Esteban Walter Gonzalez Clua

Modelagem Procedimental para Visualização de Elementos da Natureza

Dissertação de Mestrado

Departamento de Informática

Pontifícia Universidade Católica do Rio de Janeiro

EstebanWalter Gonzalez Clua

Modelagem Procedimental de Elementos da Natureza

Dissertação apresentada ao Departamento de Informática da Puc-Rio como parte dos requisitos para obtenção do título de Mestre em Informática: Ciência da Computação.

Orientador: Marcelo Gattass
Coorientador: Marcelo Dreux

Departamento de Informática
Pontifícia Universidade Católica do Rio de Janeiro

Rio de Janeiro, dezembro de 1999

Agradecimentos

- Ao Marcelo Gattass, pelo apoio e confiança depositados ao longo de todo o mestrado;

- Ao Marcelo Dreux, pelos conhecimentos e experiências transmitidas desde o primeiro instante, assim como pelo tempo dispensado numa valiosa co-orientação;

- Ao Bruno Feijó e ao Paulo Cezar C. Pinto, por suas preciosas aulas e conselhos dados durante estes anos;

- Ao Afonso Cusati, pelo apoio que me deu na escolha do tema e na colaboração na implementação de alguns algoritmos;

- Ao TeCGraf e ao Visgraf, que auxiliaram e possibilitaram parte da implementação deste trabalho;

- À CAPPES, pela ajuda financeira recebida durante o curso;

- Ao Antônio Elias Fabris, Eduardo Toledo e Hae Young Kim, que me introduziram e estimularam por seguir pelos caminhos da Compuatção Gráfica;

- Aos meus pais, pelo exemplo e apoio que sempre me deram para as atividades acadêmicas.

Resumo

Modelagem de elementos da natureza é uma área de pesquisa atual e com crescente importância para a Computação Gráfica. O objetivo deste trabalho consiste em apresentar uma ampla pesquisa bibliográfica sobre o assunto, definir conceitos fundamentais e propor soluções e novas técnicas para alguns problemas. Embora a pesquisa enfatize principalmente a modelagem procedimental, também serão apresentados métodos de iluminação e visualização adequados para certos tipos de elementos.

Abstract

Modeling natural phenomena is an area of active research and growing importance in Computer Graphics. The purposes of this work are to present a thorough bibliography about this subject, to cover fundamental concepts and to propose some new techniques. It focuses mainly on the procedural modeling techniques but it also present specific visualization and ilumination methods.

Capítulo 1

Introdução

Neste trabalho procura-se realizar uma descrição das principais técnicas conhecidas, assim como sugerir algumas soluções, na difícil tarefa de criar elementos da natureza em cenários virtuais.

Modelar a natureza com realismo visual é um grande desafio para a computação gráfica, dada a extrema complexidade que estes tipos de elementos podem vir a ter. É de crucial importância, no entanto, poder gerar com realismo este tipo de objetos, já que em grande parte de produções que envolvem computação gráfica, os elementos da natureza estão presentes.

Por que estes elementos em geral são complexos de serem modelados?

Estima-se que no universo haja em torno de 10^{60} a 10^{70} partículas [Fournier 94], tendo cada uma delas certa complexidade por si só. Mesmo que se limite a porção do mundo que se deseja modelar apenas ao que está à nossa volta, ter-se-á ainda um número alto de partículas – em torno de 10^{30}. Assim, caso se queira modelar o mundo com todo rigor, seria necessário levar em conta este fator. Contudo, em computação gráfica, na maioria das vezes estamos interessados apenas na aparência externa dos elementos, e não na sua constituição física integral. Desta forma, em muitos casos, é suficiente modelar

apenas a superfície dos objetos. Contudo, na natureza, a maioria das superfícies possui uma complexidade grande. Tome-se como exemplo os pêlos de um animal ou a superfície da água. Apenas utilizando polígonos ou superfícies de ordem paramétrica superior, haverá muitas dificuldades para poder criar este tipo de objetos. Para agravar a situação, em alguns casos modelar a forma geométrica de algum elemento natural não será suficiente. Outros fatores também precisam ser modelados, tais como o movimento destes, que em geral costuma ser bastante complexo - o movimento das ondas numa praia ou o movimento de nuvens num céu, por exemplo. A iluminação na natureza também é um agravante, já que em alguns casos não pode ser modelada simplesmente pelos métodos convencionais da computação gráfica , como o de *Phong* [Phong 75], por exemplo, pois podem haver fatores de ordem física que geralmente não são levados em conta e neste caso tornam-se fundamentais (fatores atmosféricos, por exemplo).

Grande parte dos métodos de modelagem procura descrever modelos geométricos que são relativamente simples, representando na maioria das vezes objetos artificiais. Mesmo modelos sofisticados, como por exemplo um carro com todos os detalhes possíveis, são na verdade um conjunto de elementos com formas bem definidas e comportadas.

Estes métodos, no entanto, nem sempre podem ser adequados para modelar elementos da natureza. Em alguns casos, porque estes elementos podem vir a ser compostos por uma quantidade muito grande de sub-elementos: tome-se como exemplo a modelagem de uma paisagem, com uma floresta incluída. Supondo que nela haja 500 árvores, cada uma com aproximadamente 10.000 folhas e tendo em conta que seriam necessários aproximadamente 20 polígonos para modelar uma folha, então, para cada

árvore seriam necessários por volta de 200.000 polígonos apenas para suas folhas. Já para a floresta inteira, seriam necessários 100.000.000 polígonos, apenas para modelar as folhas das árvores que compõem a floresta.

Em outros casos, os modelos são tão complexos, que nem sequer podem vir a ser modelados de forma realista através de superfícies, como por exemplo uma coluna de fumaça ou o fogo.

Há um fator que torna a modelagem de elementos da natureza num desafio maior ainda: todas as pessoas estão muito habituadas a estes tipos de formas, já que estão presentes constantemente à sua volta, de modo que qualquer discrepância de um modelo virtual com o real será facilmente percebida.

1.1 – Organização da tese

Após realizar uma revisão bibliográfica sobre o assunto, serão apresentados no capítulo 2 algumas abordagens possíveis para se modelar objetos da natureza, mostrando uma classificação já existente na literatura. A seguir divide-se o problema em duas categorias: 2D e 3D, discutindo brevemente as técnicas que serão usadas em cada uma.

O capítulo 3 apresenta a construção das funções procedimentais que posteriormente serão utilizadas no processo de modelagem. Antes de apresentar as funções fractais, apresenta-se uma pequena revisão bibliográfica sobre o assunto.

O quarto capítulo é o mais importante da tese, uma vez que nele serão expostas as técnicas de modelagens procedimentais propriamente ditas. O capítulo está subdividido em seis seções, uma para cada elemento da natureza estudado. Neste capítulo são apresentados algumas novas propostas e resultados práticos obtidos ao longo da pesquisa.

O capítulo 5 irá apresentar possíveis técnicas para a visualização e iluminação dos elementos estudados e finalmente o capítulo 6 apresenta alguns dos principais problemas em aberto neste assunto.

1.2 – Trabalhos Relevantes nesta Área

Desde o surgimento da computação gráfica vários trabalhos foram realizados para geração de elementos da natureza. *Of Growth and Form*, de D'Arcy Thompson [Thompson 61] talvez possa ser considerado o primeiro trabalho no assunto. Neste livro, o autor discute sobre a geometria e as principais leis físicas que determinam a forma de elementos da natureza. *Patterns in Nature* [Stevens 74] já é um trabalho mais recente que discute bastante a forma dos elementos da natureza e similaridades destes com algumas formulações matemáticas. No livro *The Fractal Geometry of Nature* [Mandelbrot 82] Benoit Mandelbrot introduz o conceito de fractais aplicados a fenômenos naturais, abrindo um vasto campo de pesquisa e desenvolvimento para esta área, como será visto no capítulo 2 deste trabalho. Posteriormente Peitgen e Richter estudaram como implementar a teoria dos fractais através de algoritmos computacionais apresentado no livro *The Beauty of Fractals* [Peitgen et al 86]. Em 1994 foi publicado *Texture and Modeling – A Procedural Approach* [Ebert et al 94], que contém uma coletânea de resultados obtidos por diversos autores, principalmente na área de funções procedimentais, dando bastante ênfase a elementos da natureza. Com relação a modelagem de plantas, uma ótima referência é o livro *The Algorithmic Beauty of Plants* [Prusiniewicz et al 90].

Outros trabalhos relevantes realizados na Puc-Rio, que também podem servir como ótima referência para este assunto são *Texturas Aplicadas ao Método da Radiosidade* [Birtel 98], *Técnicas Procedimentais de Texturização* [Aguilar 98], *Técnicas para Geração de Texturas em um Sistema Ray-Tracing* [Faria 96] e *Perturbação e Texturas em Superfícies NURBS* [Cavalcante 94]. Ao longo desta dissertação algumas destas referências serão discutidas com mais detalhe.

Capítulo 2

Metodologias de modelagem

Nesta seção irá apresentar-se primeiramente uma classificação para os métodos que podem ser seguidos quando se deseja criar o modelo de algum elemento da natureza e posteriormente será discutido como estes modelos são criados.

2.1 – Uma classificação para os tipos de modelagens de elementos da natureza

Há inúmeros caminhos que se podem seguir para criar elementos da natureza. Alain Fournier [Fournier 94] descreve uma proposta para classificação de diversas metodologias de modelagem:

1) Modelagem Empírica.

Consiste em digitalizar diretamente o modelo em questão. Um terreno, por exemplo, pode ser facilmente criado desta maneira a partir de medidas de altura para um conjunto de pontos do espaço, que depois serão triangularizados, gerando uma malha poligonal. No caso de uma planta, pode-se criar a modelagem individual de cada uma de suas folhas e galhos e a seguir juntá-los num único modelo.

A principal vantagem para esta classe de modelos consiste no fato de que o tipo de modelo criado é em geral uma superfície similar às criadas pelas técnicas tradicionais de modelagem, o que permite uma fácil integrabilidade com qualquer software de visualização.

As desvantagens consistem principalmente no fato de que, se os modelos em questão forem compostos por uma quantidade muito grande de elementos (é o caso de uma árvore), este método pode tornar-se muito trabalhoso e gerar modelos enormes, além de exigir um trabalho humano extremamente grande. Não se pode também realizar alterações no modelo de forma simples, como na modelagem procedimental. Tornar um terreno mais erodido, por exemplo, seria uma tarefa difícil, ou inviável de ser feita. Também é difícil aumentar a resolução e o detalhamento do modelo, uma vez que este já tenha sido gerado.

2) Modelagem baseada em modelos físico.

Situam-se aqui as técnicas que utilizam equações físicas, químicas, mecânicas, etc., que descrevem algum elemento ou fenômeno natural. Como exemplo, pode-se citar o trabalho desenvolvido por Jim Kajiya e Von Herzen [Kajiya et al 84], que desenvolve um método para modelar nuvens baseado diretamente nas equações diferenciais que descrevem o vapor de água contido no ar, em função da altitude, velocidade do vento e temperatura.

Esta forma de modelar elementos da natureza tem como vantagem o fato de que gera informações reais a respeito do objeto ou fenômeno em questão. Assim, dentro de um sistema de simulações complexo, onde vários elementos e fenômenos devem interagir entre si, estes modelos comportam-se de forma razoável. Além disso, não há muita

dificuldade para animar com realismo os objetos assim criados, já que a variável tempo costuma estar presente nestes tipos de equações. Pode-se ainda, usando esta técnica, realizar modificações sobre os modelos de forma simples e trivial, alterando alguns valores de parâmetros ou constantes envolvidas na equação. Assim, tornar uma nuvem mais esparsa ou mais densa, por exemplo, exige um trabalho muito pequeno quando se tem um modelo realista para descrevê-las.

O principal inconveniente deste método é que nem todos os elementos da natureza são descritos por equações físicas com soluções numéricas possíveis de serem implementadas eficientemente. Além disso, como na maioria das vezes estas equações não foram desenvolvidas para a computação gráfica, levam em conta geralmente uma série de fatores físicos que nem sempre podem ter alguma relevância para sua visualização ou simulação (no caso das nuvens, por exemplo, temperatura, pressão atmosférica, etc.). Por fim, geralmente esta classe de equações costumam exigir um processamento computacional muito grande.

3) Modelagem Estrutural.

Este método engloba as modelagens que procuram descrever apenas a estrutura básica de um objeto. Para entender este método, pode-se pensar na descrição de uma cadeia do DNA, que consiste numa seqüência de elementos básicos, que colocados juntos irão compor algum elemento relativamente complexo. Como exemplos desta técnica pode-se citar a modelagem baseada em gramática - muito utilizada para modelar plantas (ver [Prusiniewicz et al 90]) - e a modelagem feita por descrição de grafos.

Modelar desta forma é vantajoso quando o elemento a ser construído é composto por um número muito grande de estruturas elementares similares entre si, como acontece com as árvores e arbustos, por exemplo.

Em contrapartida, este método não prevê soluções para realizar animações dos elementos gerados. Além disso, há uma classe relativamente pequena de objetos que podem ser descritos desta forma.

4) Modelagem Impressionista

Fournier dá este nome às modelagens que utilizam funções matemáticas que não estão baseadas em leis físicas reais com relação aos fenômenos ou elementos que se deseja criar. Esta técnica recebe este nome pelo fato de que o único objetivo consiste em criar uma impressão da "aparência externa". Um bom exemplo pode ser dado com a função Fractal Brownian Motion –que será estudada melhor na seção 2.3.2 . Esta função, que consiste numa iteração fractal de alguma outra função, pode gerar imagens de nuvens, terrenos, mármore e até ondas para a superfície de um mar, sem no entanto possuir um mínimo de correlação com os modelos físicos reais que descrevem estes elementos.

Esta técnica é bem sucedida principalmente pelo fato de conseguir criar um número muito grande de tipos de elementos diferentes. Assim, há alguns anos têm-se usado extensivamente esta classe de funções para gerar um vasto conjunto de texturas, como será visto mais adiante.

A tabela abaixo apresenta um pequeno resumo dos principais elementos da natureza e como cada um dos métodos apresentados anteriormente se adequam para criá-los.

