mais do que uma vez no antecessor, e que todos os parâmetros formais na condição têm que constar no antecessor. Por exemplo, uma regra com antecessor $A(t)$, condição $t > 5$ e sucessor $B(t+1)CD(t \wedge 0.5, t-2)$ é representada como

$$A(t) : t > 5 \rightarrow B(t+1)CD(t \wedge 0.5, t-2)$$

Diz-se que uma regra de substituição combina com um módulo numa palavra paramétrica se as seguintes condições forem reunidas:

- a letra do módulo e a letra do antecessor da regra coincidem,
- o número de parâmetros actuais no módulo é igual ao número de parâmetros formais no antecessor da regra, e
- seja verdadeira a proposição que se obtém substituindo os parâmetros formais da condição pelos parâmetros actuais correspondentes.

Uma regra cujo antecessor verifique simultaneamente as três condições anteriores pode ser aplicada a um módulo. Ao ser aplicada, esse módulo é substituído pela palavra de módulos especificada no sucessor da regra e os valores das expressões são calculados de modo a produzir os parâmetros actuais. Por exemplo, a regra de substituição anterior combina com o módulo $A(16)$, pois a letra A é comum a este módulo e ao antecessor da regra, existe um parâmetro actual em $A(16)$ e um parâmetro formal em $A(t)$, e a expressão lógica $t > 5$ é verdadeira para $t = 16$. A regra será então aplicada, substituindo o módulo $A(16)$

pela palavra paramétrica $B(17)CD(4,14)$. Tal como sucede no caso não-paramétrico, se não existir no conjunto P uma regra que combine com um determinado módulo, então este será substituído por si próprio [6, p. 47].

Num PL-system G, se um módulo a produz uma palavra paramétrica χ como resultado da aplicação de uma regra de substituição, então esta situação representa-se por $a \rightarrow \chi$. Uma palavra paramétrica $\nu = \chi_1\chi_2 \ldots \chi_m$ deriva directamente de uma outra palavra $\mu = a_1a_2 \ldots a_m$, e escreve-se $\mu \Rightarrow \nu$, se e só se $a_i \rightarrow \chi_i$ para todo o $i = 1, 2, \ldots, m$. Uma determinada palavra paramétrica ν é gerada por G numa evolução de ordem n se existir uma sucessão de desenvolvimento $g_0, g_1, \ldots, g_n$ tal que $g_0 = w$, $g_n = \nu$ e $g_0 \Rightarrow g_1 \Rightarrow \ldots \Rightarrow g_n$ [14, p. 42].

Segundo Hanan [6, p. 47–48], um 0L-system paramétrico é determinístico se e só se não existirem duas regras que combinem com um mesmo módulo numa palavra. Esta exigência será satisfeita se para todos os grupos de regras que tenham antecessores com a mesma letra e com o mesmo número de parâmetros formais, não existirem duas condições que sejam simultaneamente verdadeiras. Na Figura 3.11 podem ver-se as palavras obtidas através das primeiras iterações do D0L-system paramétrico com as seguintes componentes:

$$
\begin{aligned}
&w : B(2)A(4,4)\\
&p_1 : A(x,y) : y <= 3 \rightarrow A(2 * x, x + y)\\
&p_2 : A(x,y) : y > 3 \quad \rightarrow B(x)A(x/y, 0)\\
&p_3 : B(x) \quad : x < 1 \quad \rightarrow C\\
&p_4 : B(x) \quad : x >= 1 \rightarrow B(x-1)
\end{aligned}
$$

3.3.2 Interpretação Gráfica de Palavras Paramétricas

Na secção anterior referiu-se que os PL-systems operam sobre palavras paramétricas, explicou-se como eram constituídas essas palavras e explicitou-se sumariamente a fase de geração. Vai-se agora mostrar como a "tartaruga" interpreta uma palavra paramétrica.

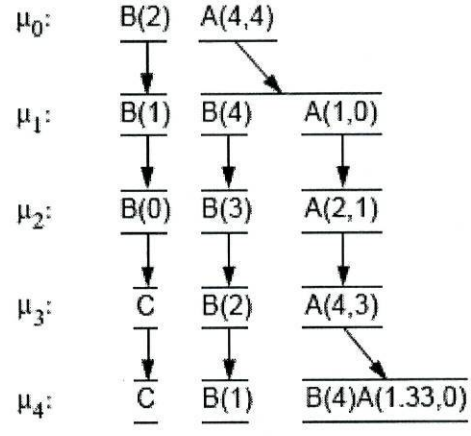

Figura 3.11: Sequência inicial de palavras geradas por um D0L-system paramétrico [14, p. 43].

Apesar do conceito básico da interpretação gráfica apresentado anteriormente (ver Secções 2.2 e 2.4) não sofrer alterações com a introdução dos parâmetros, verificam-se algumas modificações como consequência natural dessa introdução. A principal diferença reside no facto de não haver a necessidade de definir o avanço da "tartaruga" nem o incremento angular, ou seja, a alteração do estado da "tartaruga" passa a ser feita recorrendo ao valor dos parâmetros presentes nos módulos. Se existir pelo menos um parâmetro associado a um símbolo do alfabeto V, o estado da "tartaruga" será controlado pelo valor do primeiro. Considerando que o valor desse parâmetro é positivo ($a > 0$), então a "tartaruga" responde aos seguintes comandos:

$F(a)$: Avançar na direcção actual um passo de comprimento a. A posição muda para (x', y', z'), onde $x' = x + a.\vec{H}_x$, $y' = y + a.\vec{H}_y$ e $z' = z + a.\vec{H}_z$. É desenhado um segmento de recta entre os pontos (x, y, z) e (x', y', z');

$f(a)$: Avançar na direcção actual um passo de comprimento a. A posição muda para (x', y', z'), onde $x' = x + a.\vec{H}_x$, $y' = y + a.\vec{H}_y$ e $z' = z + a.\vec{H}_z$. Não é desenhado qualquer segmento entre os pontos (x, y, z) e (x', y', z');

$+(a)$: Virar para a esquerda a graus, usando a matriz de rotação $R_U(a)$;

$-(a)$: Virar para a direita a graus, usando a matriz de rotação $R_U(-a)$;

$\vee(a)$: Virar para baixo a graus, usando a matriz de rotação $R_L(a)$;

$\wedge(a)$: Virar para cima a graus, usando a matriz de rotação $R_L(-a)$;

$\backslash(a)$: Rolar para a esquerda a graus, usando a matriz de rotação $R_H(a)$;

$/(a)$: Rolar para a direita a graus, usando a matriz de rotação $R_H(-a)$.

Repare-se que os símbolos '+', '−', '∧' e '/' são usados simultaneamente como símbolos do alfabeto do L-system e como operadores em expressões lógicas e aritméticas, pelo que o seu significado vai depender do contexto onde surgem. O conceito de palavra com colchetes introduzido por Lindenmayer também pode ser aplicado a palavras paramétricas. O formalismo dos PL-systems pode ser assim extendido de modo a permitir modelar estruturas ramificadas. Tal como sucedeu para o caso não-paramétrico, essa extensão faz-se através da adição dos símbolos '[' e ']' ao alfabeto do L-system [6, p. 54–55].

A resolução do problema de desenhar um triângulo rectângulo isósceles com exactidão, anteriormente apresentada, pode agora ser totalmente contemplada. Um triângulo deste tipo (Figura 3.12) pode então ser gerado, tendo em conta as limitações do computador, através da interpretação gráfica da palavra paramétrica resultante de uma qualquer evolução de ordem maior ou igual a 1 do seguinte PL-system:

$$w : X(1)$$
$$p : X(x) \rightarrow F(x) + (90)F(x) + (135)F(2 \wedge 0.5 * x)$$

As regras para os símbolos 'F', '+' e '−' não foram apresentadas, o que significa que os módulos correspondentes serão substituídos por si próprios durante a fase de geração. A Figura 3.12 foi obtida usando como estado inicial o terno ordenado $(x_0, y_0, \alpha_0) = (0, 0, 0°)$. A introdução dos parâmetros permitiu assim aumentar a diversidade de imagens que podem ser facilmente visualizadas usando a interpretação gráfica apresentada.

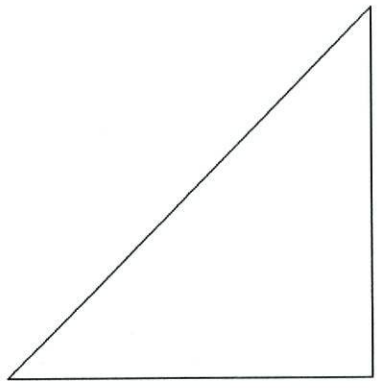

Figura 3.12: Triângulo rectângulo isósceles gerado a partir de um D0L-system paramétrico.

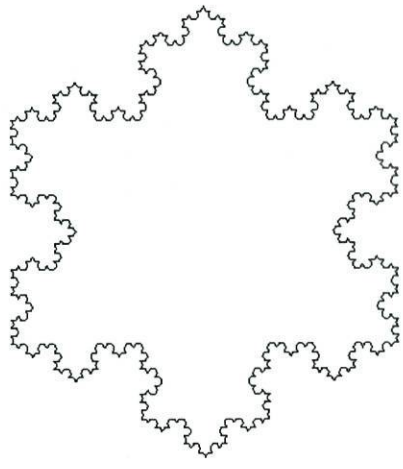

Figura 3.13: Floco de neve de Koch gerado por um D0L-system paramétrico.

Apresentam-se seguidamente mais quatro exemplos que permitem ilustrar o funcionamento dos PL-systems. Os três primeiros foram obtidos por reescrita de arestas e foi usado como estado inicial o terno ordenado $(x_0, y_0, \alpha_0) = (0, 0, 0°)$, enquanto que o último resultou de reescrita de vértices, tal como o triângulo atrás representado, e foi usado como estado inicial o terno ordenado $(x_0, y_0, \alpha_0) = (0, 0, 90°)$. Como seria de esperar, os PL-systems permitem gerar todas as figuras que os L-sytems não-paramétricos também conseguem, só que de um modo mais simples e elegante. Por exemplo, o floco de neve de Koch (Figura 3.13) pode agora ser obtido pelo seguinte L-system:

$$w : F(1) - (120)F(1) - (120)F(1)$$
$$p : F(x) \rightarrow F(x/3) + (60)F(x/3) - (120)F(x/3) + (60)F(x/3)$$

Observe-se que o axioma permite desenhar um triângulo equilátero com arestas de comprimento unitário, e que a imagem obtida resulta de uma evolução de ordem 4. O exemplo seguinte permite construir uma "linha de árvores". Na Figura 3.14 pode ver-se a curva resultante de uma evolução de ordem 5 do L-system com as seguintes componentes:

$$w : F(1)$$
$$p : F(x) \rightarrow F(x * 0.3) + (86)F(x * 0.21 \wedge 0.5) - (172)F(x * 0.21 \wedge 0.5) + (86)F(x * 0.7)$$

Observe-se que devido ao facto de todas as arestas, quer sejam compridas ou curtas, serem substituídas em cada iteração, a curva preenche as diferentes áreas com uma densidade desigual. Fazendo algumas alterações ao L-system anterior é possível obter uma curva que se encontre distribuída mais uniformemente (Figura 3.15). Essas alterações consistem em atrasar a substituição dos segmentos mais curtos em relação aos mais compridos. As componentes do L-system que permite atingir este objectivo são:

$$w : F(1, 0)$$
$$p_1 : F(x, t) : t = 0 \rightarrow F(x * 0.3, 2) + (86)F(x * 0.21 \wedge 0.5, 1) - (172)$$
$$F(x * 0.21 \wedge 0.5, 1) + (86)F(x * 0.7, 0)$$
$$p_2 : F(x, t) : t > 0 \rightarrow F(x, t - 1)$$

A Figura 3.15 é o resultado de uma evolução de ordem 9 do PL-system anterior. Por fim, pode ver-se na Figura 3.16 um exemplo de um tipo de padrão de ramificação encontrado

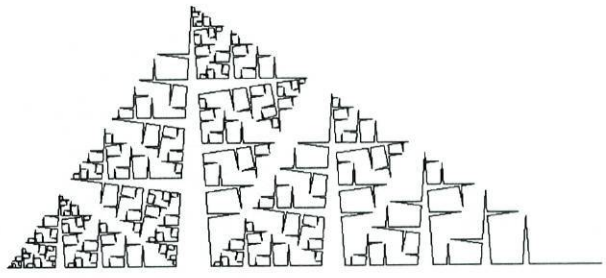

Figura 3.14: Exemplo de uma curva que sugere a representação de uma "linha de árvores".