Tipo de elemento	**Empírico**	**Físico**	**Estrutural**	**Impressionista**
Plantas	Por serem modelos em geral complexos, exigiria muito trabalho.	Técnica inadequada.	Técnica mais apropriada.	Pode ser usado para criar plantas pouco detalhadas e de rápida visualização.
Água (com ondas)	Pode gerar modelos estáticos, porém apresenta muitas dificuldades para animar as ondas.	Possibilita gerar modelos e animações bastante realistas, no entanto exige muito processamento.	Técnica inadequada	Técnica adequada para gerar modelos rápidos de serem visualizados. Difícil de criar animação das ondas.
Nuvens	Técnica inadequada.	Técnica adequada, porém exige formulações físicas muito complexas e o processamento é pesado.	Técnica inadequada.	Técnica adequada. É capaz de gerar modelos similares aos gerados pelo método físico, porém é mais simples e rápido.

Fumaça	Técnica inadequada	Pode-se utilizar esta técnica, porém as equações são relativamente complexas, e os resultados são similares aos obtidos com o método impressionista.	Técnica inadequada	Técnica mais adequada, inclusive para gerar animação.
Terreno	Pode criar modelos bons, especialmente quando se possui os dados do Terreno desejado.	Útil apenas para alguns casos particulares de visualização.	Técnica inadequada	Técnica apropriada quando o modelo não deve seguir uma especificação exata.

Tabela 1 – Principais modelos da natureza e a adequação de cada uma das metodologias de modelagem apresentadas nesta seção.

Não é possível afirmar se um método é melhor ou mais eficaz do que outro, já que isso depende muito do tipo de objeto a ser criado e qual a finalidade que deseja dar a este modelo. Além disso, é muito comum mesclar estas técnicas, de forma a obter resultados

melhores. Pode-se por exemplo criar um modelo geométrico de uma nuvem de forma impressionista e a seguir aplicar-lhe uma textura criada de forma empírica, ou criar o modelo de uma folha pelo método empírico e a seguir modelar toda a árvore de forma estrutural.

Este trabalho enfatiza a modelagem de elementos da natureza utilizando métodos procedimentais, tendo uma aproximação com a descrição do método impressionista descrito na classificação de Fournier, embora em alguns casos outros métodos também serão citados. O processo de criação por meio desta técnica oferece uma flexibilidade muito grande. Com uma mesma função, pode-se gerar os mais diversos tipos de elementos, variando apenas alguns parâmetros, assim como simular fenômenos ou leis da natureza de maneira muito mais simples, e em alguns casos mais elegante, do que ao criar modelos aproximados aos reais. Funções relativamente simples podem modelar objetos extremamente complicados, tais como água, montanhas e nuvens.

2.2 – Modelagem 2D e 3D

Os elementos a serem criados podem ter duas abordagens distintas:

- Modelagem 2D – para este trabalho, irá consistir fundamentalmente em sintetizar texturas procedimentais adequadas;
- Modelagem 3D – Corresponderá em criar modelos geométricos tridimensionais utilizando funções apropriadas. Uma técnica que será bastante explorada neste aspecto são as Hipertexturas, como será descrito mais detalhadamente na seção 2.2.2.

2.2.1 – Modelagem 2D (Texturas Procedimentais)

O processo de aplicação de uma textura consiste no cálculo de algum parâmetro de *shading* para um conjunto de pontos ou área de uma dada superfície geométrica. [Catmull 74] introduz o conceito de aplicação de texturas ao objetos geométricos, fazendo uma parametrização de uma superfície 3D em dois parâmetros (u, v). Desta forma, cada ponto da superfície pode ser mapeado por alguma função para um par de coordenadas de uma imagem digital. Partindo desta idéia surgiram variações e implementações da definição original, dentre os quais se destacam os trabalhos de [Blinn et al 76], onde introduzem o *reflection-mapping*, método que permite simular reflexos em superfícies e [Blinn 78], que introduz o *bump-mapping*, de forma a simular superfícies rugosas, alterando a orientação original das normais. Contudo, a utilização de imagens como texturas para objetos tridimensionais possui alguns inconvenientes:

- Dependendo da superfície a ser mapeada, a imagem pode vir a sofrer deformações inadequadas;
- Haverá problemas de *aliasing* ao aplicar a textura sobre a superfície, uma vez que uma imagem possui domínio discreto e a superfície 3D é contínua no espaço;
- Realizar pequenas alterações na imagem pode tornar-se uma tarefa trabalhosa ou impraticável.

Uma textura procedimental, ou textura sólida, como também costumam ser chamadas, consiste fundamentalmente numa função matemática. Geralmente esta função possui como parâmetro de entrada as coordenadas espaciais do local de um objeto 3D em que se deseja aplicar a textura. Por isto, costuma-se chamar a este tipo de funções como

funções espaciais. Enquanto no processo tradicional de mapeamento de textura, ao realizar uma chamada para um determinado ponto esta irá apenas buscar uma cor armazenada previamente numa imagem, ao realizar a chamada de uma textura procedimental, esta irá calcular uma cor, utilizando uma função com seus parâmetros de entrada.

Existe uma série de vantagens das texturas procedimentais em relação ao mapeamento de uma imagem como textura:

- Uma textura gerada matematicamente não apresentará problemas relacionados a *aliasing*, uma vez que o seu domínio é contínuo. (na seção 5.4 se apresentará inclusive uma técnica para corrigir problemas de perda de resolução em mapeamentos de texturas de imagens utilizando métodos procedimentais);
- Pode-se dizer que o método de gerar texturas de forma procedimental consiste fundamentalmente numa abstração de um elemento através de uma representação matemática do mesmo. Assim sendo, não é preciso armazenar muitos detalhes com relação a ele, o que acarreta numa grande economia de espaço, já que estas funções normalmente consistem em pequenos códigos.
- As texturas procedimentais não precisam ser mapeadas sobre uma superfície, e portanto não ocorrerá deformação do padrão gerado sobre a superfície;
- As funções que descrevem estas texturas possuem em geral uma série de parâmetros, o que possibilita realizar alterações nos padrões gerados com bastante simplicidade. Em alguns casos, como será apresentado posteriormente, estas alterações permitirão criar animações para estas texturas.

É importante também ressaltar algumas desvantagens que as texturas procedimentais possuem, principalmente quando estão sendo utilizadas com o método impressionista de modelagem, ou seja, as funções não correspondem exatamente à descrição físico-matemática dos fenômenos:

- O código da função que descreve a textura procedimental pode ser difícil de ser alterado em alguns casos, uma vez que não tem nada haver com o fenômeno que está sendo modelado;
- Usando uma imagem como textura, o resultado é fácil de ser previsto. Já com as texturas procedimentais os resultados podem não ser tão previsíveis;
- O cálculo de uma textura procedimental é mais lento do que buscar um valor de uma imagem armazenada na memória;
- Apenas um número limitado de texturas pode ser criado matematicamente.

2.2.2 - Modelagem 3D

Modelar geometricamente um elemento da natureza é em geral um processo não trivial, devido à complexidade geométrica da maioria dos elementos. Para objetos com uma superfície bem definida, como um terreno, por exemplo, podem-se combinar as técnicas tradicionais de modelagem geométrica, com funções apropriadas para gerá-las. No caso de um terreno, por exemplo, bastaria criar uma malha poligonal e a seguir, para cada vértice que a compõe, realizar um deslocamento vertical de acordo com o valor retornado por uma função que utiliza estas coordenadas como parâmetro de entrada. Na seção 3 serão discutidos com mais detalhes exemplos específicos para isto.

No entanto, gerar modelos de objetos que possuem uma geometria complexa e mal determinada, tal como gases, nuvens, fumaça, fogo, fluidos em geral, penugens de animais, não é uma tarefa com uma solução trivial.

Gardner propôs inicialmente uma técnica para modelar nuvens tridimensionais e que consiste fundamentalmente em criar elipsóides no espaço e, ao realizar a visualização destes, aplicar-lhes texturas sólidas fractais, geradas a partir de uma série de Fourier [Gardner 85] e [Gardner 94]. Esta textura, contudo, ao invés de ser aplicada como cor da superfície, é aplicada como transparência para o elipsóide, dando a impressão de que o objeto tem uma geometria complexa. Apesar de Gardner ter obtido resultados muito interessantes, este tipo de modelagem não resolve de forma adequada o problema, pois não gera propriamente a geometria do elemento (através deste método é impossível, por exemplo, penetrar no interior do objeto). Além disso, é restrito para poucos tipos de objetos.

Ken Perlin propôs uma outra técnica mais geral, as hipertexturas [Perlin et al 89], para criar modelos 3D fazendo uso das funções procedimentais, que até então eram apenas utilizadas para gerar as texturas sólidas.

Define-se a função D(x) sobre o R^3 como a função de densidade no espaço.

Nos métodos tradicionais de modelagem geométrica têm-se que:

D(x) = 0 ➔ x é um ponto que está fora de qualquer geometria;

D(x) = 1 ➔ x é um ponto que está dentro de alguma geometria.

Assim, um objeto é na verdade um conjunto de pontos x, onde D(x) = 1 sendo que o envoltório destes pontos consiste na superfície da geometria.

As hipertexturas introduzem o conceito de uma região chamada de *fuzzy*, onde a densidade de um objeto pode ser diferente de 1 ($D(x) \in [0,1]$).

O processo de modelagem com hipertexturas consiste em atribuir valores de densidades a um determinado volume do espaço, utilizando para isto funções matemáticas que façam esta distribuição de forma adequada. A estas funções costuma chamar-se de funções de densidade de volume (*Volume Density funcions*).

Esta técnica utiliza a modelagem volumétrica, que consiste basicamente em discretizar o espaço 3D. Cada ponto deve ser representado por um *voxel*, que seria uma unidade mínima do espaço. Um *voxel*, além de poder conter uma série de características do material do objeto que está representando, deverá possuir também um valor de densidade associado.

Assim, o processo de modelagem com hipertexturas resume-se em:

1) Definir um objeto no espaço que irá delimitar o modelo a ser criado;
2) Para todo *voxel v* que estiver no interior deste modelo, realiza-se o cálculo de densidade correspondente:

   $D(v)$ = Função_de_Hipertextura ($v.x$, $v.y$, $v.z$)

   onde $v.x$, $v.y$ e $v.z$ são as coordenadas espaciais x, y e z do *voxel v*.

Desta forma, o que caracterizará um elemento volumétrico será por um lado o objeto que delimitará o volume do modelo (pode ser desde uma primitiva simples, como um cubo ou uma esfera, até um objeto relativamente complexo, como um modelo criado por superfícies implícitas) e a função de hipertextura, que se encarregará de distribuir valores de densidade no interior do objeto delimitador.

Uma vez criada a distribuição de densidades, podem ser feitos “ajustes” ao elemento, de forma a refinar o resultado desejado. Para isto, lança-se mão de uma biblioteca de funções que irão, por exemplo, realizar um aumento no ganho das densidades, alterar o contraste da densidade entre os *voxels*, e assim por diante. Para maiores detalhes veja [Ebert et al 94] e [Perlin et al 89].

Capítulo 3

As Funções Procedimentais

Como já foi mencionado, a modelagem dos elementos da natureza pode ser feita através de funções matemáticas, que não necessariamente tenham alguma relação como a descrição do modelo físico real do objeto em questão. Para isso, construem-se funções básicas, que servirão posteriormente para criar funções mais complexas, utilizando-as em muitos casos, na construção de iterações fractais.

3.1 – Funções básicas

Há uma infinidade de funções matemáticas capazes de criar texturas ou modelos de elementos da natureza. Existe, no entanto, um conjunto de funções, as quais se costuma chamar de funções básicas, que funcionam como primitivas para uma grande classe de funções mais complexas a serem criadas posteriormente. Estas funções devem obedecer a algumas propriedades, sendo uma das principais o fato de apresentarem resultados aleatórios.

Inicialmente, pode-se tomar como uma função básica a função *random*(), que gera valores aleatórios independente de qualquer parâmetro de entrada. Um exemplo gráfico do resultado gerado por esta função é o chuvisco de uma TV, não sintonizada em canal

algum. Contudo, esta função possui uma série de inconvenientes, principalmente devido ao fato de que não possui parâmetros de entrada, não havendo portanto controle algum sobre o seu resultado. Assim, não é possível garantir algo a respeito de uma coerência temporal do resultado, o que é de extrema importância para funções que irão gerar texturas ou modelos. Desta forma, há algumas propriedades que são fundamentais para serem obedecidas ao projetar uma função procedimental, e que portanto são necessárias para qualquer função básica a ser construída:

- Pseudo-randomicidade: Os valores gerados pela função podem ser randômicos, mas sempre, para um mesmo valor dos parâmetros de entrada, a função deve retornar o mesmo resultado;
- Não pode haver periodicidade de padrões;
- Estas funções devem ser estacionárias, invariantes com relação a translação, e isotrópicas, invariantes com relação a rotação.

Existem inúmeras abordagens diferentes na tentativa de implementar funções que obedeçam estas propriedades. A função *noise* é sem dúvida a função que se tornou mais popular e deve-se a Ken Perlin [Perlin 85]. Esta é uma função básica, uma vez que pode ser utilizada na construção de outras funções e obedece às propriedades citadas anteriormente. Detalha-se a seguir como é esta função.

3.1.1 – *Noise*

Como o próprio nome sugere, esta é uma função que produz um ruído estocástico, ou seja, gera padrões aleatórios, obedecendo no entanto às propriedades essenciais às funções básicas.

Define-se um *lattice* como sendo uma discretização do espaço R^3 (pode-se imaginá-lo como sendo uma grade tridimensional). Uma célula é uma unidade deste *lattice* (cubos que irão compor o *lattice*).

Define-se a função de *wavelet* como sendo uma função que fora de um seu valor é zero e a sua integral aplicada de $-\infty$ a $+\infty$ vale zero, ou seja, se esta função é diferente da função nula, então possui regiões positivas e negativas, de forma que uma anula a outra. Define-se o raio de uma *wavelet* como a distância do seu centro até o ponto em que passa a valer zero no domínio em que está definida.

A idéia básica para construir a função *noise*, de acordo com o algoritmo sugerido por Ken Perlin em [Perlin 85], consiste em dividir o espaço num *lattice* regular de células cúbicas com espaçamento de 1 unidade entre cada vértice. Para cada vértice do *lattice* associa-se uma função de *wavelet* com raio equivalente ao tamanho de uma célula. Desta forma, qualquer ponto do espaço interior ao *lattice* sofrerá "influências" de 8 *wavelets* sobrepostos.

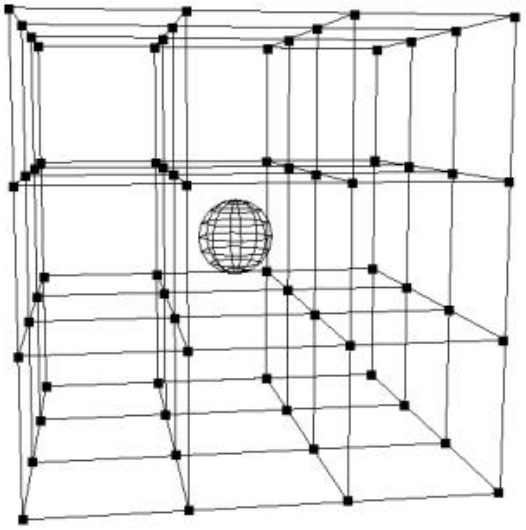


Figura 3.1 – Cada vértice do lattice possui uma função de wavelet associada. Assim, o interior de uma célula sofrerá influências de 8 wavelts dos vértices que a cercam.

Para calcular o valor de *noise* de um determinado ponto deve-se seguir as seguintes etapas:

1) Calcular em que célula do *lattice* este ponto se encontra:

O vértice inferior que encobrirá o ponto (x, y, z) será:

[i, j, k] = [floor(x), floor(y), floor(z)], onde a função floor (t) converte o número real t no número inteiro t', tal que $0 < t - t' < 1$;

Assim, os 8 vértices que definirão a célula em que o ponto se encontra serão:

[i, j, k], [i+1, j, k], [i, j+1, k], [i, j, k+1], [i+1, j+1, k], [i, j+1, k+1], [i+1, j, k+1] e [i+1, j+1, k+1].

2) Encontrar um *Wavelet* para cada vértice da célula:

Em primeiro lugar deve-se descrever como será esta função para cada vértice que compõe o *lattice*, já que cada vértice possui uma função própria. É neste ponto onde

começam a surgir algumas variantes de implementação da função *noise*. Na forma original, descrita por [Perlin 85] estas funções são construídas da seguinte forma:

- Define-se para a função o valor zero no seu vértice correspondente;

- No seu centro, esta função possui um gradiente randômico;

- À medida que se aproxima de outro vértice, esta função tende suavemente para zero.

Assim, para descrever cada um dos *wavelets* é necessário descrever apenas como será o seu gradiente no ponto zero. Para fazer isto, pode-se criar uma tabela, onde se armazenará um vetor relativo ao gradiente de cada vértice.

Este gradiente precisa ser gerado de forma randômica e pode ser feito de diversas formas. Perlin sugere o seguinte método:

- Escolher pontos aleatórios que estejam dentro do cubo $[-1...1]^3$;

- Descartar os pontos que caírem fora de uma esfera de raio 1, centrada em (0,0,0);

- Projetar os pontos que estiverem dentro da esfera para a sua superfície (para fazer isto basta normalizar o vetor obtido).

Outra técnica que pode ser adotada consiste em criar um vetor de norma 1, paralelo a um dos eixos cartesianos (eixo x, por exemplo). Gera-se a seguir um número real randômico entre -1 e 1, rotacionando o vetor sobre y num ângulo equivalente ao arco seno deste número gerado. Em seguida, gera-se outro número randômico entre 0 e 1 e gira-se o vetor com o arco seno deste número sobre o eixo formado pelo produto vetorial de y com o vetor. O resultado destas duas rotações será o gradiente que se deseja

calcular. Embora a distribuição não seja completamente uniforme, na prática este método produz resultados razoáveis.

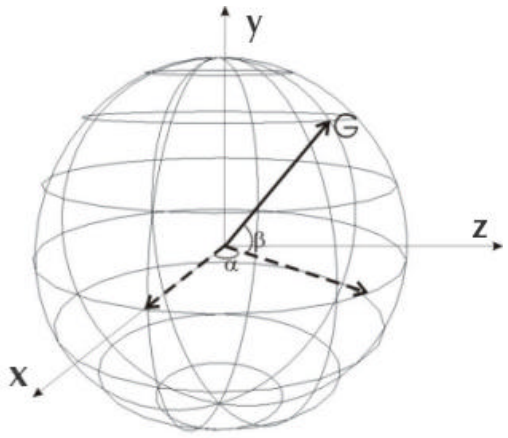


Figura 3.2 - O gradiente será o vetor G, resultante de uma rotação α^o e a seguir de β^o, sendo que estes dois ângulos são valores gerados randomicamente.

O próximo problema que deve ser resolvido consiste em indexar cada vértice do *lattice* a um destes gradientes gerados e armazenados numa tabela. Para que não ocorra uma repetição de padrões deve-se garantir que, para cada vértice, existe apenas um gradiente a ele associado. Deve-se então indexar cada [i, j, k] a um único valor n, com 0 < n < N, onde N é o tamanho da tabela de gradientes. Perlin descreve uma função, a qual chama de *fold*, rápida para fazer esta indexação:

fold (i, j, k)

n = Grad [i mod 256]

n = Grad [(n + j) mod 256]

n = Grad [(n + k) mod 256]

retorne n

sendo que *Grad* é a tabela de gradientes.

Resta agora determinar qual será o valor de *wavelet* de cada vértice que compõe a célula onde o ponto em questão se encontra.

Em primeiro lugar deve-se calcular a posição do ponto relativo a cada um dos vértices de sua célula:

$$[u, v, w] = [x - i, y - j, z - k]$$

onde $-1 \leq u, v, w \leq 1$.

Utiliza-se a função de *wavelet* descrita por Ken Perlin, que a define como sendo o produto de uma função de peso para (u, v, w), que leva a zero quando o raio é 1, por uma função linear que possui um gradiente randômico previamente calculado e armazenado na tabela de gradientes e que vale zero em (i, j, k). A função de peso é definida da seguinte forma:

$$\Omega_{(i,j,k)}(u, v, w) = \text{drop}(u) \,.\, \text{drop}(v) \,.\, \text{drop}(w)$$

sendo que a função drop corresponde à seguinte equação:

$$drop(t) = 1 - 3 \mid t \mid^2 + 2 \mid t \mid^3 \text{ se } \mid t \mid < 1 \text{ e } drop\,(t) = 0 \text{ se } \mid t \mid > 1$$

Deve-se notar que $\int_{-1}^{1} drop(t) = 0$, como se queria inicialmente. Note-se também que $drop\,(1) = drop\,(-1) = 0$, o que garante uma continuidade para a função na extremidade do *wavelet*.

A função linear por sua vez é calculada pelo seguinte produto escalar:

$G_{(i,j,k)} \bullet [u, v, w]$

Note-se que esta função assumirá valor zero quando (x, y, z) coincidir com (i, j, k), satisfazendo a condição estipulada de que o *wavelet* assume valor zero quando está sobre um vértice.

Assim sendo, tem-se que:

$$Wavelet_{(i,j,k)}(x, y, z) = \Omega_{(i,j,k)}(u, v, w) \,.\, (G_{(i,j,k)} \bullet [u, v, w])$$

Finalmente tem-se que *Noise* (*x, y, z*) corresponderá ao somatório dos 8 *wavelets* que estão em torno de (*x, y, z*).

3.1.2 – Função baseada em distância celular

Outra implementação que apresenta bons resultados e possui algumas vantagens sobre a função *noise* é a descrita por Steven Worley em [Worley 96].

Inicialmente o algoritmo cria um conjunto *P* de pontos chaves, espalhados aleatoriamente no espaço R^3, como será exposto mais adiante.

Para cada ponto *x* do espaço, existe um ponto chave *p* pertencente a *P*, que está mais próximo de *x* do que qualquer outro. Assim, define-se a função $F_1(x)$, como sendo a distância deste ponto *p* a *x*. Da mesma forma, define-se a função $F_n(x)$ como sendo a distância do n-ésimo ponto chave mais próximo de *x* pertencente a *P*, até o ponto *x*.

Pode observar-se que ao variar *x*, o valor de F_n é contínuo. Entretanto isto não ocorre com a sua derivada. Assim, ao mudar de um ponto chave para outro, a derivada de F_n irá apresentar uma descontinuidade.

Os lugares no espaço onde há uma equidistância entre dois pontos chaves são planos. Estes são os planos definidos pelo diagrama de Voronoi. Olhando para todo o conjunto de pontos chaves *P*, com relação a cada F_n, pode-se observar que o espaço será particionado por vários planos, formando células.

Esta função F_n obedece a algumas propriedades interessantes:

- F_n é contínua;
- F_n é não decrescente, ou seja, $0 \leq F_1 \leq F_2 \leq ... \leq F_n$;
- O gradiente de F_n tem a direção do vetor com a que vai do n-ésimo ponto chave mais próximo até o ponto x.

Esta função básica precisa inicialmente distribuir os pontos chaves no espaço 3D que, para garantir um resultado adequado, deve seguir uma distribuição espacial estocasticamente conveniente. Worley sugere, para tanto, seguir uma distribuição de Poisson. O espaço pode ser dividido em cubos uniformes, separados pelas coordenadas inteiras. Desta forma, cada cubo pode ser representado univocamente por suas coordenadas inteiras. Assim, para determinar em que cubo um determinado ponto se encontra, basta tomar as componentes inteiras da coordenada. Um ponto definido pelas coordenadas (1.34 , 5.65, 4.32) teria como “endereço” o cubo definido por (1, 5, 4).

O número de pontos chaves para cada cubo é definido pela distribuição estocástica. Assim, não há um número exato de pontos para cada cubo, mas sim um valor

médio para todo espaço. Supondo que este valor médio seja definido por λ então, seguindo a distribuição de Poisson, tem-se que a probabilidade de haver *m* pontos num determinado cubo dá-se pela equação a seguir:

$$P(m) = \frac{1}{l^{-m} e^{l} m!} \tag{3.1}$$

Este resultado será utilizado para determinar quantos pontos chaves devem ser calculados para um determinado cubo.

A seguir, deve-se calcular onde posicionar cada um dos pontos chaves dentro de um determinado cubo. Isto pode ser facilmente calculado, criando 3 números randômicos no intervalo de [0-1] e somando-os com as coordenadas que definem o cubo em questão.

Para realizar o cálculo de F_n correspondente a um determinado ponto *x*, não é necessário percorrer toda a lista de pontos chaves do espaço, o que levaria a um algoritmo extremamente custoso. Como se verá mais adiante, para gerar modelos ou texturas utilizando este método, na prática não é preciso ir além de F_4. Isto permite que a busca pelo ponto chave mais próximo de *x* se limite ao cubo em que está e aos seus 26 cubos vizinhos.

O processo de geração de texturas ou modelos utilizando esta técnica consiste em realizar operações básicas entre os F_n, mapeando o resultado para algum parâmetro (cor, no caso de textura, densidade, no caso de hipertexturas, por exemplo). Assim, se para cada coordenada x = (i, j) de uma matriz calcular-se o F_1 correspondente e o resultado for mapeado para uma cor, irá formar-se uma imagem com manchas circulares em torno dos pontos chaves. Resultados mais interessantes são obtidos quando utiliza-se a F_2 ou a F_3.

Contudo, a grande vantagem que esta função básica possui em relação às demais consiste no fato de que, realizando-se combinações entre estas funções, conforme mostra a equação 3.2, podem-se obter resultados muito variados:

$$F = C_1F_1 + C_2F_2 + C_3F_3 + C_4F_4 \tag{3.2}$$

Esta função, da mesma forma que a *noise*, pode ser utilizada numa iteração fractal, como se verá mais adiante, produzindo resultados excelentes.

3.2 - Fractais

Apenas utilizando as funções básicas de forma direta já seria possível criar uma infinidade de texturas e modelos. Contudo, na natureza, os elementos possuem uma riqueza muito grande de detalhes e que, em muitos casos, apenas puderam ser razoavelmente modelados matematicamente por meio dos fractais. Benoit Mandelbrot foi uma dos primeiros a aprofundar neste tema. [Mandelbrot 82] é uma excelente referência sobre o assunto.

Uma definição simples, porém adequada, para fractais é a dada por Kenton Musgrave em [Ebert et al 94, pp. 251]: “objeto geometricamente complexo, complexidade esta que surge pela repetição de uma fórmula para variações de escala”. Em teoria, para criar um objeto realmente fractal, esta repetição precisaria ser infinita, para escalas cada vez menores. Na prática esta repetição infinita não é necessária, uma vez que a partir de um determinado momento as diferenças geradas pelas recursões serão imperceptíveis na mídia em que se está amostrando o resultado. Esta característica, por

outro lado, é de grande interesse no processo de modelagem de um objeto ou ao criar uma textura: os elementos podem ser “aumentados” infinitamente sem a preocupação de perder resolução ou detalhes. Assim sendo, utilizando uma mesma fórmula pode-se, por exemplo, gerar um céu com várias nuvens pequenas ou um céu com uma única nuvem grande e rica em detalhes. Ao calcular apenas um número finito de iterações diz-se que se está limitando a banda da função fractal.

Esta fórmula a ser repetida será, neste trabalho, uma função básica. Variar a escala corresponde a ir multiplicando sucessivamente, para cada iteração fractal, o domínio da função básica por um valor estipulado, ao qual costuma-se chamar lacunaridade. Ao número destas iterações chama-se oitavas. Dá-se este nome pelo fato de que na música, duplicar a freqüência, que é exatamente o que a multiplicação pela lacunaridade faz, corresponde precisamente a encontrar uma nota numa oitava acima da anterior. Para cada resultado obtido numa iteração, antes de somá-lo com o valor acumulado, deve-se multiplicá-lo por um valor correspondente à amplitude.