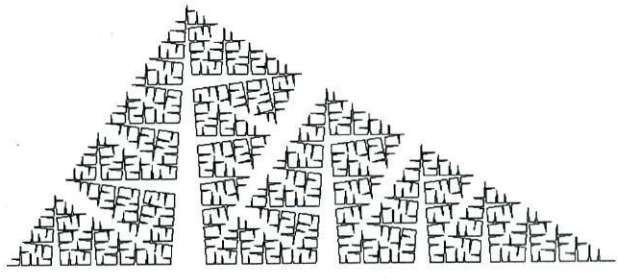

Figura 3.15: Exemplo de uma outra curva que sugere a representação de uma "linha de árvores".

em muitas plantas herbáceas e árvores [14, p. 50]. Essa figura resultou de uma evolução de ordem 11 do seguinte L-system:

$$
\begin{aligned}
&w : X \\
&p_1 : X \quad \rightarrow F(1)[+(86)X][-(86)X] \\
&p_2 : F(x) \rightarrow F(x * 1.456)
\end{aligned}
$$

3.4 Modelos de Desenvolvimento de Árvores

Segundo Prusinkiewicz e Lindenmayer [14, p. 51–53], a simulação computacional de padrões de ramificação já tem uma história relativamente longa. O primeiro modelo foi proposto por Ulam e aplicava o conceito de autómato celular (ver [10]), conceito esse que fora previamente desenvolvido por Neumann e Ulam. Este modelo deu origem a diversas extensões, nomeadamente por Meinhardt, por Greene e por Cohen. Todos estes modelos

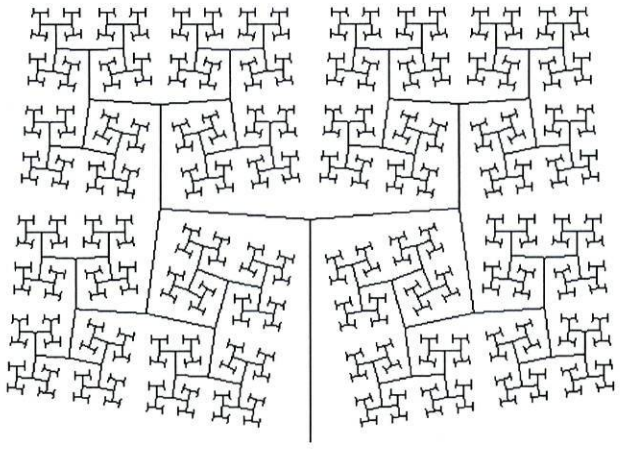

Figura 3.16: Padrão de ramificação gerado por um D0L-system paramétrico.

eram bastante complexos visto que davam grande ênfase às interacções entre os vários elementos de uma estrutura em desenvolvimento, tanto quanto à própria estrutura e ao ambiente envolvente. Embora as interacções influenciem o desenvolvimento das plantas, ao serem tidas em consideração na elaboração dos modelos, vão provocar um aumento da sua complexidade. Este facto permite compreender o prevalecimento de modelos mais simples. O primeiro desses modelos simples foi proposto por Honda, que estudou a influência dos ângulos de ramificação e do comprimentos dos ramos na forma das árvores, admitindo as seguintes suposições:

- Os segmentos da árvore são rectilíneos e a sua espessura não é considerada;
- Um segmento dá origem a dois novos segmentos através de um processo de ramificação;
- Os comprimentos dos novos segmentos são encurtados pelas constantes r_1 e r_2, relativamente ao segmento que lhes deu origem;
- O segmento "mãe" e os dois novos segmentos estão contidos no mesmo plano, designado por plano do ramo. Os segmentos "filhos" formam ângulos de ramificação constantes, a_1 e a_2, relativamente ao segmento "mãe";

- O plano do ramo está orientado de modo a estar o mais próximo possível de um plano horizontal. Uma excepção é feita para os ramos ligados ao tronco principal. Nesse caso, é conservado um ângulo constante de divergência α entre os segmentos laterais que nele se formam.

Juntamente com Fisher e Tomlinson, Honda realizou alguns melhoramentos ao seu modelo e aplicou-o para investigar padrões de ramificação de árvores reais. Aono e Kunii basearam-se nos resultados de Honda para apresentar os seus próprios modelos de árvores e sugeriram algumas extensões ao modelo de Honda. A mais importante dessas extensões consistia na alteração da posição dos ramos de modo a permitir simular os efeitos provocados pelo vento, pelo fototropismo e pela gravidade. Houve outros trabalhos no mesmo sentido, nomeadamente os de de Reffye e Armstrong, que desenvolveram métodos mais precisos, baseados na Física, para simular a curvatura dos ramos.

De acordo com Prusinkiewicz e Lindenmayer [14, p. 53], os modelos de Honda e de Aono e Kunii eram interpretados usando segmentos rectilíneos, de diâmetro constante ou variável, para representar apenas o "esqueleto das árvores". Um melhoramento substancial na criação de imagens realistas foi alcançando por Bloomenthal e Oppernheimer, através da introdução de ramos com curvatura, de casca com textura e de folhas.

As abordagens originadas a partir dos trabalhos de Honda definem as estruturas ramificadas usando algoritmos determinísticos. Contudo, existe um outro conjunto de abordagens que se baseia em mecanismos estocásticos. Estes modelos baseiam-se no paradigma de especificar as estruturas das árvores de acordo com leis estocásticas características da espécie a ser modelada.

Aono e Kunii foram os primeiros a aplicar os L-systems para modelar graficamente o desenvolvimento de árvores, tendo utilizado, numa fase inicial, a definição original deste formalismo apresentada por Lindenmayer em 1968. Contudo, dado que a estrutura das árvores, particularmente em climas moderados, é fortemente dependente dos factores ambientais, os quais não são directamente capturados pelos L-systems, e dada a dificuldade

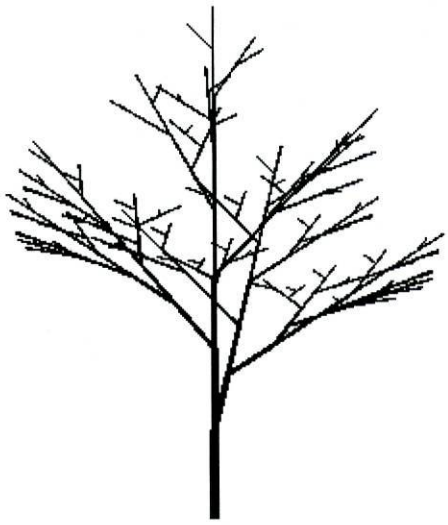

Figura 3.17: Exemplo de uma estrutura gerada por um PL-system segundo o modelo de Honda.

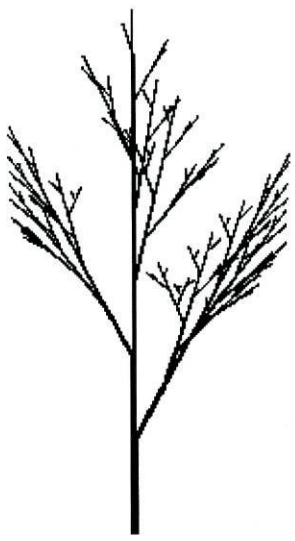

Figura 3.18: Exemplo de uma outra estrutura gerada por um PL-system segundo o modelo de Honda.

em expressar com precisão alguns comprimentos de segmentos e amplitudes de ângulos de ramificação, eles consideraram este tipo de L-systems inadequado para modelar os complexos padrões de ramificação de plantas de grande dimensão. No entanto, com a adição de parâmetros aos L-systems clássicos, estes problemas foram resolvidos e os PL-systems com a interpretação gráfica de palavras tornaram-se assim numa poderosa ferramenta para modelar o desenvolvimentos de árvores [6, p. 68]. Por exemplo, o seguinte L-system permite implementar um modelo de Honda, no qual um dos ângulos de ramificação é igual a zero ($a_1 = 0$):

$$
\begin{aligned}
& w : \wedge(90)A(1, 0.04) \\
& p_1 : A(l, w) \rightarrow !(w)F(l)[\vee(a_0)B(r_2 * l, w_r * w)]/(d)A(r_1 * l, w_r * w) \\
& p_2 : B(l, w) \rightarrow !(w)F(l)[-(a_2)@C(r_2 * l, w_r * w)]C(r_1 * l, w_r * w) \\
& p_3 : C(l, w) \rightarrow !(w)F(l)[+(a_2)@B(r_2 * l, w_r * w)]B(r_1 * l, w_r * w)
\end{aligned}
$$

onde $r_1 = 0.9$ (razão de encurtamento do tronco), $r_2 = 0.8$ (razão de encurtamento dos ramos), $a_0 = 45$ (ângulo de ramificação a partir do tronco), $a_2 = 45$ (ângulo de ramificação para os eixos laterais), $d = 137.5$ (ângulo de divergência) e $w_r = 0.707$ (taxa de diminuição do diâmetro dos segmentos). Na Figura 3.17 pode ver-se uma imagem gerada numa evolução de ordem 8 do L-system anterior. Note-se que em todos os exemplos apresentados nesta secção, a "tartaruga" estava inicialmente posicionada na origem e a sua orientação inicial era dada pelos vectores $\vec{H} = (1,\ 0,\ 0)$, $\vec{L} = (0,\ 1,\ 0)$ e $\vec{U} = (0,\ 0,\ 1)$. Através da modificação das diferentes constantes numéricas presentes nas regras de substituição deste L-system é possível obter-se uma vasta variedade de estruturas com a forma de árvore. A Figura 3.18 consiste numa imagem obtida através do processo supracitado. Neste último exemplo foram usadas as seguintes constantes: $r_1 = 0.9$, $r_2 = 0.7$, $a_0 = 30$, $a_2 = -30$, $d = 137.5$ e $w_r = 0.707$. Nas regras deste L-system observa-se a presença de dois novos símbolos do alfabeto para os quais ainda não foi explicitada a sua função. O módulo '$!(w)$' define como w o diâmetro dos segmentos, enquanto que o símbolo '@' ordena que a "tartaruga" role em torno do vector $\vec{H}$, de modo a que o vector $\vec{L}$ seja colocado na posição horizontal, tal como é exigido pelo modelo de Honda. Na prática, este último símbolo

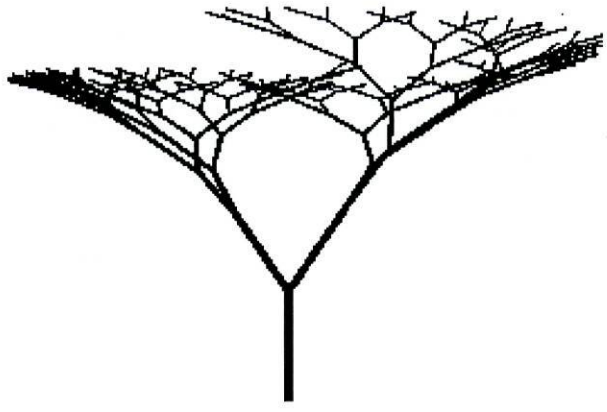

Figura 3.19: Exemplo de uma estrutura gerada por um PL-system segundo o modelo de Aono e Kunii.

modifica a orientação da "tartaruga" no espaço de acordo com as seguintes fórmulas:

$$\vec{H}' = \vec{H}, \qquad \vec{L}' = \frac{\vec{V} \times \vec{H}}{\left\| \vec{V} \times \vec{H} \right\|} \qquad e \qquad \vec{U}' = \vec{H} \times \vec{L}'$$

onde $\vec{H}$, $\vec{L}$ e $\vec{U}$ são os vectores associados à "tartaruga" definidos na Secção 2.4, e $\vec{V}$ indica o sentido oposto ao da gravidade. O exemplo que se segue permite implementar um modelo de Aono e Kunii, no qual ambos os ângulos de ramificação são diferentes de zero ($a_1 \neq 0$ e $a_2 \neq 0$):

$$w : \wedge(90)A(1, 0.04)$$
$$p_1 : A(l, w) \rightarrow !(w)F(l)[\vee(a_1)B(r_1 * l, w_r * w)]/(180)[\vee(a_2)B(r_2 * l, w_r * w)]$$
$$p_2 : B(l, w) \rightarrow !(w)F(l)[+(a_1)@B(r_1 * l, w_r * w)][-(a_2)\$B(r_2 * l, w_r * w)]$$

onde $r_1 = 0.9$ e $r_2 = 0.8$ (razões de encurtamento dos segmentos), $a_1 = 35$ e $a_2 = 35$ (ângulos de ramificação), e $w_r = 0.707$ (taxa de diminuição do diâmetro dos segmentos). Na Figura 3.19 pode ver-se uma imagem gerada após uma evolução de ordem 8 do L-system anterior. Modificando novamente as diferentes constantes numéricas presentes nas regras deste L-system é possível gerar uma grande variedade de estruturas com a forma de árvore. A Figura 3.20 foi obtida através da alteração dos valores dos ângulos de ramificação para $a_1 = 20$ e $a_2 = 50$.