Há uma propriedade muito importante dentro da teoria dos fractais e que possuirá relevância nas aplicações deste trabalho: trata-se da dimensão fractal. Enquanto na dimensão inteira Euclidiana as dimensões podem ser apenas números inteiros (0 corresponde a um ponto, 1 a uma reta, 2 a um plano e 3 ao espaço), nos fractais pode-se ter dimensões não inteiras. Assim, é possível mudar de uma dimensão para outra de forma contínua. Graficamente pode-se pensar que este processo corresponde a partir de um plano, por exemplo, ir *preenchendo* aos poucos partes do espaço 3D, até formá-lo por completo.

3.2.1 – Funções Fractais utilizando funções básicas

As funções fractais, em conjunto com as funções básicas, irão formar um grupo de inúmeras funções, capazes de gerar os mais diversos modelos e texturas. Note-se que em momento algum até agora, fez-se menção às propriedades físicas ou químicas de elementos da natureza que irão ser descritos. Reforça-se desta maneira que o tipo de modelagem que se está trabalhando corresponde principalmente à impressionista, conforme mencionado na seção 2. A seguir, descrevem-se as principais funções fractais que fazem uso de funções básicas em suas recursões. Posteriormente, na seção 4 irá descrever-se como estas funções são utilizadas para gerar texturas ou modelos.

3.2.2 – Turbulência

Esta função, descrita inicialmente por Ken Perlin [Perlin 85], nada mais é do que uma iteração fractal utilizando a função *noise*. É interessante ressaltar que os resultados obtidos são muito similares aos que Gardner em [Gardner 84] conseguiu por meio de suas transformadas inversas de Fourier.

Inicialmente poderia escrever-se esta iteração da seguinte forma:

```
Resultado = 0;
frequência = Freq_Inicial;
Enquanto (frequência  <  Freq_Máxima)
        Resultado += noise (P * frequência) * Amplitude (frequência);
        frequência *= 2;
```

onde *P* é o ponto para o qual se deseja encontrar o valor fractal e Amplitude (frequência) é uma função que gera valores que limitam o resultado de cada iteração, em função da freqüência.

Uma forma simples de construir a função *Amplitude*( ) é a de defini-la como sendo inversamente proporcional ao valor da freqüência.

Ao construir esta função, Ken Perlin acrescentou um fator de forma a gerar certa descontinuidade na função: supondo que a função *noise* gere valores no intervalo de –1 a 1, a função turbulência utilizará apenas o valor absoluto do resultado retornado pela função *noise* aplicada ao ponto *P*.

```
Resultado = 0;
frequência = Freq_Inicial;
Enquanto (frequência < Freq_Máxima)
        Resultado += abs( noise (P*frequência)) / frequência;
        frequência *= 2;
```

3.2.3 – fBm (Fractal Brownian Motion) [Mandelbrot 82]

A função fBm consiste numa generalização da definição dos fractais.

Primeiramente define-se como será a função para a amplitude. De uma forma diferente e mais genérica do que a apresentada na seção 3.2.2, procura-se agora construir uma tabela com valores de amplitude específicos para cada iteração a ser realizada. Para que esta tabela seja correta, estes valores devem ser decrescentes e dentro do intervalo [0,1].

Mostra-se a seguir uma possível forma de inicializar esta tabela:

f*requência = 1.0*

i = 1

Enquanto (i < oitavas)

Tabela de amplitudes[i] = Amplitude (frequência,H)

*frequência = frequência * lacunaridade*

sendo que *oitavas* corresponde ao número total de iterações fractais que serão realizadas e *H* corresponde a um ou mais parâmetros adequados para a função Amplitude. Esta função pode ser definida em outro local. Alguns exemplos de funções adequadas para isto podem ser:

- *Amplitude (f, H)* = $\frac{1}{f}$ (a)
- *Amplitude (f, H)* = $\left(\frac{1}{f}\right)^{H}$ (b)
- *Amplitude (f, H)* = $\frac{1}{f}\cos(f + \frac{p}{H})$ (c)

(3.3)

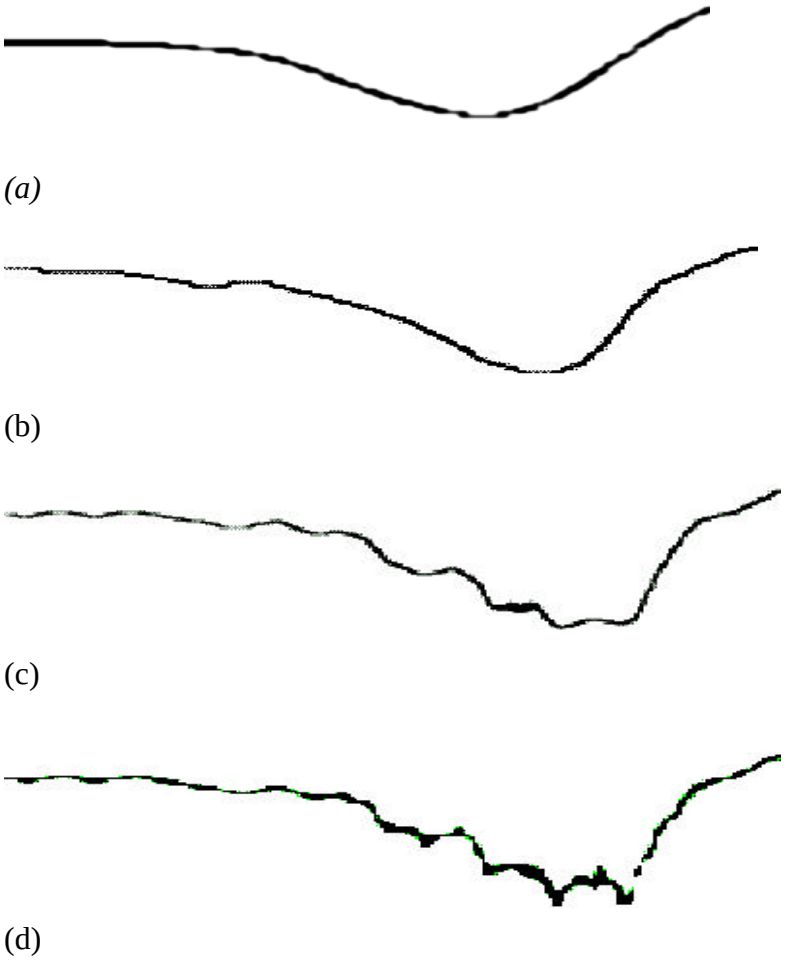
Uma vez inicializada esta tabela, resta apenas realizar as iterações fractais utilizando alguma função básica adequada. Como foi discutido anteriormente, para que uma função seja verdadeiramente fractal, o número de iterações deve ser infinito. Porém viu-se que na computação gráfica é possível limitar-se a banda desta função, realizando apenas algumas iterações. Mais ainda, alguns modelos, como será visto posteriormente,

podem ser criados com um número pequeno de iterações. Na verdade, pode-se dizer que estes modelos são pseudo-fractais, uma vez que são construídos apenas por um número finito e pequeno de iterações de uma função fractal original.

```
Para i = 1 até  Oitavas  faça
    resultado = resultado + função_Básica(P) * Tabela de Amplitudes[i];
    P = P * lacunaridade;
```

Pode-se observar como neste pseudo-código a Tabela de Amplitudes delimita o valor a ser acumulado em cada iteração. Ao multiplicar *P* pela lacunaridade se está realizando uma nova amostragem do ponto *P* com a função básica para um espaço com resolução maior que a anterior.

Na seqüência de figuras a seguir pode-se observar diferentes valores de fBm obtidos com diversos valores de iterações fractais, utilizando como função básica a função *noise*.



(a)

(b)

(c)

(d)

Figura 3.3 – fBm com 1 (a), 3 (b), 5 (c) e 8 (d) iterações fractais respectivamente.

Para gerar as imagens acima utilizou-se como função para amplitude o inverso da freqüência. As imagens abaixo ilustram duas curvas geradas com a mesma implementação da fBm, porém utilizando diferentes valores para o parâmetro H, e portanto diferentes expoentes para a função apresentada na equação 3.3 (b).

(a)

(b)

Figura 3.4 – fBm com H = 0.2 (a) e H = 0.6 (b) respectivamente.

Para alguns tipos de modelos a função fBm possui um inconveniente, que é o fato de ela ser estatisticamente *homogênea,* possui a mesma distribuição em todo o espaço, e *isotrópica,* possui a mesma distribuição em todas as direções. Para tanto [Musgrave et al 89] criou uma variação da definição inicial, a qual chamou de funções multifractais.

3.3 – Funções Multifractais

As funções fractais vistas até agora possuem como característica o fato de serem estatisticamente homogênea em todo o espaço. [Evertsz et al 1992] definem as iterações fractais discutidas nas seções 3.2.2 e 3.2.3 como sendo cascatas aditivas, ou seja, em cada iteração, o novo resultado encontrado é simplesmente somado ao valor acumulado das iterações anteriores. As funções multifractais são construídas em cascata multiplicativa, ou seja, ao invés de somar o resultado em cada iteração, realiza-se uma multiplicação.

Realizar isto faz com que a distribuição estocástica deixe de ser homogênea, uma vez que o resultado final de uma iteração dependerá de todas as chamadas a uma função básica. Basta que numa iteração, uma das chamadas à função retorne um valor pequeno, próximo a zero, para que todo o resultado desta iteração também seja pequeno. Note-se que os picos desta função serão apenas nas situações em que todas as chamadas à função básica retornarem valores altos. Por isto, costuma-se chamar também estas funções de heterogêneas.

```
Para i = 1 até  Oitavas  faça
    resultado= resultado* função_Básica(P)*Tabela de Amplitudes[i]*Contraste;
    P = P * lacunaridade;
```

Acrescenta-se um novo parâmetro, chamado aqui de *Contraste*, que pode ser entendido como uma taxa de multifractalidade. Quando este valor for zero, a função retornará zero sempre, quando for um valor alto, haverá um contraste muito alto entre os valores retornados.

Como será visto na seção 4.1, este tipo de função é muito adequado para gerar modelos de terrenos. Pode-se realizar uma série de alterações nestas funções, de forma a conseguir resultados bastante variados, como será mostrado posteriormente.

Capítulo 4

Modelagem para Visualização de elementos da natureza

Esta seção discute como modelar alguns elementos específicos, aplicando os métodos expostos nas seções anteriores e fazendo uso das funções apresentadas. Apresentam-se também algumas contribuições deste trabalho de pesquisa.

4.1 – Terrenos

Dos elementos da natureza a serem abordados neste trabalho, este é o mais simples de ser modelado, uma vez que um terreno possui uma superfície geométrica bem definida.

Existem inúmeras formas de se criar modelos de terrenos e relevos. Cabe neste trabalho realizar uma abordagem dos métodos procedimentais, ou mais especificamente, utilizando fractais e lançando mão das funções básicas vistas anteriormente. Vale a pena ressaltar que nem sempre este método é a melhor solução, como se verá mais adiante.

Os modelos geométricos que serão criados aqui serão gerados a partir de uma malha de polígonos. As coordenadas espaciais de cada vértice serão os parâmetros de

entrada para as funções procedimentais e o resultado obtido corresponderá ao valor de deslocamento da altura do vértice em questão.

A vantagem de se criar terrenos de forma procedimental consiste principalmente no fato de que se pode ter um controle paramétrico relativamente sofisticado com relação ao modelo. Além disso, não há problemas relativos à resolução do mapa que será aplicado à malha. No entanto, a principal desvantagem de se modelar um terreno de forma procedimental é o fato de que se impossibilita, em parte, a criação de um terreno específico (os métodos procedimentais criam padrões aleatórios), sendo necessário para isto uma imagem digital que sirva como mapa de altura para a malha. Apresenta-se mais adiante uma técnica que procura mesclar ambas as técnicas, de forma a adquirir as vantagens de cada uma.

A idéia de modelar terrenos de forma procedimental foi introduzida por Mandelbrot [Mandelbrot 82], que observou uma ligeira semelhança de uma curva gerada pela fBm no plano, com o contorno de montanhas. Realizando uma extensão para o espaço tridimensional, procurou criar uma superfície semelhante a uma cordilheira. O resultado obtido assemelhou-se ao modelo da figura 4.1, que, como se pode observar, ainda não corresponde exatamente à geometria de uma montanha. Isto porque as funções fractais que utilizam iterações de alguma função básica resultam geralmente em funções estatisticamente homogêneas e isotrópicas, quando construídas em cascata aditiva. Desta forma, este tipo de função não irá representar de maneira adequada um relevo, pois na natureza os padrões de terrenos raramente são uniformemente espalhados. Uma cordilheira costuma soerguer-se de uma planície, há partes de montanhas mais desgastadas que outras, pelo efeito da erosão, existem vales formados por rios, etc.

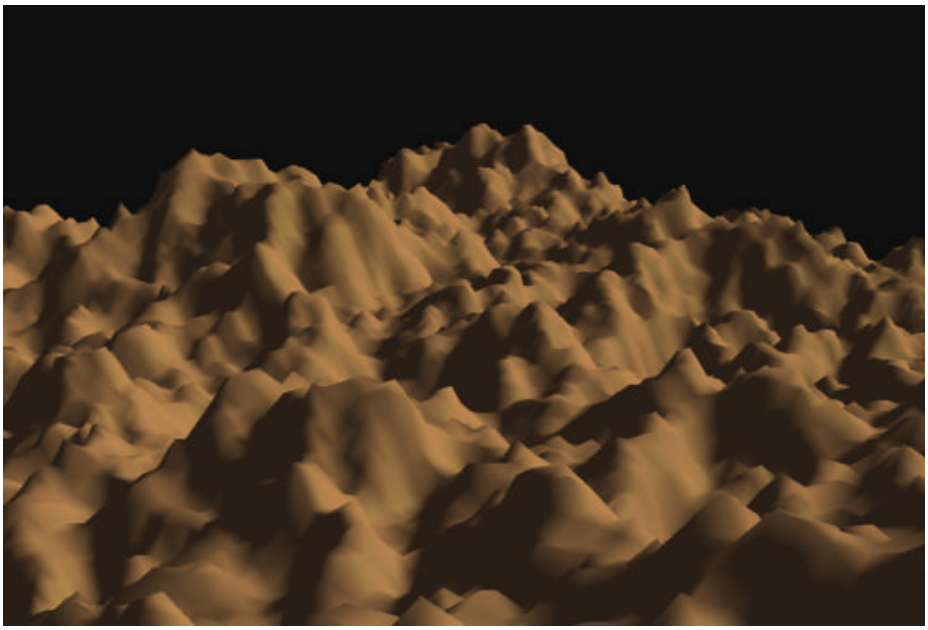


Figura 4.1 - Terreno gerado utilizando diretamente a função fBm.

Por esta razão, procura-se construir as funções fractais em cascata multiplicativa, como foi mencionado na seção anterior, obtendo assim as funções multifractais. A função multifractal mais básica, como foi mostrada na seção anterior, consiste apenas em trocar a somatória dentro da iteração fractal por uma multiplicação.

O efeito desta nova função consiste em criar um espectro heterogêneo. Quando aplicado a uma malha, produzirá regiões mais planas e outras mais acidentadas, assemelhando-se mais com os acidentes geográficos. Partindo desta idéia, podem-se criar diversas variações das funções multifractais, de forma a simular algum fenômeno na formação geológica dos terrenos, de acordo com o método de modelagem impressionista, apresentado na seção 2.

Inicialmente, observa-se que em algumas formações montanhosas, as planícies, de onde emergem as montanhas, são regiões mais desgastadas pela erosão, e portanto mais arredondadas, enquanto as partes mais altas do relevo, possuem um ruído maior, pois são geologicamente mais recentes. De acordo com Musgrave [Musgrave 99], para criar este tipo de modelo, pode-se acrescentar à iteração multifractal um coeficiente, que irá tornar os valores mais baixos (correspondendo às regiões mais baixas) em valores mais homogêneos, enquanto para os valores mais altos (correspondendo às regiões mais altas) procura-se tornar o ruído mais evidente. Uma das formas de se fazer isto consiste em fazer uma pré-avaliação do ponto em questão: se o ponto corresponder a uma região mais alta, utiliza-se um coeficiente próximo a um, o que manterá o ruído original da função. À medida que os valores de altura forem sendo menores, utiliza-se coeficientes cada vez menores, anulando de certa forma o ruído original. O pseudo-código abaixo mostra como isto pode ser feito:

```
Altura = função_Básica (P)   // Pré avaliação da altura do ponto P no terreno
Para i = 1 até  Oitavas  faça
    incremento = função_Básica(P) * Tabela de Amplitudes[i] * Altura;
    Altura = Altura + incremento;   // O coeficiente incremento tende a não mudar
                                       muito quando a altura for pequena.
    P = P * lacunaridade;
```

A figura 4.2 é um exemplo de um terreno criado com esta variação da função multifractal.

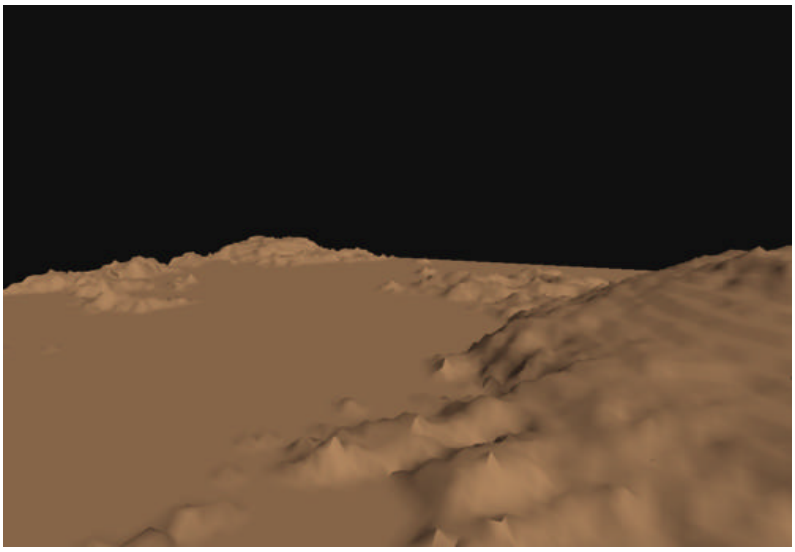


Figura 4.2 - Terreno gerado utilizando função multifractal com coeficiente de amortecimento de ruído inversamente proporcional à altura.

Uma variação para este tipo de formação consiste em não permitir que os planaltos tenham ruído, independente da altura em que se encontrem. Para isto, deve-se fazer com que o coeficiente cresça ou decresça em função da variação da altura, e não mais da altura absoluta. A figura 4.3 mostra uma implementação desta função.

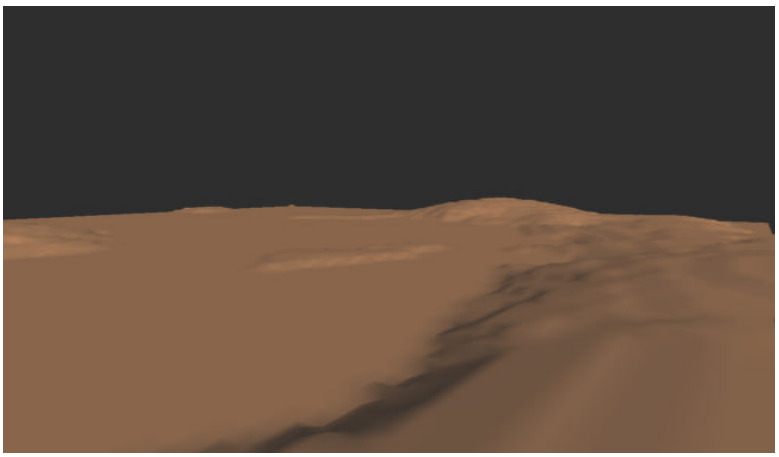


Figura 4.3 - Terreno gerado utilizando função multifractal com coeficiente de amortecimento de ruído inversamente proporcional à variação da altura.

Outro tipo de formação que pode-se criar consiste em terreno com relevos desgastados pela erosão (como por exemplo, colinas). Neste caso, a implementação é mais simples, pois basta reduzir o número de iterações fractais, tornando o ruído pequeno. O relevo da figura 4.4 foi gerado utilizando a função multifractal padrão, com apenas 2 iterações fractais.

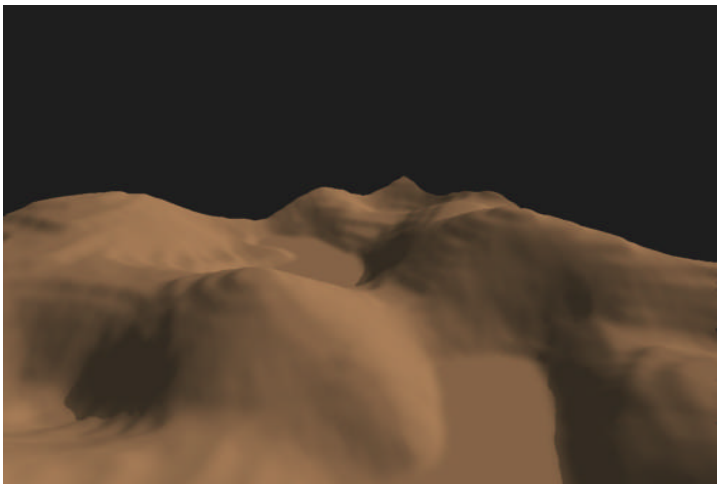


Figura 4.4 - Terreno gerado utilizando função multifractal com um número pequeno de iterações fractais.

Podem-se construir inúmeras variações das funções multifractais, seguindo a idéia de modelagem impressionista. O seguinte terreno foi obtido utilizando a função usada para gerar o relevo da figura 4.5, mas utilizando a idéia da função turbulência, descrita por Ken Perlin. Aqui a função básica retorna um valor que pode ser negativo. Quando isto acontece converte-se o valor para o seu absoluto correspondente.

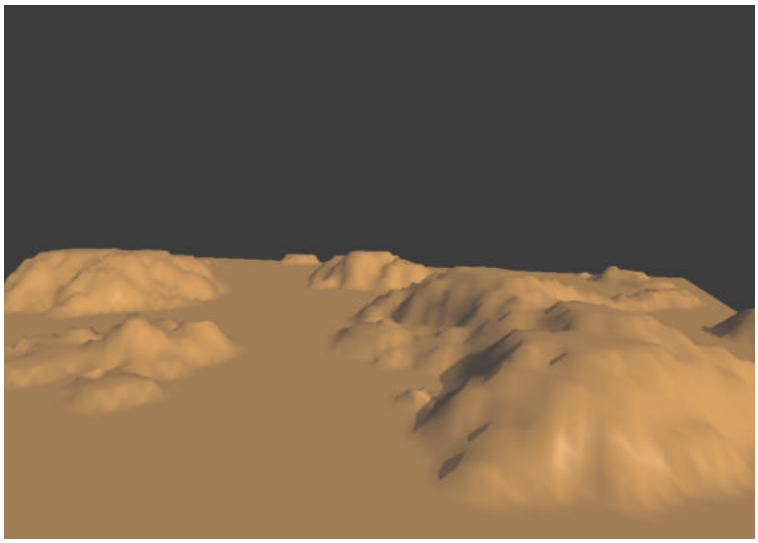


Figura 4.5 - Terreno gerado utilizando função multifractal usando a idéia da função turbulência, convertendo os valores negativos nos seus absolutos correspondentes.

Muitos terrenos são fortemente determinados pela formação rochosa que sofreram. [Musgrave 99] apresenta uma técnica que permite simular terrenos que são de origem vulcânica, possibilitando acrescentar bastante realismo a algumas cenas da natureza. A idéia básica consiste em distorcer o domínio da função geradora do terreno utilizando, para isto outra função, que pode ou não ser também de origem fractal. Assim, até agora o processo para gerar um terreno resumia-se em:

Para todos os pontos P da malha faça

Função_Relevo (P)

Realizando a distorção do domínio, o processo pode ser resumido como:

Para todos os pontos P da malha faça

Função_Relevo (Distorce_Dimínio (P))

O que eqüivale a:

$$P' = G\,(x + f(x, y, z), y + f(x, y, z), z + f(x, y, z))$$

onde *P'* é o ponto da malha perturbado pela função geradora de terreno *G* e *f* é a função que irá distorcer o domínio. (Esta função *f* pode inclusive ser a própria *fBm* apresentada anteriormente)

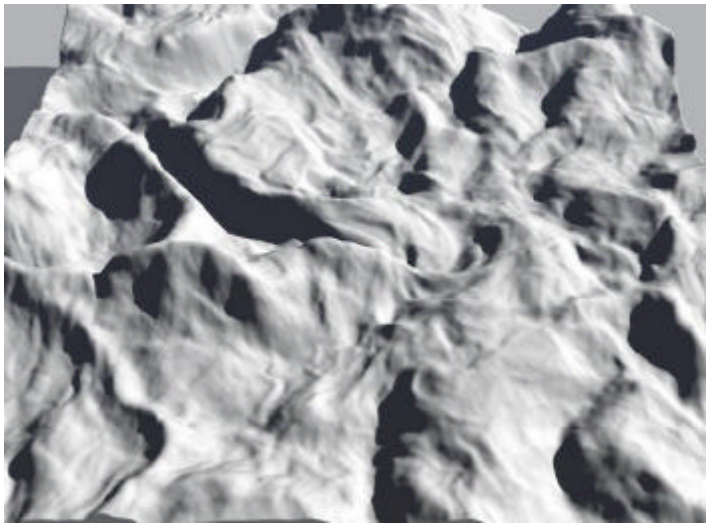


Figura 4.6 – Terreno gerado utilizando a distorção do domínio pela função *fBm* [Musgrave 99].

4.1.1 – Modelagem de Terrenos utilizando mapas de relevo com interpolação Fractal

Como foi apresentado anteriormente, o grande inconveniente da modelagem procedimental de terrenos é a impossibilidade de se criar modelos baseados em mapas reais. Para poder modelar este tipo de relevos, é necessário utilizar uma imagem ou algum outro tipo de fonte de dados na forma de mapa de altura. Por outro lado, criar modelos desta forma também possui seus inconvenientes, como a impossibilidade de um controle paramétrico e problemas relativos à amostragem.

Propõe-se neste trabalho um método para mesclar a modelagem procedimental com a modelagem baseada em mapas de altura, procurando extrair benefícios de ambas as técnicas.

Inicialmente, quando se deseja construir um terreno utilizando mapas, extrai-se o valor da altura para um ponto de uma malha a partir de uma imagem. Supõe-se neste trabalho que antes de utilizar este resultado para mapear a altura diretamente sobre a malha, é necessário realizar uma interpolação para este valor, uma vez que a imagem utilizada é discreta, podendo ocorrer alguns saltos bruscos de altura. Assim, torna-se necessário realizar inicialmente uma determinada interpolação para estes valores. A técnica descrita procurará acrescentar à interpolação linear um ruído gerado por alguma função fractal.

Ao calcular o mapeamento para um ponto P , obtêm-se um valor $U = (u_1, u_2, ..., u_i)$, onde i é a dimensão da função f que será mapeada sobre a superfície (caso a função seja uma imagem 2D então i = 2). Neste caso U corresponde ao parâmetro de entrada para a função f. Ao realizar-se o cálculo $f(U)$ obtêm-se como resultado o parâmetro que

se deseja mapear (por exemplo, a altura especificada pelo valor de uma cor, caso *f* seja uma imagem). Como *f* é uma função discreta no domínio, *U* deverá ser aproximado para um valor válido, por exemplo para uma coordenada composta apenas por números inteiros. Para realizar a interpolação linear, ao invés de simplesmente aproximar *U* para um valor válido, calcula-se a distância deste ponto até cada um dos pontos válidos que o cercam:

Seja

$U_1 = (\lfloor u_1 \rfloor, \lfloor u_2 \rfloor, ..., \lfloor u_i \rfloor)$

$U_2 = (\lceil u_1 \rceil, \lfloor u_2 \rfloor, ..., \lfloor u_i \rfloor)$

...