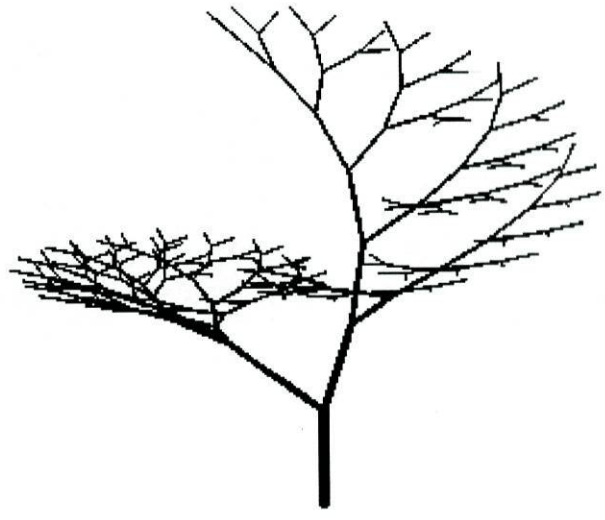

Figura 3.20: Exemplo de uma outra estrutura gerada por um PL-system segundo o modelo de Aono e Kunii.

Os exemplos anteriores permitem mostrar que os modelos de Honda e de Aono e Kunii fazem parte do conjunto daqueles que podem ser expressos usando o formalismo dos PL-systems. Os resultados obtidos por estes investigadores demonstram que os L-systems podem desempenhar um papel importante enquanto ferramenta para fazer uma simulação correcta, do ponto de vista biológico, do desenvolvimento de árvores e para gerar imagens realistas [14, p. 62].

3.5 L-systems Sensíveis ao Contexto

Até ao presente momento foram apresentados apenas L-systems livres de contexto, isto é, aqueles em que as regras de substituição são aplicadas sem se ter em consideração o contexto onde os símbolos surgem. Vai-se agora explicitar muito sumariamente o conceito de L-systems sensíveis ao contexto. Nesta categoria de L-systems, a aplicação das regras vai depender do contexto onde os caracteres se encontram inseridos.

Segundo Hanan [6, p. 7–9], os modelos de desenvolvimento de plantas podem ser caracteri-

zados pelo modo como circula a informação que determina o processo de desenvolvimento dos organismos modelados. Quando a informação passa directamente do módulo "pai" para o módulo "filho", diz-se que o desenvolvimento é controlado por mecanismos de linhagem. Este tipo de mecanismos pode ser encarado como o controlo do desenvolvimento através da transmissão de informação genética. Por sua vez, diz-se que o desenvolvimento é controlado por mecanismos interactivos quando a informação vem do exterior do módulo. Este fluxo de informação vindo do exterior pode ser endógeno ou exógeno. No primeiro caso, de que são exemplos o fluxo de nutrientes e de hormonas, a informação passa entre módulos adjacentes dentro da estrutura. No segundo caso, de que são exemplos a reacção à luz e à gravidade, a informação é transferida através do meio onde a estrutura está a crescer.

A classificação dos L-systems tem em conta os dois tipos de mecanismos acabados de apresentar. Os L-systems usados para modelar situações onde ocorrem apenas mecanismos de linhagem, ou seja, o contexto onde os módulos surgem não influencia o seu desenvolvimento, são designados por L-systems livres de contexto ou, simplesmente, por 0L-systems (termo resultante da expressão "*zero-interaction L-systems*"), como já foi referido na Secção 2.1. Por sua vez, os L-systems usados para modelar organismos onde ocorrem mecanismos interactivos, ou seja, o contexto onde os módulos surgem influencia o seu desenvolvimento, são designados por L-systems interactivos ou, simplesmente, por IL-systems (termo resultante da expressão "*interactive L-systems*").

De acordo com Prusinkiewicz [12], foram propostas e profundamente estudadas várias extensões aos 0L-systems. Salomaa (1973), Herman e Rozenberg (1975) e Lindenmayer e Rozenberg (1976) foram alguns dos investigadores que se dedicaram a este assunto. Destes estudos resultou a apresentação de dois tipos de IL-systems: os 1L-systems ("*one-interaction L-systems*") e os 2L-systems ("*two-interaction L-systems*"). Nos 2L-systems, as regras de substituição têm a forma

$$a_l < a > a_r \rightarrow \chi$$

onde o símbolo a, designado por antecessor estrito, é substituído pela palavra χ se e só se

for precedido por a_l e sucedido por a_r. Pode assim dizer-se que a_l e a_r constituem, respectivamente, o contexto à esquerda e o contexto à direita de a nesta regra de substituição. No caso dos 1L-systems, as regras têm a forma

$$a_l < a \rightarrow \chi \qquad ou \qquad a > a_r \rightarrow \chi$$

uma vez que nestes L-systems se tem em atenção apenas o contexto existente de um dos lados. Considera-se que, se num L-system existirem regras sensíveis ao contexto e outras que não o sejam (com o mesmo antecessor estrito), as primeiras têm precedência sobre as últimas. O seguinte 1L-system permite ilustrar de um modo muito simples o funcionamento deste tipo de L-systems:

$$w : baaa$$
$$p_1 : b < a \rightarrow b$$
$$p_2 : \quad b \rightarrow a$$

As primeiras cinco gerações obtidas por este D1L-system podem ser vistas em baixo.

$$baaa$$
$$abaa$$
$$aaba$$
$$aaab$$
$$aaaa$$

Observe-se que a letra b se vai deslocando da esquerda para a direita, simulando-se assim a propagação de um sinal ao longo de uma palavra.

Os contextos também podem ser introduzidos nos restantes tipos de L-systems já estudados. Contudo, nos BL-systems essa introdução é mais difícil do que nos L-systems sem colchetes, uma vez que a palavra que representa uma árvore axial não preserva a vizinhança dos segmentos. Assim sendo, durante o processo de verificação dos contextos para determinar qual a regra a ser aplicada, há a necessidade de "passar por cima" de conjuntos de símbolos que representam ramos ou partes de ramos. Por exemplo, uma regra com antecessor $BC < S > G[H]M$ pode ser aplicada ao símbolo S na seguinte

palavra com colchetes:

$$ABC[DE][SG[HI[JK]L]MNO]$$

Nesta situação particular, passou-se por cima dos símbolos $[DE]$ na verificação do contexto à esquerdo, e sobre $I[JK]L$ na procura do contexto à direita [14, p. 30–32].

No caso dos 2L-systems paramétricos, uma regra de substituição tem o seguinte formato:

$$a_l < antecessor > a_r : cond \rightarrow sucessor$$

Cada uma das três componentes do antecessor (contexto à esquerda a_l, antecessor estrito e contexto à direita a_r) é uma palavra paramétrica formada por letras do alfabeto V e por parâmetros formais do conjunto Σ, sendo o antecessor estrito constituído por um único módulo. As restantes componentes da regra (a condição *cond* e o sucessor) definem-se do mesmo modo que nos 0L-systems paramétricos. Pode ver-se em baixo um exemplo de uma regra de substituição deste tipo de L-systems.

$$A(x) < B(y) > C(z) : x + y + z > 10 \rightarrow E((x + y)/2)F((y + z)/2)$$

Esta regra pode ser aplicada ao módulo $B(5)$ na seguinte palavra paramétrica

$$\ldots A(4)B(5)C(6)\ldots$$

uma vez que a regra combina com esse módulo. Assim sendo, o módulo $B(5)$ será substituído por $E(4.5)F(5.5)$. Os 2L-systems paramétricos constituem uma ferramenta adequada para expressar modelos de desenvolvimento que envolvam a difusão de substâncias ao longo de um organismo. De um modo análogo ao anteriormente apresentado, também nos 1L-systems paramétricos apenas se verifica a existência de uma única interacção (contexto à esquerda ou à direita) [14, p. 43].

Capítulo 4

Modelação de Árvores com o Maple

Este capítulo inicia-se com algumas noções básicas de *Maple* de modo a que os procedimentos usados na criação da maioria das imagens presentes nesta dissertação possam ser compreendidos mais facilmente, e termina com a apresentação desses mesmos procedimentos.

4.1 Algumas Noções Básicas de Maple

4.1.1 Introdução ao Maple

O *Maple* faz parte de uma família já relativamente vasta de ambientes científicos designados por sistemas de Álgebra computacional, da qual também fazem parte, por exemplo, o *Mathematica*, o *MathLab* e o *Derive*. A Álgebra computacional é uma área que teve um forte impulso nas décadas de 60 e 70 do século passado, período em que foram criados importantes algoritmos para a integração analítica e para a factorização de polinómios [11, p. 3]. Os sistemas de Álgebra computacional usados durante os anos 70, tais como o *Reduce* ou o *Macsyma*, requeriam muitos megabytes de memória *RAM* e muito tempo para efectuar cálculos matemáticos rotineiros, pelo que apenas um pequeno número de

investigadores com acesso a poderosos computadores podiam utilizar essa tecnologia. Com o intuito de inverter esta situação, o Grupo de Computação Simbólica ("*Symbolic Computation Group*") da Universidade de Waterloo concebeu o projecto *Maple* no final do ano de 1980. O principal objectivo deste projecto era a produção de um sistema de Álgebra computacional que, devido à sua eficiência em termos de memória e tempo de processamento, seria acessível a um grande número de investigadores e alunos [25]. Eles criaram um pequeno núcleo escrito em linguagem *C* e, a partir desse núcleo, desenvolveram uma nova linguagem, sendo o próprio *Maple* escrito nessa linguagem. A grande maioria dos algoritmos disponibilizados neste sistema estão assim escritos em linguagem *Maple* [11, p. 3]. A partir de 1982 começaram as demonstrações públicas do sistema *Maple* e, no ano seguinte, já era usado em diversas instituições para realizar trabalhos de investigação e na leccionação de algumas disciplinas. Desde 1988, o *Maple* tem sido comercializado pela *Waterloo Maple Software*, a qual opera actualmente sob a marca comercial *Maplesoft* [25]. O desenvolvimento deste software resulta de um esforço combinado entre a *Waterloo Maple Software*, o Grupo de Computação Simbólica da Universidade de Waterloo e algumas organizações de investigação [22]. Deste trabalho colectivo já surgiram diversas versões do sistema *Maple*, sendo a versão 11 a mais recente [24].