$U_k = (\lceil u_1 \rceil, \lceil u_2 \rceil, ..., \lceil u_i \rceil)$

e $m_1 = | U - U_1|$

$m_2 = | U - U_2|$

...

$m_k = | U - U_k|$

onde $k = 2^i$

Para encontrar um mapeamento para um dado ponto *U* de uma superfície utilizando a interpolação linear realiza-se o seguinte cálculo:

$$C = t(U_1) \cdot \frac{m_1}{m_1 + m_2 + ... + m_k} + t(U_2) \cdot \frac{m_2}{m_1 + m_2 + ... + m_k} + ... + t(U_k) \cdot \frac{m_k}{m_1 + m_2 + ... + m_k}$$

(4.1)

onde *C* é o parâmetro que se deseja encontrar através da função de mapeamento e *t* é a função de mapeamento de textura.

Para acrescentar um fator fractal, sugere-se utilizar a seguinte modificação da equação 4.1:

$$C = t(U_1).\frac{m_1}{m_1 + m_2 + ... + m_k} + ... + t(U_i).\frac{m_i}{m_1 + m_2 + ... + m_k} + i^i ruído(P) \cdot \frac{(m_1 \cdot m_2 \cdot ... \cdot m_i)}{(m_1 + m_2 + ... + m_k)^i} \cdot TaxaRuído$$

(4.2)

onde *ruído(P)* corresponde à chamada para uma função básica ou fractal apropriada e *TaxaRuído* é um fator que escala o ruído para uma taxa desejada.

Mostra-se a seguir a aplicação desta teoria para o caso de estar realizando uma interpolação para mapas de altura, baseada em curvas de níveis. Observando a figura 4.7, suponha-se que um ponto *P* da imagem esteja sendo mapeado para uma malha 3D. Neste caso m_1 será a menor distância do ponto P até uma das curvas de nível e m_2 será a menor distância até a outra curva de nível. Assim, a altura que será atribuída para a região da malha correspondente ao ponto *P* será dada pela equação abaixo:

$$Altura = \frac{m_1}{m_1 + m_2} \cdot Cv_2 + \frac{m_2}{m_1 + m_2} Cv_1 + 4 \cdot ruído(P) \cdot \frac{m_1 \cdot m_2}{(m_1 + m_2)^2} \cdot TaxaRuído \tag{4.3}$$

onde Cv_1 e Cv_2 correspondem aos pontos mais próximos das curvas de nível cercando P.

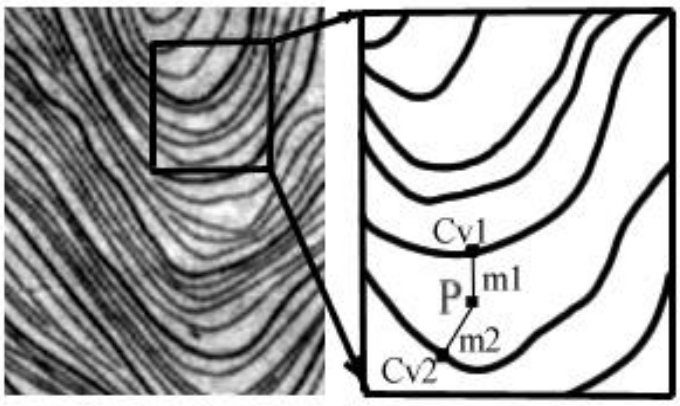


Figura 4.7 - mapa de altura baseado em curvas de nível. Cada curva do mapa corresponde a uma altura que será mapeado para uma malha.

4.2 – Solos

Texturas sobre os solos dos terrenos gerados são de fundamental importância para gerar imagens realistas. Poderia inicialmente projetar-se uma textura previamente gerada sobre o modelo geométrico. No entanto, na maioria das vezes, além dos terrenos possuírem uma extensão relativamente grande, a superfície costuma ser irregular (morros, vales, etc), o que irá provocar uma grande distorção da imagem mapeada. Assim, melhores resultados são obtidos, caso se consiga descrever este solo de forma procedimental.

Uma forma muito simples de criar este tipo de textura consiste em chamar a função *noise* para cada ponto da superfície. Este procedimento criará sobre o solo uma aparência de manchas, com cores que podem ser controladas e editadas. Este método por um lado é simples, mas seus resultados não são muito satisfatórios.

Uma forma um pouco mais sofisticada para criar uma textura de solo consiste em utilizar a função fBm [Ebert et al 94]. Neste caso, utiliza-se o resultado obtido para indexar uma tabela de cores previamente calculada.

solo = fBm (ponto) // Supõe-se que fBm retorna um valor entre 0 e 1.

*índice = (int) (solo * (Tamanho da tabela de cores))*

cor do solo = Tabela_de_cores[índice]

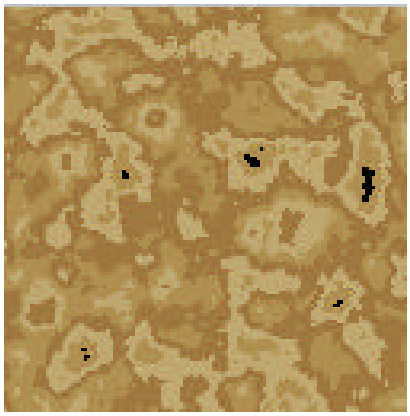


Figura 4.8 - Textura procedimental de solo utilizando função fBm e uma tabela de diversos tons de marrom.

Nota-se que na figura 4.8, há um salto brusco de uma cor para outra. Para suavizar este efeito , deve-se realizar uma interpolação qualquer entre os valores da tabela, o que não é difícil.

Para um terreno completo pode-se utilizar um conjunto de tabelas de cores pré-definidas (uma para vegetação, outra para terra, outra para neve, etc), e dependendo da localidade do ponto a ser visualizado, realiza-se a indexação para uma tabela adequada. Assim, se um ponto está localizado a uma determinada altura, onde deverá haver neve, então realiza-se a consulta na tabela correspondente à neve; se o ponto estiver numa região de mata, utiliza-se uma tabela correspondente à vegetação, etc. A seção 5.4 descreve detalhadamente uma aplicação deste tipo de texturas.

4.3 – Água

O principal desafio na modelagem de água consiste em simular as ondas da superfície (caso se deseje criar um lago cristalino e completamente calmo, espelhado, não haverá maiores problemas).

Existe uma série de trabalhos que procuram realizar uma descrição física sobre as ondas, tais como [Miyata 86], onde o autor leva em consideração a velocidade da água, a pressão atmosférica, a profundidade do local e uma série de informações do meio físico, resultando por um lado numa simulação realista, mas por outro lado num sistema de equações diferenciais relativamente complexo e portanto caro em termos computacionais.

N. Foster e D. Metaxas, em [Foster et al 99], também apresentam métodos computacionais baseados em modelos físicos para simular movimentos de líquidos com realismo.

Paralelamente a estes, desenvolveram-se métodos procedimentais, a fim de simular ondas e seus movimentos com mais simplicidade e eficácia computacional.

Em [Ebert et al 94] Musgrave descreve uma técnica de simulação de ondas utilizando a função fBm e que é bastante simples, oferecendo resultados bastante satisfatórios.

Esta função é aplicada sobre uma malha e isto pode ser feito de duas maneiras básicas:

a) Utilizar o valor retornado da função como um valor para perturbação das normais, sem alterar no entanto a modelagem. Esta técnica é conhecida como *bump-mapping* [Blinn 78] e apresenta bons resultados se a deformação que se procura é relativamente pequena em relação ao objeto inteiro. Esta técnica possui uma grande vantagem, pois o cálculo pode ser feito em tempo de rendering, não sendo necessário pré-processamento algum.

b) Utilizar o valor retornado para criar uma deformação real da geometria do objeto. É necessário recorrer a este método quando as ondas são relativamente grandes. Para isto deve-se realizar um pré-processamento da malha em questão, antes de realizar a visualização.

O método que será descrito consiste basicamente em gerar uma textura procedimental, que é em seguida aplicada sobre a malha em questão como *bump-mapping*.

Realizando uma iteração fractal relativamente pequena obtêm-se padrões como os da figura a seguir:

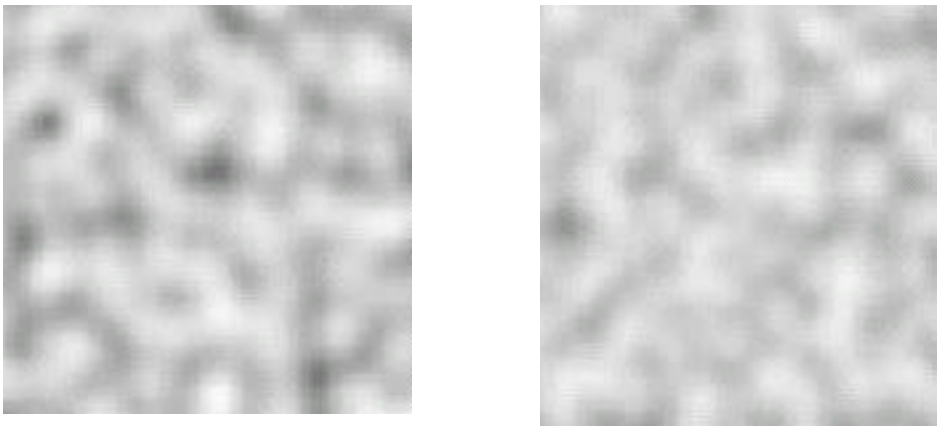


Figura 4.9- Textura gerada com 2 e 3 iterações fractais da função *noise*, respectivamente.

Musgrave sugere simplesmente aplicar estas texturas sobre uma malha. Neste trabalho, no entanto, criou-se um novo e bem sucedido método procurando criar uma camada de várias texturas fractais, cada uma com freqüências diferentes. Na verdade, o número de camadas irá depender do comportamento da água que se deseja modelar. Se, por exemplo, deseja-se criar um oceano, deverá utilizar-se muitas camadas, sendo as primeiras correspondentes às ondas com amplitudes grandes e as camadas finais correspondentes às ondas que possuem amplitudes mais baixas e freqüências maiores. No caso da modelagem de um pequeno lago, apenas serão utilizadas camadas com freqüências altas.

Para isto, primeiramente é necessário pré-calcular uma tabela que limitará o valor de influência para cada camada, de forma a evitar uma saturação na textura que será criada. Assim, por exemplo, assumindo que na textura final, o maior valor a ser mapeado para cada pixel seja 1, e sabendo que serão utilizadas 3 camadas, então a primeira poderia estar limitada para 0.5, a segunda para 0.35 e a terceira para 0.15. Fazendo isto, garante-

se que ao somar as três camadas, o maior valor possível será 1. Para pré-calcular esta tabela de limitações, pode-se utilizar o seguinte algoritmo:

resto = 1

Para i = 1 até (Número_de_Camadas – 1) faça

Limite_da_Camada[i] = resto . 0,6

resto = resto . 0,4

Limite_da_Camada [n] = resto

Finalmente, para criar a composição das camadas, deve-se alocar um *buffer* com o tamanho da textura a ser criada e realizar a seguinte iteração:

Para k = 1 to Número_de_Camadas faça

Para cada P do buffer faça

Buffer(P)=fBm(P,n) . Limite_da_Camada [k]+ Buffer(P)

onde *P* é a coordenada (x,y) dos pontos do buffer, *fBm* é a função mencionada anteriormente, retornando um valor entre 0 e 1, e *n* é o fator para a função *noise* (valores altos para *n* geram uma frequência baixa, enquanto valores baixos geram uma frequência alta).

Para criar uma animação, torna-se necessário realizar uma simulação do movimento das ondas. Na prática, deve-se realizar uma variação na amplitude ao longo

do tempo, assim como o seu deslocamento em algum sentido, de acordo com a direção do vento, por exemplo.

Para realizar o deslocamento da onda pode-se mover o ponto *P* da superfície, somando-o com um vetor de direção, antes de realizar a chamada da função fBm para este ponto. Para realizar uma variação da amplitude da onda, pode-se fazer uma chamada a alguma função qualquer (*noise, turbulência* ou até mesmo a própria fBm) e multiplicar o valor de amplitude original pelo resultado. Como o ponto *P* está sofrendo um deslocamento sobre o espaço, o valor de amplitude retornado para um mesmo ponto, em tempos diferentes será também diferente. O seguinte código resume esta idéia:

P = P + t . Direção_do_Movimento

*Altura_da_Onda = amplitude * fBm (P)*

freq_vento = 0.1

freq_Amplitude = 1

vento_minimo = 0.3

*turbulência = fBm (P * freq_vento)*

*vento = vento_minimo + freq_Amplitude * turbulência*

*onda = altura do ponto + vento * Altura_da_Onda*

Para uma simulação mais realista, este cálculo deverá ser feito para cada camada de textura que se está usando. Note-se que neste caso, cada camada deverá ter valores próprios para a amplitude, direção do vento, frequência do vento, etc.

(a)

(b)

(c)

Figura 4.10 - (a) malha utilizando textura de baixa frequência, com uma camada gerada com fBm e aplicada com deformação da geometria. (b) malha utilizando textura com alta freqüência, com fBm e aplicada como *bump-mapping* na geometria. (c) malha utilizando textura com 4 camadas, com freqüências variando da utilizada para (a) até a utilizada para (b) e aplicada como *bump-mapping* sobre a superfície.

O método descrito gera imagens muito realistas de ondas. Contudo, para gerar animação e alguns fenômenos mais sofisticados, tais como quebra de ondas, esta técnica não é tão eficiente. [Tessendorf 99] apresenta uma abordagem procedimental para

simulação de oceanos, baseada, em parte, em modelos oceanograficos reais. Para isto, o autor lança mão das ondas de Gerstner [Fournier 86], que consistem em equações utilizadas para aproximações em simulação da dinâmica dos fluidos. Este modelo basicamente descreve o movimento circular que partículas individuais irão possuir numa superfície com ondas. Define-se para esta partícula, no plano XZ, que $P_X = (x_0 , y_0)$ e, com relação à altura, que $P_y = 0$. Assim, o deslocamento da partícula *P* quando uma onda com amplitude *A* passa por ela será dada pela equação abaixo:

$$X = x_0 - (\frac{K_x}{k}) A.\text{sen}(K_x.x_0 - wt) \quad e \quad Y = A.\cos(K_x.x_0 - wt) \tag{4.4}$$

onde *K* é chamado de um vetor de onda, que é horizontal com relação à direção com que as ondas se movem e com magnitude *k*, relacionado com o comprimento λ da onda da seguinte forma:

$$k = 2.p / l \tag{4.5}$$

A figura 4.11 mostra como é o movimento desta partícula com ondas de diversas amplitudes e seguindo as equações descritas previamente.

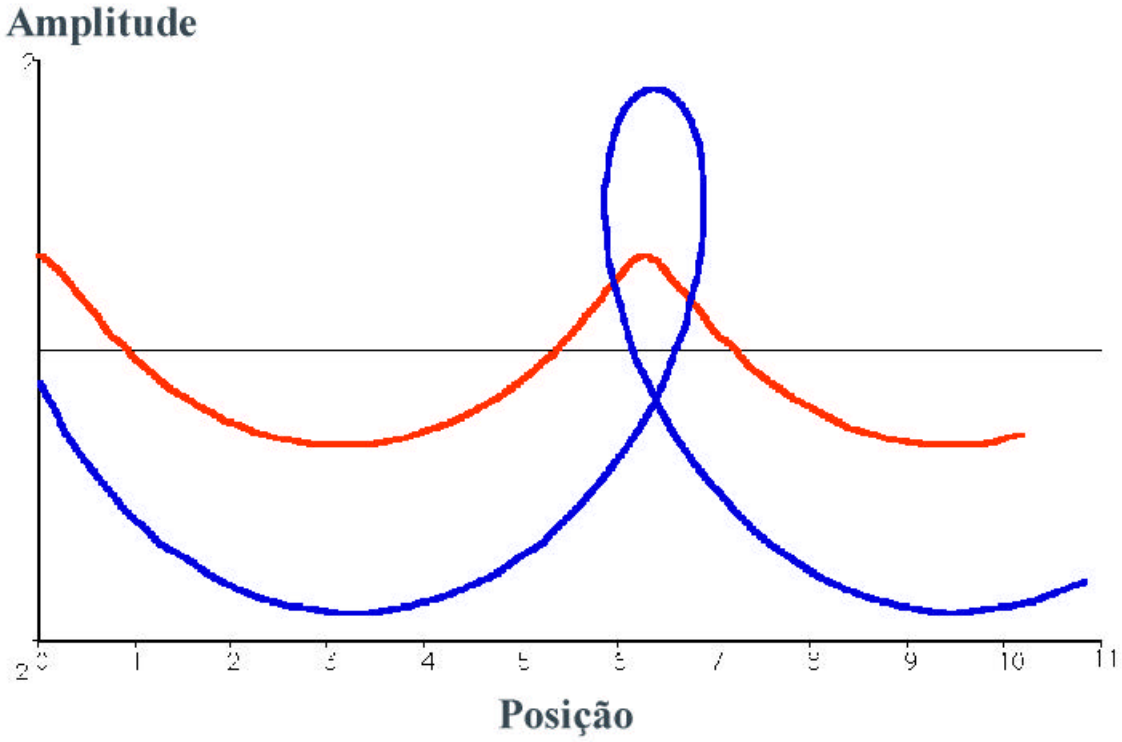


Figura 4.11 - Movimento de uma partícula numa superfície seguindo o modelo de Gerstner para dois valores diferentes de amplitude.

As equações de Gerstner, contudo, apenas irão criar modelos realistas de ondas quando forem colocadas num somatório de equações com diversas amplitudes, freqüências e vetores de onda *K*. O somatório final será da seguinte forma:

$$X = x_0 - \sum_{i-1}^{N} (\frac{K_i}{k_i}).A_i.\text{sen}(K_i.x_0 - w_i t + f_i) \qquad Y = \sum_{i-1}^{N} A_i.\cos(K_i.x_0 - w_i t + f_i)$$

(4.6)

A figura 4.12 mostra o perfil de uma onda gerada com este tipo de equação.

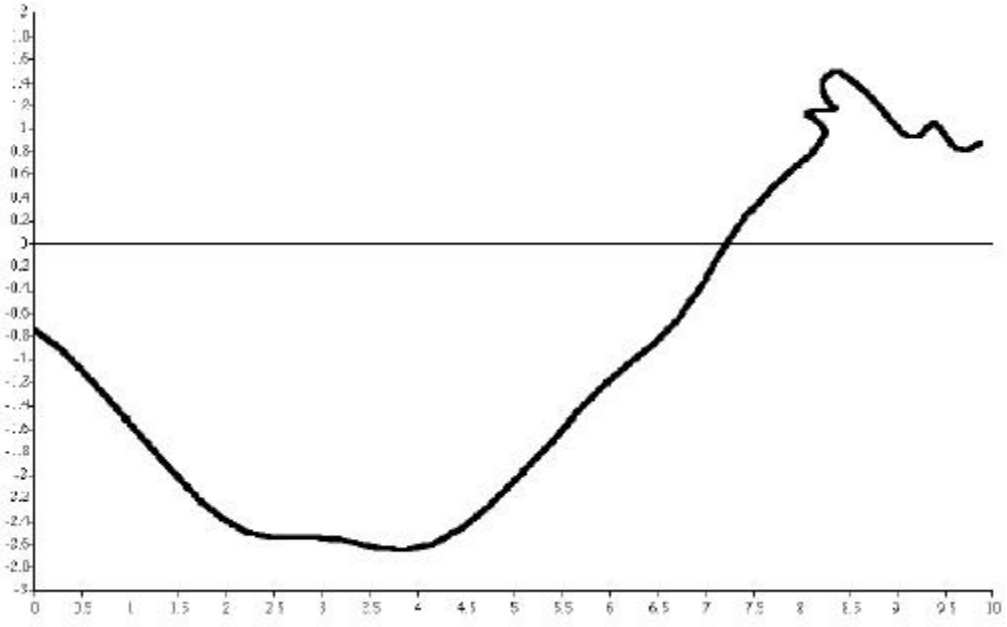


Figura 4.12 - Perfil de uma onda gerada com somatório de múltiplas equações de Gerstner .

Baseando-se neste modelo, podem-se criar movimentos mais realistas de ondas, especialmente para aquelas que possuem uma amplitude relativamente grande. Para um estudo mais detalhado sobre este método, ver [Tessendorf 99] e [Foster et al 99].

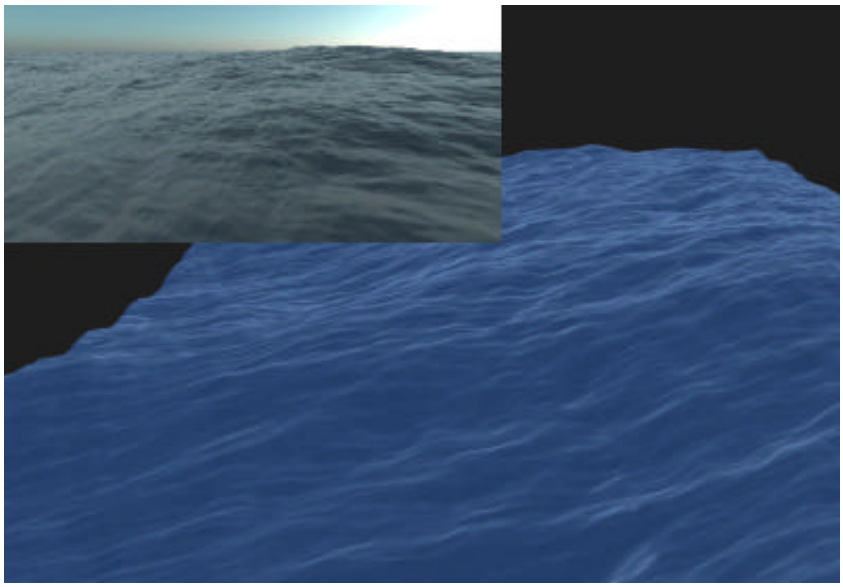


Figura 4.13 – Superfície gerada com somatório de múltiplas equações de Gerstner.

4.4 - Nuvens

Dentre todos os modelos da natureza, as nuvens são talvez os elementos mais complexos de serem modelados e visualizados, principalmente pelo fato de não possuírem uma superfície bem definida. Existem diversas abordagens que podem ser feitas na construção destes elementos:

- Gerar texturas procedimentais, que serão utilizadas posteriormente para mapeá-las sobre algum objeto;
- Gerar um modelo tridimensional de uma camada de nuvens;
- Gerar o modelo tridimensional de uma nuvem.

4.4.1– Síntese de texturas de nuvens

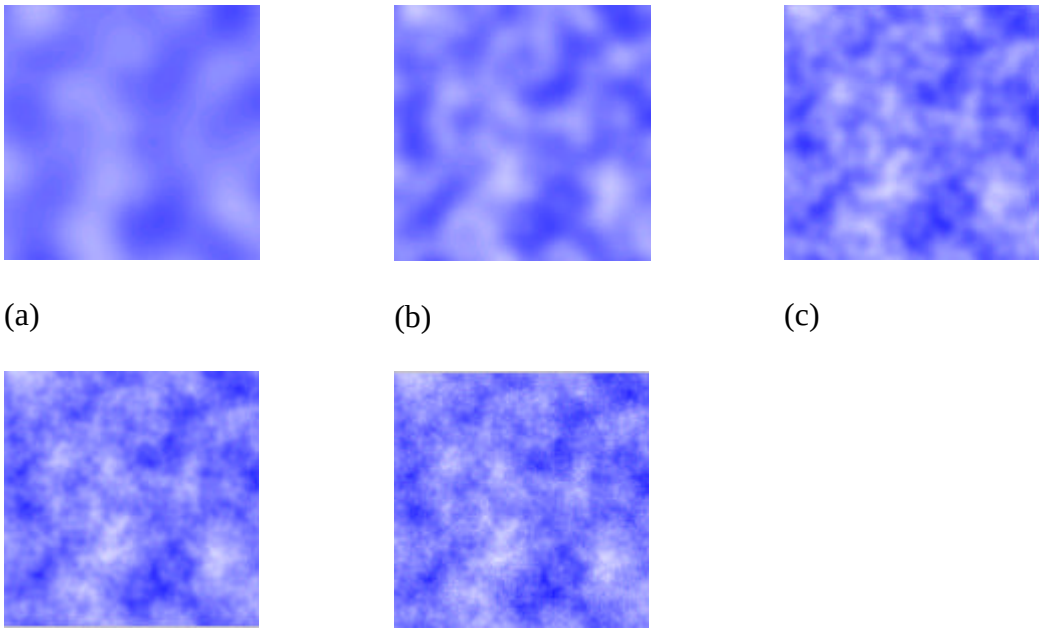
Os primeiros resultados relevantes obtidos na tentativa de sintetizar imagens de nuvens procedimentalmente são atribuídos a Gardner [Gardner 85] e a Ken Perlin [Perlin 85]. Ambos observaram que aplicando diretamente suas funções fractais sobre uma matriz bidimensional, a imagem formada assemelhava-se a um céu com nuvens. Gardner utilizou uma série de Fourier de cossenos [Gardner 94] e Perlin utilizou sua função *Turbulência*. Posteriormente, Musgrave [Ebert et al 94] e Worley [Worley 96] também sintetizaram imagens interessantes utilizando a fBm e a iteração fractal da função básica baseada em distância celular, apresentada na seção 3.1.2.

Uma vez construída a função fractal adequada, sintetizar uma imagem de nuvens torna-se bastante simples, bastando mapear as cores de forma correta. Para gerar as figuras 4.14 , (a) a (e) foi utilizada a função Turbulência, aplicada da seguinte forma:

Para cada ponto P = (x, y) da imagem faça

*Cor_Pixel(x, y) = (255, 255, Turbulência (x, y) * 255)*

onde Cor_Pixel é da forma (R, G, B)



(a) (b) (c)

(d) (e)

Figura 4.14 - Função Turbulência mapeada com cores brancas e azuis, com diversas iterações fractais: (a) 2 iterações fractais (b) 3 iterações fractais (c) 4 iterações fractais (d) 5 iterações fractais (e) 8 iterações fractais.

4.4.2 – Modelos tridimensionais de camadas de nuvens

Apresenta-se a seguir uma proposta inovadora para a construção de camadas de nuvens, lançando mão das hipertexturas, apresentadas na seção 2.2.2. [Clua et al 98]

De acordo com a esta técnica define-se inicialmente uma região *fuzzy*, que é, no volume da cena, um paralelepípedo alinhado com os eixos e definido por 2 vértices:

v1 = (0, Altura Inicial das nuvens, 0)

v2 = (largura, Altura Final das nuvens, profundidade)

Todas as nuvens estarão dentro deste retângulo, que será dividido em três fatias horizontais. Denomina-se à fatia inferior de *zona nebulosa inferior*, à fatia do meio de *zona nebulosa central* e à fatia superior de *zona nebulosa superior*.

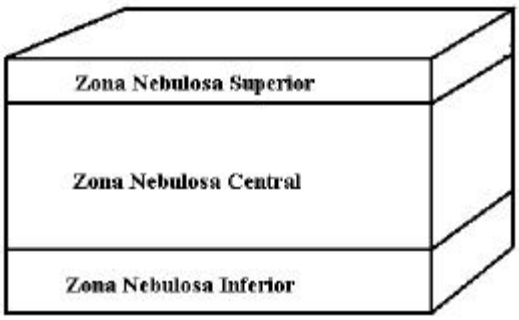


Figura 4.15 - Diversas regiões *fuzzy* para as camadas atmosféricas de nuvens.

Normalmente, o processo de geração de nuvens deve ser o último passo da criação de um cena, como geralmente acontece quando se trabalha com hipertexuras, pois deve-se dar prioridades a outros elementos que já vinham ocupando um espaço anteriormente, não permitindo que a nuvem os substitua (um pedaço de uma montanha, por exemplo, não pode deixar de existir por causa da inserção de uma nuvem).