O *Maple* é uma poderosa ferramenta matemática que possui inúmeros recursos algébricos, numéricos e gráficos, além de também funcionar como uma linguagem de programação. Com o *Maple* é possível realizar cálculos que contenham símbolos como π, ∞ ou $\sqrt{2}$ sem haver a necessidade de fazer aproximações numéricas, e realizar simplificações e cálculos com expressões algébricas como $ax^2 + bx + c$ ou $x^3 + log(x)$ sem ser preciso atribuir valores numéricos às variáveis ou constantes. Devido a estas propriedades é possível encontrar soluções exactas para problemas que envolvam a resolução de equações, de derivadas, de integrais, o cálculo matricial, entre outros, tudo isto com a possibilidade de utilizar recursos que permitem visualizar gráficos bidimensionais ou tridimensionais. Como todos os recursos estão integrados num único programa, é possível fazer de imediato, por exemplo, uma análise gráfica ou numérica partindo de um resultado algébrico [11, p. 3]. O *Maple*

possui mais de 2500 funções, entre as quais se encontram as funções matemáticas básicas disponíveis em qualquer calculadora científica ou gráfica, e como incorpora uma linguagem de programação de alto nível permite ainda que o utilizador defina as suas próprias funções [23].

4.1.2 Iniciar uma Sessão de Maple

Ao iniciar-se o *Maple* surge uma janela composta por alguns menus, por botões e pela folha de trabalho ("*worksheet*"). É na folha de trabalho que são escritas e executadas as instruções e onde se podem ver os resultados obtidos. A *worksheet* é um caderno virtual de anotações de cálculos que pode ser gravada para ser lida e, eventualmente, modificada numa posterior sessão de trabalho. Quando uma *worksheet* é lida novamente, os resultados que lá surgem não estão na memória do *Maple*, sendo necessário executá-los novamente para os tornar activos. Uma folha de trabalho pode ainda ser impressa ou convertida num ficheiro LaTeX.

São quatro os tipos diferentes de linhas que podem ser visualizados numa *worksheet*:

- linhas de entrada para execução de comandos;
- linhas de saída de comandos executados;
- linhas de texto;
- linhas de comentário a erros.

As linhas de entrada de comandos são vermelhas, por defeito, e iniciam-se com o caracter '>'. Cada comando digitado deve terminar com ';' (ponto e vírgula) ou com ':' (dois pontos), premindo-se seguidamente a tecla [*Enter*]. Se o comando terminar com ponto e vírgula, o resultado da execução será mostrado logo em seguida numa linha de saída. Pelo contrário, se terminar com dois pontos, o resultado não será mostrado, podendo contudo

ser usado posteriormente. Por sua vez, as linhas de saída têm, por defeito, a cor azul e surgem centradas, logo após uma linha de entrada terminada com ';'. Estas linhas podem ser compostas por expressões escritas na notação matemática usual ou gráficos. As linhas de texto têm, por defeito, a cor preta e são inseridas após se seleccionar o botão que apresenta a letra 'T'. Por fim, as linhas de comentários a erros aparecem, por defeito, com a cor lilás e num tamanho de letra menor. Estas linhas surgem quando o utilizador introduz uma instrução não decifrável pelo *Maple* devido à existência de algum erro.

Depois de se iniciar uma sessão *Maple*, o sistema fornece uma linha de entrada de comandos, encontrando-se assim à espera de instruções fornecidas pelo utilizador.

4.1.3 A Linguagem Maple

De modo a evitar alguns erros inesperados, pode utilizar-se o comando '**restart:**'. Este comando faz com que o *Maple* actue como se tivesse acabado de ser inicializado, ou seja, faz, entre outras coisas, com que os valores atribuídos às variáveis sejam eliminados.

As quatro operações aritméticas básicas são indicadas pelos símbolos '+' (adição), '−' (subtracção), '∗' (multiplicação) e '/' (divisão). Para a exponenciação usa-se o símbolo '^' e para a raiz quadrada recorre-se à função '**sqrt**'. Se numa expressão existir mais do que um operador, a ordem de realização dos cálculos é a usualmente utilizada na Matemática.

O *Maple* usualmente trabalha com os números de maneira exacta.

```
> 90*Pi/180;
```

$$\frac{\pi}{2}$$

```
> sqrt(2)+3^2-1;
```

$$\sqrt{2}+8$$

Para se obter uma aproximação decimal recorre-se à função '**evalf**' ("*evaluate using floating-point arithmetic*").

```
> evalf(90*Pi/180);
```

```
> evalf(sqrt(2)+3^2-1);
```

$$1.570796327$$

$$9.414213562$$

Tal como foi referido anteriormente, o *Maple* possui um vasto conjunto de funções. Contudo, quando este programa é iniciado nem todas ficam imediatamente acessíveis ao utilizador, sendo necessário activá-las para as poder usar. As funções estão agrupadas em módulos ("*packages*") de acordo com a sua finalidade ou área da Matemática onde são aplicadas. Assim sendo, para se ter acesso a uma dessas funções é preciso carregar o módulo que a contém, escrevendo o comando '**with**(*nome do módulo*):'. Ao carregar-se um módulo ficam disponíveis todas as funções nele contidas. Os módulos '**plots**', '**plottools**', '**ListTools**' e '**linalg**' são apenas alguns exemplos de *packages* do *Maple*.

Geralmente, uma qualquer expressão do *Maple* não precisa ter um nome atribuído. Contudo, esta situação altera-se quando há a necessidade de lhe fazer referência diversas vezes. O processo de atribuição de uma expressão a uma variável é realizado através do operador ':='. Embora o *Maple* ofereça uma grande liberdade na escolha de nomes para as variáveis, o utilizador tem que respeitar algumas restrições: o nome deve começar por uma letra, que depois pode ser seguida por outras letras, por dígitos ou por caracteres *underscore*; não podem ser usadas variáveis cujo nome seja coincidente com uma palavra reservada do *Maple* ou com funções pré-definidas ('**abs**', '**and**', '**from**', '**sin**', '**seq**', '**if**', '**Pi**', etc.). O *Maple* diferencia letras minúsculas de letras maiúsculas, pelo que 'x' e 'X' são consideradas variáveis diferentes.

O *Maple* permite que o utilizador crie as suas próprias funções. Um dos processos possíveis para definir uma função é através do operador de atribuição acabado de introduzir. É igualmente necessário inserir o operador "seta" ('->') logo após a variável da função. Este último operador é constituído pelo símbolo "menos" seguido do símbolo "maior que". A partir do momento em que uma função é definida passa a poder ser usada como qualquer uma das funções pré-definidas, ou seja, colocando qualquer expressão válida no seu argumento. Por exemplo, para se definir a função $f(x) = x^2 + \sqrt{x} - 1$ escreve-se o

comando

```
> f:=x->x^2+sqrt(x)-1;
```

$$f := x \rightarrow x^2 + \sqrt{x} - 1$$

É possível então determinar a imagem de qualquer objecto do domínio de f.

```
> f(x);
```

$$x^2 + \sqrt{x} - 1$$

```
> f(4);
```

$$17$$

Quando se pretende definir uma função descrita por ramos recorre-se à função '**piecewise**'. Com esta função também se podem usar os operadores lógicos '**and**', '**or**' e '**not**'. Por exemplo, para definir a função

$$g(x) = \begin{cases} x-1, & se\ x^2 > 4\ e\ x < 8 \\ 1-x, & se\ |x| \leq 2\ ou\ x \geq 8 \end{cases}$$

escreve-se o comando

```
> g:=x->piecewise(x^2>4 and x<8,x-1,abs(x)<=2 or x>=8,1-x);
```

$$g := x \rightarrow \text{piecewise}(4 < x^2 \textbf{ and } x < 8,\ x-1,\ |x| \leq 2 \textbf{ or } 8 \leq x,\ 1-x)$$

É possível determinar agora a imagem de qualquer objecto pertencente ao domínio da função g.

```
> g(7);
```

$$6$$

```
> g(2);
```

$$-1$$

O *Maple* possui diversos tipos de objectos, sendo as sequências, as listas, os conjuntos e os *arrays* alguns dos principais. Dado que vários comandos têm estes objectos como argumentos e como resultado, é necessário compreender a sua estrutura para se poder seleccionar e operar com os elementos desses objectos, e assim usar a linguagem *Maple* de

uma forma mais eficiente.

No *Maple*, uma sequência tem a forma

$$expr1,\ expr2, \ldots,\ exprn$$

```
> s1:=a,b,c;
```

$$s1 := a,\ b,\ c$$

```
> s2:=d,e,a;
```

$$s2 := d,\ e,\ a$$

No exemplo seguinte pode constatar-se que o objecto resultante da "junção" de duas sequências é ainda uma sequência.

```
> s:=s1,s2;
```

$$s := a,\ b,\ c,\ d,\ e,\ a$$

```
> whattype(s);
```

$$exprseq$$

Este último comando permite verificar qual o tipo de objecto com que se está a trabalhar. Neste caso, e tal como era esperado, o objecto é uma sequência (*exprseq*). A sequência vazia define-se através do comando

```
> s0:=NULL;
```

$$s0 :=$$

A partir deste momento vai usar-se com frequência o termo "operando" para designar um elemento de um objecto. Para seleccionar um operando de uma sequência escrever-se entre colchetes a posição do elemento, tal como se observa no exemplo que se segue.

```
> s[3];
```

$$c$$

Sobre as sequências pode ainda dizer-se que preservam a ordem dos seus elementos e que não eliminam as repetições que possam eventualmente existir.

No *Maple*, as listas são representadas como de um vector, tendo a forma

$$[expr1,\ expr2, \ldots,\ exprn]$$

```
> L:=[a,b,c,d,e,a];
```

$$L := [a,\ b,\ c,\ d,\ e,\ a]$$

Uma vez que já se tinha definido previamente a sequência s, um resultado igual ao anterior poderia ser obtido escrevendo o comando

```
> L:=[s];
```

$$L := [a,\ b,\ c,\ d,\ e,\ a]$$

```
> whattype(L);
```

$$list$$

As listas também preservam a ordem dos seus elementos e não eliminam as repetições que possam eventualmente existir. Vão agora ser apresentadas duas funções extremamente úteis para operar com listas. A primeira consiste na função '**nops**' que é o acrónimo de "*number of operands*". Tal como o nome sugere, esta função permite determinar o tamanho da lista, ou seja, o número de operandos que a compõem.

```
> nops(L);
```

$$6$$

A outra função é a '**op**'. Esta função *Maple* permite analisar a estrutura de uma expressão. O comando

```
> op(L);
```

$$a,\ b,\ c,\ d,\ e,\ a$$

devolve os operandos da expressão analisada na forma de uma sequência. Contudo, a função também devolve o n-ésimo operando da expressão se for acrescentado um primeiro argumento opcional. Por exemplo, o comando

```
> op(3,L);
```

$$c$$

devolve o terceiro elemento da lista L. Assim sendo, a função '**op**' pode ser usada em vez do

operador de selecção '[ ]'. Algumas das funções que operam sobre listas não estão activas aquando do arranque do *Maple*, sendo necessário carregar o *package* '**ListTools**' para se ter acesso a todas elas. Uma das funções que se torna activa é a '**Search**'. Esta função permite determinar a posição que um elemento dado ocupa numa lista. Se o elemento procurado não existir nessa lista, a função devolve o valor 0 (zero), caso contrário, devolve a posição correspondente à primeira ocorrência desse elemento. Os seguintes comandos permitem ilustrar o que acabou de ser exposto.

```
> with(ListTools):
> Search(e,L); Search(a,L); Search(r,L);
```

$$5$$

$$1$$

$$0$$

No *Maple*, os conjuntos têm a forma

$$\{expr1,\ expr2, \ldots,\ exprn\}$$

```
> C:={-4,3,1,0,1};
```

$$C := \{-4,\ 0,\ 1,\ 3\}$$

```
> whattype(C);
```

$$set$$

Este exemplo permite mostrar que nos conjuntos, ao contrário do que sucede com os dois tipos de objectos anteriores, os operandos podem ser reordenados e, além disso, são sempre eliminadas as repetições que ocorrem. Dado que estes objectos são inspirados nos conjuntos usados na Matemática, o *Maple* permite determinar a reunião, a intersecção e a diferença de conjuntos por intermédio das funções '**union**', '**intersect**' e '**minus**', respectivamente. As funções '**nops**' e '**op**' também podem ser utilizadas com conjuntos.

No *Maple*, os *arrays* podem ser vistos como tabelas com uma dimensão definida pelo utilizador.

```
> A:=array(1..6);
```

$$A := \text{array}(1..6, [\,])$$

O comando anterior criou um *array* unidimensional de comprimento 6, cujas entradas não foram definidas. Para fazer essa definição pode proceder-se do seguinte modo:

```
> A[1]:=a:  A[2]:=b:  A[3]:=c:  A[4]:=d:  A[5]:=e:  A[6]:=a:
```

A visualização do conteúdo de um *array* pode ser conseguida através da função '**print**', tal como se mostra no exemplo abaixo.

```
> print(A);
```

$$[a,\ b,\ c,\ d,\ e,\ a]$$

As *strings* e as pilhas ("*stacks*") são outros dois tipos de objectos que também tiveram uma grande importância na implementação dos procedimentos construídos no âmbito desta dissertação.

Uma *string* é uma sequência de caracteres que se encontra delimitada por aspas.

```
> s:="Exemplo de uma string";
```

$$s := \text{"Exemplo de uma string"}$$

```
> whattype(s);
```

string

Como se pode ver no exemplo seguinte, a concatenação de *strings* faz-se com recurso à função '**cat**' e o resultado desse processo continua a ser uma *string*.

```
> cat("str","ing");
```

"string"

Para se determinar o número de caracteres que compõem uma determinada *string* utiliza-se a função '**length**'.

```
> length("string");
```

Uma outra função usada com *strings* é a '**parse**'. Esta função analisa a cadeia de caracteres delimitada pelas aspas e transforma-a na expressão *Maple* correspondente. Tal como se pode constatar através do seguinte exemplo, essa expressão é apresentada mas o seu valor não é determinado.

```
> parse("sin(Pi/2)");
```

$$\sin(\frac{\pi}{2})$$

Para se forçar a avaliação da expressão é necessário usar a opção '**statement**'

```
> parse("sin(Pi/2)",statement);
```

$$1$$

ou escrever o comando

```
> evalf(parse("sin(Pi/2)"));
```

$$1.$$

Relativamente ao último objecto atrás referido e que ainda não foi explicitado, pode dizer-se que o procedimento '**SimpleStack**' é um construtor de pilhas de elementos, podendo esses elementos ser de diversos tipos. Uma expressão é do tipo '**Stack**' se for um objecto que possui pelo menos os métodos '**empty**', '**push**', '**pop**' e '**top**'.

```
> S:=SimpleStack();
```

$S :=$ **module**()
local *data*, *nitems*;
export *empty*, *push*, *pop*, *top*, *depth*, *map*, *toList*, *init*, *isStack*;
description "a simple stack";
end module

```
> type(S,'Stack');
```

true

Na sintaxe exigida para usar os diversos métodos surge o operador ':-'. Vai-se seguidamente explicar apenas a função dos métodos utilizados nos procedimentos construídos. O método '**empty**' permite verificar a existência de elementos na pilha. Se esta estiver vazia devolve

o valor "*true*", caso contrário devolve "*false*". Para inserir um elemento na pilha recorre-se ao método '**push**' e para remover o que está no topo (último elemento a ser introduzido) utiliza-se o '**pop**'. Por fim, o método '**init**' inicializa a pilha, esvaziando-a, enquanto que o '**toList**' constrói uma lista com todos os elementos nela presentes mas não a esvazia. Note-se que a ordem dos elementos dessa lista corresponde à ordem de entrada dos elementos para a pilha.

```
> S:-push("L"): S:-push("-"): S:-push("systems"):
> S:-toList();
```

["L", "-", "systems"]

```
> S:-empty();
```

false

```
> S:-pop();
```

"systems"

```
> S:-init():
> S:-empty();
```

true

O *Maple* possui várias funções para construção de gráficos, tendo cada uma delas várias opções disponíveis. Para se construir o gráfico em duas dimensões de uma função definida pela sua expressão algébrica recorre-se à função '**plot**', enquanto que para o caso tridimensional se utiliza a função '**plot3d**'. As opções inserem-se como igualdades do tipo 'opção=valor' e devem estar separadas por vírgulas. As opções '**axes**' (valores: *normal*, *none*, *boxed*, *frame*) e '**scaling**' (valores: *constrained*, *unconstrained*) actuam sobre os eixos do referencial e a opção '**thickness**' (valores: $0, 1, 2, \ldots$) afecta o próprio gráfico da função que se pretende representar. Estas são apenas algumas das muitas opções que o utilizador tem ao seu dispor. Um outro modo de fazer representações gráficas é através da função '**display**'. As três opções apresentadas anteriormente também estão disponíveis para esta função. Para se poder usar a função '**display**' é necessário carregar o módulo '**plots**'. Ao activar este módulo também fica disponível a função '**tubeplot**', a qual é usada para construir um tubo, de diâmetro variável, em torno de curvas tridimensionais. No módulo

'**plottools**' encontra-se, entre muitas outras, a função '**line**'. Esta função permite construir um segmento de recta entre dois pontos fornecidos.

Nos procedimentos que se encontram na Secção 4.2 surgem ainda outras três funções cujas funcionalidades convém explicitar. Para calcular a norma de um vector e para determinar o produto externo entre dois vectores utilizam-se as funções '**norm**' e '**crossprod**', respectivamente, que se encontram no módulo '**linalg**'. Por fim, para se obter um número aleatório recorre-se à função '**rand**'. A seguinte sequência de comandos permite ilustrar o modo de se obter aleatoriamente um número inteiro entre 1 e 100.

```
> gerar:=rand(1..100):
> gerar();
```

92

4.1.4 Programação em Maple

Para além das potencialidades ao nível do cálculo simbólico, o *Maple* é também uma linguagem de programação com bastantes semelhanças com outras linguagens conhecidas (*C*, *PASCAL*, *BASIC*, etc.).

As ferramentas universais da programação são:

- execução condicional;
- iteração;
- procedimentos.

A execução condicional é implementada por intermédio da estrutura '**if**'. A forma geral desta estrutura é:

if *condição 1* then

comandos 1

elif *condição 2* then

comandos 2

...

elif *condição n* then

comandos n

else

comandos por defeito

end if;

As diversas condições que podem surgir nesta estrutura são expressões lógicas avaliadas de verdadeiro (*true*) ou falso (*false*). O comando (ou bloco de comandos) que se encontra imediatamente após a palavra '**then**' só será executado se a condição que lhe está associada for verdadeira. Se todas as condições forem avaliadas como falsas então será executado o comando (ou bloco de comandos) se encontra depois da palavra '**else**'. A possibilidade de visualizar o resultado da execução dos comandos depende do terminador desta estrutura que foi utilizado (; ou :).

A iteração, isto é, a execução repetitiva de um comando ou bloco de comandos é realizada através dos ciclos '**for**' e '**while**'. A sintaxe geral do ciclo '**for**' é a seguinte:

for *variável* from *início* by *passo* to *fim* do

comando 1

comando 2

...

comando n

end do;

A variável de controlo *variável* é inicializada com o valor *início* e é incrementada, em cada iteração, por *passo*, até que o seu valor exceda o de *fim* (ou até se tornar mais pequeno que *fim*, no caso do incremento ser negativo). A opção '**from**' pode ser omitida quando se pretende que a variável de controlo seja inicializada com o valor 1. O mesmo pode ser feito com a opção '**by**', no caso em que o valor de *passo* é igual a 1. As expressões *início*, *passo* e *fim* podem tomar valores inteiros, racionais ou reais. O corpo principal de um ciclo '**for**' é constituído por um número arbitrário de comandos, sendo cada um deles executado para cada valor que a variável de controlo assumir no intervalo [*início*, *fim*]. Uma outra maneira de efectuar iterações é através do ciclo '**while**'. A sua sintaxe geral é:

while *condição* do

comando 1

comando 2

...

comando n

end do;

Se a condição for verdadeira então o corpo do ciclo é executado e a condição é novamente avaliada. O ciclo termina quando o valor da condição for *false*. Uma vez iniciado o ciclo, o seu corpo deve conter algum comando que modifique a condição, caso contrário o ciclo repetir-se-á indefinidamente. Tal como sucede na estrutura '**if**', também nestes dois ciclos acabados de apresentar, a visualização do resultado da execução dos comandos depende do terminador do ciclo (; ou :).

Relativamente aos procedimentos, pode dizer-se que são um caso mais geral de uma função

usual, cujo tipo de *output* pode ser bastante diversificado. De um modo simplificado, a sintaxe geral de um procedimento pode ser descrita da seguinte maneira:

nome:= proc(*arg1*::*tipo1*,...,*argn*::*tipon*)

local *variáveis*

global *variáveis*

option *opções*

comandos

end proc:

Para evitar uma linha de saída com a descrição do procedimento coloca-se ':' em vez de ';' após a expressão '**end proc**'. A invocação de um procedimento é realizada do mesmo modo que para uma função usual, ou seja, escrevendo o seu nome e colocando entre parêntesis curvos tantos "valores" quantos os argumentos do procedimento. Ao se invocar um procedimento são executados todos os comandos existentes no seu corpo, mas o resultado devolvido é relativo à última operação efectuada pelo *Maple* nesse procedimento. Contudo, pode forçar-se a saída de outros valores através da função '**print**'. Tal como se pode observar na sintaxe apresentada, as variáveis podem ser declaradas como locais ou globais. A diferença entre estes dois tipos de variáveis reside na região do programa onde elas são reconhecidas. Enquanto que uma variável local é reconhecida apenas dentro do procedimento, uma variável global é reconhecida mesmo fora deste. No caso de nada ser especificando a respeito de uma variável utilizada num procedimento, então o *Maple* declara-a automaticamente como sendo do tipo *local*. Os comandos que se seguem permitem ilustrar estas situações.

```
> a:=1:  b:=2:
> h:=proc(n::integer)
> global b:
```

```
> option remember:

> a:=n+3:

> print(a):

> b:=n-3:

> end proc:

   Warning, 'a' is implicitly declared local to procedure 'h'

> h(0);
```

$$3$$

```
> a;
```

$$-3$$

```
> b;
```

$$1$$

$$-3$$

Uma maneira de tornar um procedimento mais eficiente, principalmente os recursivos (não é o caso do exemplo anterior), é através da utilização da opção '**remember**'. Com esta opção economiza-se bastante tempo, uma vez que ela permite armazenar os resultados de uma invocação do procedimento e, posteriormente, recuperá-los (sem repetir os cálculos) quando o procedimento é invocado novamente com os mesmos argumentos. Uma propriedade extremamente útil de um procedimento é a possibilidade de fazer referência a si próprio e a outros procedimentos. A elaboração de procedimentos constitui uma das partes mais importantes da programação em *Maple*, uma vez que torna mais simples e elegante a execução de rotinas que envolvam um grande número de comandos, para além de permitir utilizar variáveis locais e restringir os argumentos ao tipo desejado.

Para quem estiver interessado em aprofundar os seus conhecimentos sobre o *Maple* pode consultar [2], [11] e [20]. Uma lista de referências mais completa pode ser encontrada em [24]. Pode ainda consultar a informação disponível no menu "*Help*" deste programa.

4.2 Procedimentos Utilizados

Para gerar algumas das imagens que se encontram ao longo da presente dissertação foram criados, em *Maple 10.04*, diversos procedimentos. Todos eles foram elaborados tendo em conta os fundamentos teóricos apresentados nos capítulos 2 e 3. Em cada um dos procedimentos 'Lsystem' é possível identificar as duas fases consideradas por Prusinkiewicz *et al* [18]: a fase de geração e a fase de interpretação. A parte inicial de cada um deles corresponde à fase de geração, enquanto que a parte final corresponde à fase de interpretação. Esta última fase é concretizada através da invocação do procedimento 'GRAFICO'. Para cada tipo de L-system apresentado nesta dissertação foi criado um procedimento 'Lsystem' e o correspondente procedimento 'GRAFICO'.

O "cabeçalho" de cada uma das folhas de trabalho *Maple* onde se encontram os procedimentos construídos é composto pelos seguintes comandos:

```
> restart:

> with(plots): with(plottools): with(ListTools):

> with(linalg):
```

O último destes módulos somente é necessário ser carregado no caso do procedimento que permite modelar o desenvolvimento do "esqueleto" de árvores tridimensionais através de um D0L-system paramétrico.

4.2.1 D0L-systems Ramificados

Os procedimentos que se encontram nesta secção permitem gerar imagens bidimensionais e tridimensionais através de D0L-systems ramificados.

A invocação destes procedimentos é feita através do comando

```
> Lsystem(AXIOMA,REGRAS,INCREMENTO,N);
```

onde AXIOMA, REGRAS e INCREMENTO correspondem ao axioma, ao conjunto de regras de substituição e ao incremento angular, respectivamente, de uma evolução de ordem N de um D0L-system ramificado. O estado inicial da "tartaruga" e o avanço d são definidos directamente no procedimento 'GRAFICO'. No caso bidimensional, essa definição é feita através da modificação dos valores atribuídos às variáveis PONTOA (posição), ANGULO (orientação) e SEGMENTO (avanço). Observe-se que a orientação deve estar em radianos. No caso tridimensional, a definição da posição e do avanço faz-se do mesmo modo que no caso anterior, no entanto, a definição da orientação é feita através da modificação dos valores atribuídos aos vectores vH ("*heading*"), vL ("*left*") e vU ("*up*").

Tal como se pode constatar nos procedimentos 'Lsystem' apresentados nesta secção, o argumento *axioma* é do tipo *string*, *regras* é uma lista e n é um número inteiro. Tem-se ainda que *incremento* é um número real que deve ser estar em graus. Note-se que cada um dos operandos da lista que corresponde ao conjunto de regras é uma *string*. Essa lista tem o formato $[P_1, P_2, \ldots, P_m]$, onde para cada i se tem que P_i é uma sequência com dois operandos "a_i","s_i", sendo a_i e s_i o antecessor e o sucessor da regra P_i, respectivamente.

O caso bidimensional é então assegurado pelo procedimento 'Lsystem' que se segue.

```
> GRAFICO:=proc(s::string,incremento)
> local PONTOA, ANGULO, SEGMENTO, ANGSTEP, GRAFICOTEMP, MEM,
> i, PONTOB, VOLTAR:
> PONTOA:=[0,0]:  ANGULO:=0.0:  SEGMENTO:=1.0:
> ANGSTEP:=evalf(incremento*Pi/180):
> GRAFICOTEMP:=NULL: MEM:=SimpleStack():
> for i from 1 to length(s) do
>   if (s[i]="F") then
```

```
>       PONTOB:=PONTOA+SEGMENTO*[cos(ANGULO),sin(ANGULO)]:
>       GRAFICOTEMP:=GRAFICOTEMP,line(PONTOA,PONTOB):
>       PONTOA:=PONTOB:
>     end if:
>     if (s[i]="f") then
>       PONTOA:=PONTOA+SEGMENTO*[cos(ANGULO),sin(ANGULO)]:
>     end if:
>     if (s[i]="+") then ANGULO:=ANGULO+ANGSTEP: end if:
>     if (s[i]="-") then ANGULO:=ANGULO-ANGSTEP: end if:
>     if (s[i]="[") then MEM:-push(PONTOA,ANGULO): end if:
>     if (s[i]="]") then
>       VOLTAR:=MEM:-pop():
>       PONTOA:=VOLTAR[1]:
>       ANGULO:=VOLTAR[2]:
>     end if:
>   end do:
>   GRAFICOTEMP:
>   end proc:
```

```
> Lsystem:=proc(axioma::string,regras::list,incremento,n::integer)
> local LTEMP, LTEMP2, A, i, j, k, indice:
> option remember:
> LTEMP:=axioma:  LTEMP2:=SimpleStack():  A:=NULL:
> for i from 1 to (nops(regras)-1) by 2 do
>   A:=A,regras[i]:
> end do:
> for j from 1 to n do
>   LTEMP2:-init():
>   for k from 1  to length(LTEMP) do
>     indice:=Search(LTEMP[k],[A]):
>     if (indice=0) then
>       LTEMP2:-push(LTEMP[k]):
>     else
>       LTEMP2:-push(regras[2*indice]):
>     end if:
>   end do:
>   LTEMP:="":
>   while (not LTEMP2:-empty()) do
>     LTEMP:=cat(LTEMP2:-pop(),LTEMP):
>   end do:
```

```
> end do:
> display(GRAFICO(LTEMP,incremento),axes=none,scaling=constrained):
> end proc:
```

Note-se que é necessário ter algum cuidado ao introduzir os argumentos do procedimento, uma vez que este é sensível a maiúsculas e minúsculas.

Seguidamente apresenta-se o caso tridimensional.

```
> GRAFICO:=proc(s::string,incremento)
> local PONTOA, vH, vL, vU, SEGMENTO, ANGSTEP, phi, GRAFICOTEMP,
> MEM, i, PONTOB, vHtemp, vLtemp, vUtemp, VOLTAR:
> PONTOA:=[0,0,0]:  vH:=[1,0,0]:  vL:=[0,1,0]:  vU:=[0,0,1]:
> SEGMENTO:=1.0:  ANGSTEP:=evalf(incremento*Pi/180):  phi:=ANGSTEP:
> GRAFICOTEMP:=NULL: MEM:=SimpleStack():
> for i from 1 to length(s) do
>   if (s[i]="F") then
>     PONTOB:=PONTOA+SEGMENTO*vH:
>     GRAFICOTEMP:=GRAFICOTEMP,line(PONTOA,PONTOB):
>     PONTOA:=PONTOB:
>   end if:
>   if (s[i]="f") then
>     PONTOA:=PONTOA+SEGMENTO*vH:
>   end if:
```

```
>    if (s[i]="+") then
>      vHtemp:=[vH[1]*cos(phi)+vL[1]*sin(phi),
> vH[2]*cos(phi)+vL[2]*sin(phi),vH[3]*cos(phi)+vL[3]*sin(phi)]:
>      vLtemp:=[-vH[1]*sin(phi)+vL[1]*cos(phi),
> -vH[2]*sin(phi)+vL[2]*cos(phi),-vH[3]*sin(phi)+vL[3]*cos(phi)]:
>      vH:=vHtemp:
>      vL:=vLtemp:
>      vU:=vU:
>    end if:
>    if (s[i]="-") then
>      vHtemp:=[vH[1]*cos(-phi)+vL[1]*sin(-phi),
> vH[2]*cos(-phi)+vL[2]*sin(-phi),vH[3]*cos(-phi)+vL[3]*sin(-phi)]:
>      vLtemp:=[-vH[1]*sin(-phi)+vL[1]*cos(-phi),
> -vH[2]*sin(-phi)+vL[2]*cos(-phi),-vH[3]*sin(-phi)+vL[3]*cos(-phi)]:
>      vH:=vHtemp:
>      vL:=vLtemp:
>      vU:=vU:
>    end if:
>    if (s[i]="&") then
>      vHtemp:=[vH[1]*cos(phi)-vU[1]*sin(phi),
> vH[2]*cos(phi)-vU[2]*sin(phi),vH[3]*cos(phi)-vU[3]*sin(phi)]:
```

```
>      vUtemp:=[vH[1]*sin(phi)+vU[1]*cos(phi),
> vH[2]*sin(phi)+vU[2]*cos(phi),vH[3]*sin(phi)+vU[3]*cos(phi)]:
>      vH:=vHtemp:
>      vL:=vL:
>      vU:=vUtemp:
>    end if:
>    if (s[i]="^") then
>      vHtemp:=[vH[1]*cos(-phi)-vU[1]*sin(-phi),
> vH[2]*cos(-phi)-vU[2]*sin(-phi),vH[3]*cos(-phi)-vU[3]*sin(-phi)]:
>      vUtemp:=[vH[1]*sin(-phi)+vU[1]*cos(-phi),
> vH[2]*sin(-phi)+vU[2]*cos(-phi),vH[3]*sin(-phi)+vU[3]*cos(-phi)]:
>      vH:=vHtemp:
>      vL:=vL:
>      vU:=vUtemp:
>    end if:
>    if (s[i]="?") then
>      vLtemp:=[vL[1]*cos(phi)-vU[1]*sin(phi),
> vL[2]*cos(phi)-vU[2]*sin(phi),vL[3]*cos(phi)-vU[3]*sin(phi)]:
>      vUtemp:=[vL[1]*sin(phi)+vU[1]*cos(phi),
> vL[2]*sin(phi)+vU[2]*cos(phi),vL[3]*sin(phi)+vU[3]*cos(phi)]:
>      vH:=vH:
```

```
>      vL:=vLtemp:
>      vU:=vUtemp:
>    end if:
>    if (s[i]="/") then
>      vLtemp:=[vL[1]*cos(-phi)-vU[1]*sin(-phi),
> vL[2]*cos(-phi)-vU[2]*sin(-phi),vL[3]*cos(-phi)-vU[3]*sin(-phi)]:
>      vUtemp:=[vL[1]*sin(-phi)+vU[1]*cos(-phi),
> vL[2]*sin(-phi)+vU[2]*cos(-phi),vL[3]*sin(-phi)+vU[3]*cos(-phi)]:
>      vH:=vH:
>      vL:=vLtemp:
>      vU:=vUtemp:
>    end if:
>    if (s[i]="|") then
>      vHtemp:=[vH[1]*cos(Pi)+vL[1]*sin(Pi),
> vH[2]*cos(Pi)+vL[2]*sin(Pi),vH[3]*cos(Pi)+vL[3]*sin(Pi)]:
>      vLtemp:=[-vH[1]*sin(Pi)+vL[1]*cos(Pi),
> -vH[2]*sin(Pi)+vL[2]*cos(Pi),-vH[3]*sin(Pi)+vL[3]*cos(Pi)]:
>      vH:=vHtemp:
>      vL:=vLtemp:
>      vU:=vU:
>    end if:
```

```
>    if (s[i]="[") then MEM:-push(PONTOA,vH,vL,vU): end if:
>    if (s[i]="]") then
>      VOLTAR:=MEM:-pop():
>      PONTOA:=VOLTAR[1]:
>      vH:=VOLTAR[2]:
>      vL:=VOLTAR[3]:
>      vU:=VOLTAR[4]:
>    end if:
> end do:
> GRAFICOTEMP:
> end proc:

> Lsystem:=proc(axioma::string,regras,incremento,n::integer)
> local LTEMP, LTEMP2, A, i, j, k, indice:
> option remember:
> LTEMP:=axioma:  LTEMP2:=SimpleStack():  A:=NULL:
> for i from 1 to (nops(regras)-1) by 2 do
>    A:=A,regras[i]:
> end do:
> for j from 1 to n do
>    LTEMP2:-init():
```

```
>    for k from 1  to length(LTEMP) do
>      indice:=Search(LTEMP[k],[A]):
>      if (indice=0) then
>        LTEMP2:-push(LTEMP[k]):
>      else
>        LTEMP2:-push(regras[2*indice]):
>      end if:
>    end do:
>    LTEMP:="":
>    while (not LTEMP2:-empty()) do
>      LTEMP:=cat(LTEMP2:-pop(),LTEMP):
>    end do:
> end do:
> display(GRAFICO(LTEMP,incremento),axes=none,scaling=constrained,
> thickness=2):
> end proc:
```

Tal como acontece no caso bidimensional, é necessário ter algum cuidado aquando da introdução dos argumentos, uma vez que o procedimento é sensível a maiúsculas e minúsculas. Note-se que no procedimento 'GRAFICO' houve a necessidade de usar o símbolo '&' em vez de '∨' e o símbolo '?' em vez de '\'.

4.2.2 0L-systems Estocásticos

O procedimento que se encontra nesta secção permite gerar imagens bidimensionais através de um 0L-system estocástico.

A invocação deste procedimento é feita através do comando

```
> Lsystem(AXIOMA,REGRAS,PROB,INCREMENTO,N);
```

Tudo o que foi dito na secção anterior sobre os D0L-systems ramificados 2D também se aplica a este procedimento, no entanto, nem tudo é igual. A diferença entre ambos reside apenas na necessidade de introduzir uma lista PROB de valores numéricos reais que correspondem à probabilidade associada a cada regra de substituição. Tem-se assim que $PROB[i]$ é a probabilidade associada à regra P_i. Note-se ainda que o somatório de todos os $PROB[i]$ tem que ser igual a 1. Embora o procedimento que se segue seja extremamente simples e pouco generalista, visto que todas as regras têm que ter o mesmo antecessor, permite mesmo assim ilustrar o funcionamento e a utilidade deste tipo de L-systems.

```
> GRAFICO:=proc(s::string,incremento)
> local PONTOA, ANGULO, SEGMENTO, ANGSTEP, GRAFICOTEMP, MEM,
> i, PONTOB, VOLTAR:
> PONTOA:=[0,0]:  ANGULO:=Pi/2:  SEGMENTO:=1.0:
> ANGSTEP:=evalf(incremento*Pi/180):
> GRAFICOTEMP:=NULL: MEM:=SimpleStack():
> for i from 1 to length(s) do
>   if (s[i]="F") then
>     PONTOB:=PONTOA+SEGMENTO*[cos(ANGULO),sin(ANGULO)]:
>     GRAFICOTEMP:=GRAFICOTEMP,line(PONTOA,PONTOB):
```

```
>       PONTOA:=PONTOB:
>     end if:
>     if (s[i]="f") then
>       PONTOA:=PONTOA+SEGMENTO*[cos(ANGULO),sin(ANGULO)]:
>     end if:
>     if (s[i]="+") then ANGULO:=ANGULO+ANGSTEP: end if:
>     if (s[i]="-") then ANGULO:=ANGULO-ANGSTEP: end if:
>     if (s[i]="[") then MEM:-push(PONTOA,ANGULO): end if:
>     if (s[i]="]") then
>       VOLTAR:=MEM:-pop():
>       PONTOA:=VOLTAR[1]:
>       ANGULO:=VOLTAR[2]:
>     end if:
>   end do:
>   GRAFICOTEMP:
>   end proc:

>   Lsystem:=proc(axioma::string,regras::list,prob::list,incr,n::integer)
>   local LTEMP, LTEMP2, A, roll, m, S, soma, i, j, k, indice, ind:
>   LTEMP:=axioma:  LTEMP2:=SimpleStack():
>   A:=NULL: roll:=rand(1..100):  m:=nops(prob):
```

```
> S:=array(1..m+1): S[1]:=0: soma:=0:
> for i from 1 to m do
>   soma:=soma+prob[i]:
>   S[i+1]:=soma:
> end do:
> for j from 1 to (nops(regras)-1) by 2 do
>   A:=A,regras[j]:
> end do:
> for i from 1 to n do
>   LTEMP2:-init():
>   for j from 1 to length(LTEMP) do
>     indice:=Search(LTEMP[j],[A]):
>     if (indice=0) then
>       LTEMP2:-push(LTEMP[j]):
>     else
>       ind:=roll():
>       for k from 1 to m do
>         if (S[k]<ind and ind<=S[k+1]) then
>           LTEMP2:-push(regras[2*k]):
>         end if:
>       end do:
```

```
>       end if:
>     end do:
>     LTEMP:="":
>     while (not LTEMP2:-empty()) do
>       LTEMP:=cat(LTEMP2:-pop(),LTEMP):
>     end do:
>   end do:
>   display(GRAFICO(LTEMP,incr),axes=none,scaling=constrained):
>   end proc:
```

4.2.3 D0L-systems Paramétricos

Os procedimentos que se encontram nesta secção permitem gerar imagens bidimensionais e tridimensionais através de D0L-systems paramétricos.

A invocação destes procedimentos é feita através do comando

```
> Lsystem(AXIOMA,N);
```

onde AXIOMA corresponde ao axioma de uma evolução de ordem N de um D0L-system paramétrico. O estado inicial da "tartaruga" e o avanço são definidos directamente no procedimento 'GRAFICO', tal como foi apresentado na Secção 4.2.1 para os D0L-systems ramificados.

Dado que um PL-system opera com módulos formados por um símbolo do alfabeto ao qual está associada uma quantidade arbitrária de parâmetros numéricos, há a necessidade de definir essa estrutura em linguagem *Maple*. Assim sendo, um módulo é uma lista formada por dois elementos (uma *string* e uma lista de parâmetros), ou seja, tem a forma

$$[\text{“A”},[a_1, a_2, \ldots, a_m]]$$

onde $A \in V$ e $a_1, a_2, \ldots, a_m \in \mathbb{R}$.

Tal como se pode constatar nos procedimentos 'Lsystem' apresentados nesta secção, o argumento *axioma* é uma lista (de módulos) e n é um número inteiro. Note-se que as regras de substituição são definidas na folha de trabalho sob a forma de funções. Essas funções têm o formato

$$\text{A:=L->}[modseq]$$

onde 'A' é uma letra do alfabeto do L-system, 'L' é a variável (lista de parâmetros) e *modseq* é uma sequência de módulos. Mais uma vez, para que os procedimentos que se seguem funcionem apropriadamente, é necessário ter algum cuidado ao introduzir os argumentos devido à sua sensibilidade a maiúsculas e minúsculas.

O caso bidimensional é então assegurado pelo procedimento 'Lsystem' que se segue.

```
> GRAFICO:=proc(s::list)
> local PONTOA, ANGULO, GRAFICOTEMP, MEM, i, PONTOB, VOLTAR:
> PONTOA:=[0,0]:  ANGULO:=0.0:
> GRAFICOTEMP:=NULL: MEM := SimpleStack():
> for i from 1 to nops(s) do
>   if (s[i][1]="F") then
>     PONTOB:=PONTOA+(s[i][2][1])*[cos(ANGULO),sin(ANGULO)]:
>     GRAFICOTEMP:=GRAFICOTEMP,line(PONTOA,PONTOB):
>     PONTOA:=PONTOB:
>   end if:
```

```
>    if (s[i][1]="f") then
>      PONTOA:=PONTOA+(s[i][2][1])*[cos(ANGULO),sin(ANGULO)]:
>    end if:
>    if (s[i][1]="+") then
>      ANGULO:=ANGULO+evalf(s[i][2][1]*Pi/180):
>    end if:
>    if (s[i][1]="-") then
>      ANGULO:=ANGULO-evalf(s[i][2][1]*Pi/180):
>    end if:
>    if (s[i][1]="[") then
>      MEM:-push(PONTOA,ANGULO):
>    end if:
>    if (s[i][1]="]") then
>      VOLTAR:=MEM:-pop():
>      PONTOA:=VOLTAR[1]:
>      ANGULO:=VOLTAR[2]:
>    end if:
> end do:
> GRAFICOTEMP:
> end proc:
```

```
> Lsystem:=proc(axioma::list,n::integer)
> local LTEMP, LTEMP2, i, j, funcTemp:
> option remember:
> LTEMP:=axioma:  LTEMP2:=SimpleStack():
> for i from 1 to n do
>   LTEMP2:-init():
>   for j from 1  to nops(LTEMP) do
>     if (LTEMP[j][1]<"A" or LTEMP[j][1]>"Z") then
>       LTEMP2:-push(LTEMP[j]):
>     else
>       funcTemp:=parse(LTEMP[j][1]):
>       LTEMP2:-push(op(funcTemp(LTEMP[j][2]))):
>     end if:
>   end do:
>   LTEMP:=LTEMP2:-toList():
> end do:
> display(GRAFICO(LTEMP),axes=none,scaling=constrained):
> end proc:
```

Por fim, apresenta-se o caso tridimensional, no qual se recorreu à função '**tubeplot**' para construir o "esqueleto" das árvores. Note-se que no procedimento 'GRAFICO' houve a necessidade de usar o símbolo '&' em vez de '$\vee$' e o símbolo '?' em vez de '\'.

```
> GRAFICO:=proc(s::list)
> local PONTOA, vH, vL, vU, diam, GRAFICOTEMP, MEM, i, PONTOB, v, phi,
> vHtemp, vLtemp, vUtemp, VOLTAR:
> PONTOA:=[0,0,0]:  vH:=[1,0,0]:  vL:=[0,1,0]:  vU:=[0,0,1]:
> diam:=0.01:  GRAFICOTEMP:=NULL: MEM:=SimpleStack():
> for i from 1 to nops(s) do
>   if nops(s[i])>1 then
>     if (op(s[i][1])="F") then
>       PONTOB:=PONTOA+(s[i][2][1])*vH:
>       v:=PONTOB-PONTOA:
>       GRAFICOTEMP:=GRAFICOTEMP,tubeplot([PONTOA[1]+v[1]*t,
> PONTOA[2]+v[2]*t,PONTOA[3]+v[3]*t],t=0..1,radius=diam,tubepoints=30):
>       PONTOA:=PONTOB:
>     end if:
>     if (op(s[i][1])="f") then
>       PONTOA:=PONTOA+(s[i][2][1])*vH:
>     end if:
>     if (op(s[i][1])="+") then phi:=evalf(s[i][2][1]*Pi/180):
```

```
>        vHtemp:=[vH[1]*cos(phi)+vL[1]*sin(phi),
> vH[2]*cos(phi)+vL[2]*sin(phi),vH[3]*cos(phi)+vL[3]*sin(phi)]:
>        vLtemp:=[-vH[1]*sin(phi)+vL[1]*cos(phi),
> -vH[2]*sin(phi)+vL[2]*cos(phi),-vH[3]*sin(phi)+vL[3]*cos(phi)]:
>        vH:=vHtemp:
>        vL:=vLtemp:
>        vU:=vU:
>      end if:
>      if (op(s[i][1])="-") then phi:=evalf(s[i][2][1]*Pi/180):
>        vHtemp:=[vH[1]*cos(-phi)+vL[1]*sin(-phi),
> vH[2]*cos(-phi)+vL[2]*sin(-phi),vH[3]*cos(-phi)+vL[3]*sin(-phi)]:
>        vLtemp:=[-vH[1]*sin(-phi)+vL[1]*cos(-phi),
> -vH[2]*sin(-phi)+vL[2]*cos(-phi),-vH[3]*sin(-phi)+vL[3]*cos(-phi)]:
>        vH:=vHtemp:
>        vL:=vLtemp:
>        vU:=vU:
>      end if:
>      if (op(s[i][1])="&") then phi:=evalf(s[i][2][1]*Pi/180):
>        vHtemp:=[vH[1]*cos(phi)-vU[1]*sin(phi),
> vH[2]*cos(phi)-vU[2]*sin(phi),vH[3]*cos(phi)-vU[3]*sin(phi)]:
>        vUtemp:=[vH[1]*sin(phi)+vU[1]*cos(phi),
```

```
> vH[2]*sin(phi)+vU[2]*cos(phi),vH[3]*sin(phi)+vU[3]*cos(phi)]:
>         vH:=vHtemp:
>         vL:=vL:
>         vU:=vUtemp:
>       end if:
>       if (op(s[i][1])="^") then phi:=evalf(s[i][2][1]*Pi/180):
>         vHtemp:=[vH[1]*cos(-phi)-vU[1]*sin(-phi),
> vH[2]*cos(-phi)-vU[2]*sin(-phi),vH[3]*cos(-phi)-vU[3]*sin(-phi)]:
>         vUtemp:=[vH[1]*sin(-phi)+vU[1]*cos(-phi),
> vH[2]*sin(-phi)+vU[2]*cos(-phi),vH[3]*sin(-phi)+vU[3]*cos(-phi)]:
>         vH:=vHtemp:
>         vL:=vL:
>         vU:=vUtemp:
>       end if:
>       if (op(s[i][1])="?") then phi:=evalf(s[i][2][1]*Pi/180):
>         vLtemp:=[vL[1]*cos(phi)-vU[1]*sin(phi),
> vL[2]*cos(phi)-vU[2]*sin(phi),vL[3]*cos(phi)-vU[3]*sin(phi)]:
>         vUtemp:=[vL[1]*sin(phi)+vU[1]*cos(phi),
> vL[2]*sin(phi)+vU[2]*cos(phi),vL[3]*sin(phi)+vU[3]*cos(phi)]:
>         vH:=vH:
>         vL:=vLtemp:
```

```
>        vU:=vUtemp:
>      end if:
>      if (op(s[i][1])="/") then phi:=evalf(s[i][2][1]*Pi/180):
>        vLtemp:=[vL[1]*cos(-phi)-vU[1]*sin(-phi),
> vL[2]*cos(-phi)-vU[2]*sin(-phi),vL[3]*cos(-phi)-vU[3]*sin(-phi)]:
>        vUtemp:=[vL[1]*sin(-phi)+vU[1]*cos(-phi),
> vL[2]*sin(-phi)+vU[2]*cos(-phi),vL[3]*sin(-phi)+vU[3]*cos(-phi)]:
>        vH:=vH:
>        vL:=vLtemp:
>        vU:=vUtemp:
>      end if:
>      if (op(s[i][1])="!") then diam:=s[i][2][1]:  end if:
>    else
>      if (op(s[i][1])="|") then
>        vHtemp:=[vH[1]*cos(Pi)+vL[1]*sin(Pi),
> vH[2]*cos(Pi)+vL[2]*sin(Pi),vH[3]*cos(Pi)+vL[3]*sin(Pi)]:
>        vLtemp:=[-vH[1]*sin(Pi)+vL[1]*cos(Pi),
> -vH[2]*sin(Pi)+vL[2]*cos(Pi),-vH[3]*sin(Pi)+vL[3]*cos(Pi)]:
>        vH:=vHtemp:
>        vL:=vLtemp:
>        vU:=vU:
```

```
>       end if:
>       if (op(s[i][1])="@") then
>         vH:=vH:
>         vL:=[-vH[2],vH[1],0]/norm([-vH[2],vH[1],0],2):
>         vU:=crossprod(vH,vL):
>       end if:
>       if (op(s[i][1])="[") then MEM:-push(PONTOA,vH,vL,vU): end if:
>       if (op(s[i][1])="]") then
>         VOLTAR:=MEM:-pop():
>         PONTOA:=VOLTAR[1]:
>         vH:=VOLTAR[2]:
>         vL:=VOLTAR[3]:
>         vU:=VOLTAR[4]:
>       end if:
>     end if:
> end do:
> GRAFICOTEMP:
> end proc:

> Lsystem:=proc(axioma::list,n::integer)
> local LTEMP, LTEMP2, i, j, funcTemp:
```

```
> option remember:
> LTEMP:=axioma:  LTEMP2:=SimpleStack():
> for i from 1 to n do
>   LTEMP2:-init():
>   for j from 1  to nops(LTEMP) do
>     if (LTEMP[j][1]<"A" or LTEMP[j][1]>"Z") then
>       LTEMP2:-push(LTEMP[j]):
>     else
>       funcTemp:=parse(LTEMP[j][1]):
>       LTEMP2:-push(op(funcTemp(LTEMP[j][2]))):
>     end if:
>   end do:
>   LTEMP:=LTEMP2:-toList():
> end do:
> display(GRAFICO(LTEMP),axes=none,scaling=constrained):
> end proc:
```

Capítulo 5

Conclusão

Um modo de descrever fenómenos de desenvolvimento é através da utilização de mecanismos de reescrita. Os L-systems, que foram originalmente criados para modelar o desenvolvimento de organismos multicelulares simples, são um exemplo de um mecanismo de reescrita que opera sobre palavras. Este formalismo permite simular as divisões ou os estados das células de um organismo através da aplicação em paralelo das regras de substituição, ocorrendo assim, em cada iteração, uma substituição simultânea de todas as letras que constituem uma determinada palavra. A introdução do conceito da interpretação gráfica das palavras geradas por um L-system permitiu a visualização dos modelos, tornando possível modelar organismos com uma estrutura mais complexa.

De modo a simular o desenvolvimento de organismos com uma estrutura ramificada é necessário que exista uma descrição matemática dessas estruturas, para além de métodos e modelos para as gerar. Para modelar as estruturas ramificadas foi introduzido o conceito de palavra com colchetes. O colchete esquerdo indica o vértice onde se vai formar o novo ramo, enquanto que o correspondente colchete direito vai terminar essa ramificação, ficando entre eles todos os elementos que constituem o ramo. A natureza discreta da aplicação das regras de substituição condiciona o poder de modelação dos L-systems. Com o objectivo de colmatar algumas das limitações, foi feita a associação entre os símbolos do

alfabeto e um número arbitrário de parâmetros numéricos, surgindo assim os L-systems paramétricos como uma extensão do conceito original deste formalismo. Esses parâmetros podem ser usados durante a fase de interpretação para quantificar facilmente algumas características, tais como o comprimento ou o diâmetro dos segmentos e a amplitude dos ângulos de ramificação. As regras de substituição podem agora ter incorporadas expressões aritméticas que permitem a actualização do valor dos parâmetros ao longo do processo de substituição, e expressões lógicas que permitem seleccionar a regra a ser aplicada. Os L-systems podem ser classificados como sendo sensíveis ao contexto ou livres de contexto, consoante se tenha ou não em consideração os caracteres adjacentes ao qual uma determinada regra de substituição é aplicada. Também podem ser classificados como estocásticos quando a aplicação das regras depende da probabilidade que lhes foi atribuída. A simulação computacional de padrões de ramificação já têm uma história relativamente longa. Nesta dissertação foram abordados apenas os modelos de Honda e de Aono e Kunii, que usam segmentos rectilíneos, de diâmetro constante ou variável, para representar apenas o "esqueleto das árvores".

O *Maple* faz parte de uma família já relativamente vasta de ambientes científicos designados por sistemas de Álgebra computacional. Recorrendo a esta linguagem, foram elaborados diversos procedimentos que permitem ilustrar o funcionamento dos diferentes tipos de L-systems estudados. Devido às limitações do hardware utilizado e à memória requerida pelo *Maple*, não foi possível apresentar imagens resultantes de evoluções de ordem muito elevada desses L-systems.

De modo a ser possível fazer uma modelação mais realista, a extensão dos L-systems paramétricos para modelar os efeitos ambientais e a incorporação das folhas é um passo lógico para futura pesquisa. Uma vez que o *Maple* pode não ser acessível a todos, esta pesquisa futura pode passar ainda pela utilização de software *open source* para realizar um trabalho análogo ao que foi apresentado no Capítulo 4 desta dissertação.

Bibliografia

[1] H. Abelson, A. diSessa (1986), *Turtle Geometry: The Computer as a Medium for Exploring Mathematics*, MIT Press.

[2] D. Almeida (2004), *O Maple em Sistemas Dinâmicos - uma abordagem para o Ensino Secundário*, Tese de Mestrado em Ensino da Matemática, Faculdade de Ciências da Universidade do Porto.

[3] N. Crato (2004), *De Fi a Fibonacci*, Revista Actual: Ciência - Expresso de 09 de Outubro de 2004 (disponível em: http://pascal.iseg.utl.pt/~ncrato/personal.html).

[4] U. Dudley (1992), *Mathematical Cranks*, The Mathematical Association of America, Washington D. C., 137–158.

[5] L. Endo (2004), *Simulação de Mini-Ecossistemas Vegetais em Tempo Real*, Tese de Mestrado em Ciência da Computação, Instituto de Matemática e Estatística da Universidade de São Paulo, 24–28 (disponível em: www.ime.usp.br/dcc/posgrad/teses/endo.pdf).

[6] J. Hanan (1992), *Parametric L-systems and their Application to the Modelling and Visualization of Plants*, Ph.D. dissertation, Department of Computer Science of the University of Calgary (disponível em: http://algorithmicbotany.org/papers/).

[7] Y. Lima, F. Gomes (1997), *Xeq Mat - Matemática 11º ano*, Editorial O Livro, Lisboa, 320–325.

[8] A. Lindenmayer (1968), Mathematical models for cellular interactions in development I. Filaments with one-sided inputs, *Journal of Theoretical Biology* Vol. 18, 280–299 (disponível em: http://www.sciencedirect.com/).

[9] B. Mandelbrot (1982), *The fractal geometry of nature*, W. H. Freeman, San Francisco, 39–40.

[10] H. Noser, D. Thalmann, R. Turner (1992), *Animation based on the Interaction of L-systems with Vector Force Fields* (disponível em: http://vrlab.epfl.ch/).

[11] R. Portugal (1992), *Introdução ao Maple*, Laboratório Nacional de Computação Científica, Petrópolis (disponível em http://www.lncc.br/~portugal/curso.pdf).

[12] P. Prusinkiewicz (1986), Graphical applications of L-systems, *Proceedings of Graphics Interface '86*, 447–453 (disponível em: http://algorithmicbotany.org/papers/).

[13] P. Prusinkiewicz (1986), Score generation with L-systems, *Proceedings of the 1986 International Computer Music Conference*, 455–457 (disponível em: http://algorithmicbotany.org/papers/).

[14] P. Prusinkiewicz, A. Lindenmayer (1990), *The Algorithmic Beauty of Plants*, Springer-Verlag, New York (disponível em: http://algorithmicbotany.org/papers/).

[15] P. Prusinkiewicz (1993), Modeling and Visualization of Biological Structures, *Proceeding of Graphics Interface '93*, 128–137 (disponível em: http://www.graphicsinterface.org/pre1996/93-Prusinkiewicz.pdf).

[16] P. Prusinkiewicz *et al*(1996), L-systems: from the theory to visual models of plants, In M. T. Michalewicz (ed.), *Plants to ecosystems: Advances in computational life sciences* I, 1–27 (disponível em: http://algorithmicbotany.org/papers/).

[17] P. Prusinkiewicz *et al* (1999), An L-system-based plant modeling language, *In Proceedings of AGTIVE 1999, Lecture Notes in Computer Science 1779*, 395–410 (disponível em: http://algorithmicbotany.org/papers/).

[18] P. Prusinkiewicz *et al* (1999), Interactive Arrangement of Botanical L-System Models, *Proceedings of the 1999 Symposium on Interactive 3D Graphics*, 175–182 (disponível em: http://algorithmicbotany.org/papers/).

[19] P. Prusinkiewicz (2003), Introdution to Modeling with L-systems, *L-systems and Beyond - SIGGRAPH 2003 Course Notes*, part 1: 1–26 (disponível em: http://algorithmicbotany.org/papers/).

[20] A. Sousa (2004), *MAPLETS - Modelos Interactivos no Ensino da Matemática*, Tese de Mestrado em Ensino da Matemática, Faculdade de Ciências da Universidade do Porto.

[21] D. Wright (1996), *Dynamical Systems and Fractals Lecture Notes* (disponível em: http://www.math.okstate.edu/mathdept/dynamics/lecnotes/node1.html).

[22] http://web.mit.edu/afs/athena.mit.edu/software/maple/www/home.html#refathena (consultado em 15/01/2008)

[23] http://www.indiana.edu/~statmath/math/maple/overview.html (consultado em 15/01/2008)

[24] http://www.maplesoft.com/ (consultado em 15/01/2008)

[25] http://www.scg.uwaterloo.ca/history.shtml (consultado em 15/01/2008)