Para cada *voxel* que estiver dentro da região nebulosa central utiliza-se a função fractal apropriada (neste trabalho, a fBm), e o valor fornecido como retorno corresponde à densidade deste ponto.

Note-se que, se a zonas extremas (superior e inferior) forem de espessura 0, a camada de nuvens acabará abruptamente, dando uma aparência irreal. Estas zonas, portanto, têm um tratamento diferente, de forma a amortecer o encontro de uma região com nuvens com outra sem nuvens. Assim, para cada *voxel* que esteja na zona nebulosa inferior ou na zona nebulosa superior, pode-se dar o seguinte tratamento na sua respectiva densidade d :

$$d = \left(\frac{d}{\Delta Z} \right)^n \times fBm(ponto) \tag{4.7}$$

onde d é a distância do ponto até a zona central, ΔZ é a espessura da camada da zona onde o ponto se encontra e n é o expoente que aumentará ou diminuirá a velocidade de variação da densidade à medida que se afasta da zona central.

Ainda assim, podem-se obter resultados não tão satisfatórios com esta fórmula, pois a queda de densidade ocorre homogeneamente à medida que se afasta da zona central. Para corrigir este problema pode-se acrescentar à equação um valor aleatório. Adequa-se bem a esta situação uma chamada à função *noise*. Assim, a equação resultará em:

$$d = \left(\frac{d}{\Delta Z} \right)^n \times fBm(ponto) \times noise(ponto) \tag{4.8}$$

Em geral, a parte inferior das camadas de nuvens tende a ser mais achatada que a parte superior, devido às massas de ar quente que sobem. Para simular este efeito, basta colocar uma espessura maior para a zona nebulosa superior do que para a zona nebulosa inferior. Além disso, podem-se colocar expoentes *n* diferentes para cada uma das regiões.

(a) (b) (c)

Figura 4.16 - cortes seccionais em camadas de nuvens colocadas no espaço. A densidade e a espessura de cada uma possuem valores diferentes.

4.4.3 – Modelagem de uma nuvem 3D

Vários trabalhos foram realizados na tentativa de criar modelos realistas de nuvens 3D. Neyret [Neyret 97] obteve resultados convincentes, usando para isto formulações físicas reais. Contudo, este método é pouco flexível e relativamente complexo de ser manipulado. Ebert, em [Ebert 97] e [Ebert 99] faz uma abordagem utilizando funções procedimentais aplicadas em volumes, obtendo resultados também realistas e de fácil manipulação para um usuário final. Este é o método que será abordado neste trabalho, sendo que algumas pequenas alterações foram adicionadas pelo autor.

A modelagem de uma nuvem 3D volumétrica possui duas etapas: a definição da sua macroestrutura, que será criada através de superfícies implícitas, e da sua

microestrutura, que será criada através de funções procedimentais. Juntando-se estas duas componentes irá obter-se o modelo final da nuvem.

As funções implícitas consistem numa técnica de modelagem, que devido a um campo de atração criado entre os objetos ao colocá-los relativamente próximos entre si, serão gerados modelos resultantes de uma fusão suave entre estes. Para um estudo mais detalhado sobre superfícies implícitas ver [Bloomenthal et al 97].

Utilizar esferas como superfícies implícitas consistirá num método adequado para modelar a macroestrutura da nuvem. Para modelar esta estrutura, basta posicionar o centro das esferas no espaço, de maneira a dar uma forma geral para o elemento. Esta distribuição também pode ser feita automaticamente, realizando uma distribuição de pontos aleatórios no espaço, sendo que estes pontos correspondem ao centro das esferas.

Para gerar o objeto composto pelas superfícies implícitas, deve-se computar a densidade para cada *voxel* devido a cada uma das funções implícitas inseridas no espaço. Existem várias funções que servem para calcular esta densidade, sendo uma das mais tradicionais, e que se adequam perfeitamente aos propósitos do trabalho, a função cúbica de Wyvill [Wyvill et al 86]:

$$F(r) = -\frac{4.r^6}{9.R^6} + \frac{17.r^4}{9.R^4} - \frac{22.r^2}{9R^2} + 1 \qquad (4.9)$$

onde r é a distância de um ponto ao centro da superfície implícita e R é o raio da superfície implícita. Nesta função, r deverá variar apenas entre 0 e R. Caso r seja maior do que R, então atribui-se o valor de R a r, e neste caso o resultado da função será zero.

O resultado final de um sistema de funções implícitas será dado pela seguinte somatória [Wyvill et al 86]:

$$d(V_{ijk}) = \sum_{j=1}^{IS} w_j F_j(r)$$

(4.10)

onde *d* será a densidade final do *voxel* V_{ijk} relativo ao sistema de funções implícitas, descritas pelas funções F_j, sendo *r* a distância deste *voxel* ao centro da superfície F_j, w_j é o peso que a F_j terá e *IS* o número total de funções implícitas que compõem o sistema.

Figura 4.17 – Macroestrutura criada utilizando esferas descritas por equações implícitas.

Por outro lado, a microestrutura da nuvem, como foi visto, será criada por alguma função procedimental (A função *turbulência* produz bons resultados). Assim, cada *voxel* V_{ijk} do interior do volume que contém a nuvem 3D deve terá uma densidade dada por a uma função procedimental:

$$d(V_{ijk}) = turbulência\ (V_{ijk})$$

(4.11)

Para realizar a composição da macroestrutura com a microestrutura, deve ser feita uma combinação do resultado obtido nos dois cálculos de densidade:

$$d = (B \,.\, d_1 \;\; + (1\text{-}B) \,.\, d_2) \tag{4.12}$$

onde $B \in [0,1]$ e corresponde a um parâmetro de agrupamento para as duas densidades.

Para obter resultados mais adequados pode-se perturbar as coordenadas dos pontos do interior do volume que conterá a nuvem. Para isto, de acordo com [Ebert 99], pode-se aplicar a função *noise* ou *turbulência* sobre estes pontos.

Assim, o algoritmo para gerar uma nuvem 3D pode ser resumido da seguinte forma:

Nuvem_Volumétrica (ponto, Coef_Mesclagem)

Ponto_Perturbado = Coordenadas perturbadas por noise ou turbulência;

Densidade1 = Sistema de funções implícitas (Ponto_Perturbado);

Densidade2 = turbulência (Ponto_Perturbado);

*Densidade=Coef_Mescalegem*Densidade1+(1 – Coef_Mesclagem)*Densidade2*

Densidade = Controle de contraste de densidade (exponeciação, p. exemplo)

Retorne (Densidade);

Figura 4.18 – Nuvem formada aplicando-se a função turbulência sobre a macroestrutura da figura 4.17 [Ebert 99].

Através desta função, pode-se criar um variado número de tipos de nuvens diferentes. Isto será controlado especialmente pelo parâmetro *Coef_Mesclagem*, que determina se o modelo é predominantemente definido pelas superfícies implícitas ou pelas funções procedimentais e por possíveis alterações das funções procedimentais e das superfícies implícitas. Assim, Ebert exemplifica em [Ebert 99] que para modelar uma nuvem do tipo *Cumulus* deve-se utilizar superfícies implícitas elípticas, um coeficiente de mesclagem em torno de 40% e um fator de exponenciação em torno de 0.5 sobre a densidade de cada *voxel*. Para nuvens do tipo *Cirrus* deve-se utilizar poucas superfícies implícitas esféricas, um coeficiente de mesclagem em torno de 30% e uma densidade pequena para cada *voxel*.

Este tipo de modelagem de nuvens permite, de forma simples, que o usuário crie nuvens com formas específicas, como uma letra ou algum animal, bastando para isso posicionar as superfícies implícitas de forma adequada.

Figura 4.19 – Simulação de uma nuvem do tipo *cirrus*, utilizando os parâmetros descritos anteriormente [Ebert 99].

Para gerar animações destas nuvens, o processo também é simples. Para realizar transformações globais (escala, rotação e translação), basta aplicá-las diretamente sobre as superfícies implícitas. Para realizar animações internas ao modelo (como por exemplo simular o "nascimento" de uma nuvem), basta realizar alterações nos parâmetros das funções procedimentais e nos valores de densidade.

4.5 - Gases

A modelagem de gases é na verdade uma generalização com relação à modelagem de nuvens. Pode ser utilizada quando se deseja criar diversos tipos de fumaça, neblinas ou até mesmo alguns efeitos atmosféricos.

Para criar este tipo de elementos deve-se utilizar, assim como nas nuvens, a modelagem volumétrica, baseada em hipertexturas (ver seção 2.2.2) e para realizar a visualização, deve-se utilizar modelos específicos de iluminação (ver seção 5.1).

Há uma série de técnicas para dar “forma” aos gases. [Ebert et al 94] descreve uma função de potenciação genérica para realizar este cálculo:

$$densidade(V) = (F(turbulência(V)) \; . \; K1)^{K2} \qquad (4.13)$$

Esta função realiza uma perturbação nas coordenadas do *voxel V* utilizando a função *turbulência* e a seguir aplica ao resultado obtido alguma função *F* conveniente. *K1* é um coeficiente que será multiplicado sobre o resultado, de forma a poder realizar uma escala na densidade e *K2* é a constante de exponenciação para a expressão. O modelo criado irá depender da função *F* e dos coeficientes *K1* e *K2* escolhidos. Observe-se que se o valor de densidade deve ser restrito ao intervalo [0,1] dos reais, então $F(x) \subset [0,1]$ e *K1* ≤ 1.

Por exemplo, para criar uma névoa, [Ebert et al 94] sugere utilizar os seguintes parâmetros: *F(x) = x/MAX_TURB*, *K1* = 0.5 e *K2* = 3, onde MAX_TURB é o maior valor que *turbulência* pode gerar no volume sendo utilizado.

Para criar uma coluna de fumaça simples, pode-se utilizar como parâmetros *F(x)=sen(x), K1* = 0.57 e *K2* = 6.

Partindo desta definição, pode-se realizar uma série de alterações e ajustes. A coluna de fumaça citada acima, por exemplo, possui um inconveniente: independentemente da altura, a densidade média da fumaça é a mesma. Uma coluna mais realista deveria ter a densidade decrescente à medida que a fumaça sobe. Também pode-se detalhar mais com relação aos objetos que delimitam os volumes. Ao invés de serem simples cubos, podem-se utilizar objetos mais sofisticados.

4.6 – Outros elementos da natureza

Outros modelos da natureza não são abordados neste trabalho, principalmente pelo fato de não serem conhecidas propóstas para aproximações procedimentais muito adequadas, como é o caso das plantas e flores, que podem ser modeladas utilizando o método estrutural, como foi exposto na seção 2; o fogo e a neve, que podem ser modelados por sistemas de partículas [Aguilar 98]. Contudo, é de extremo interesse para a computação gráfica encontrar e aperfeiçoar soluções procedimentais para estes elementos também, de forma a usufruir das vantagens que este método de modelagem traz, como já foi exposto anteriormente.

Capítulo 5

Iluminação e Visualização

O principal objetivo deste trabalho consiste em realizar um estudo sobre a modelagem dos elementos da natureza. Ficaria incompleto, no entanto, se não se apontassem algumas diretrizes com relação à iluminação e à visualização dos modelos discutidos, uma vez que os algoritmos tradicionais não se adequam completamente para alguns casos. Nesta seção, além disso, irá apresenta-se uma sugestão de trabalho futuro, que são as texturas volumétricas. Embora não seja um assunto diretamente ligado à modelagem de elementos da natureza, está colocado aqui pelo fato de sua utilização acarretar numa otimização no processo de visualização para elementos da natureza gerados de forma volumétrica.

5.1 - Iluminação

Para os modelos com superfície "sólida" e bem definida, como os terrenos e a água, os modelos de iluminação tradicionais adequam-se perfeitamente (os de Phong [Phong 75] e Blinn [Blinn 77], por exemplo). O maior problema está com relação aos modelos de nuvens e gases, que necessitam um modelo de iluminação dedicado. Descreve-se a seguir uma formulação, proposta inicialmente por [Blinn 82]. Embora os

autores tenham realizado este trabalho para nuvens compostas por sistemas de partículas, pode-se utilizá-lo, sem perda de generalidades, para modelos criados por volumes, uma vez que um *voxel* pode ser tratado como um caso particular de uma partícula. Este modelo pode ser também utilizado para o cálculo de iluminação de fumaça e gases em geral, bastando retirar alguns elementos da equação principal.

A iluminação é descrita basicamente pela seguinte equação:

$$B = \sum_{t_{início}}^{t_{fim}} e^{-t \cdot \sum_{t_{início}}^{t} r(x(u),y(u),z(u)).\Delta u} .I.r(x(t), y(t), z(t)).\Delta t \tag{4.9}$$

onde $t_{início}$ é o ponto de entrada do volume e t_{fim} é o ponto de saída, r é densidade do material em função das coordenadas x, y, z de um ponto no seu interior. O coeficiente t da equação 4.9 é chamado de comprimento óptico do material e é calculado pela seguinte formula:

$$t = \sum_{t_{início}}^{t} b.r(x(u), y(u), z(u)).\Delta u \tag{4.10}$$

onde b é o coeficiente de atenuação sobre o percurso no interior do volume.

Ainda na equação 4.9, o termo I corresponde ao cálculo da fase de um raio que percorre o volume ligando o ponto de entrada à saída. Esta fase corresponde ao brilho que uma partícula do volume terá, levando em conta o ângulo de entrada do raio.

$$I = \sum_i I_i (x(t), y(t), z(t)).\, fase(q) \tag{4.11}$$

onde $I_i(x(t), y(t), z(t))$ é a intensidade de luz provinda da fonte *i* refletida sobre o elemento em questão.

Para o cálculo da fase [Cornette et al 92] afirma que um bom resultado pode ser conseguido através da seguinte equação:

$$fase(q, g) = \frac{3(1-g^2)}{2(2+g^2)} . \frac{(1+\cos^2 q)}{(1+g^2-2g.\cos q)^{3/2}} \tag{4.12}$$

onde *g* é o fator de assimetria e seu valor depende do tipo e condições do gás.

Um modelo ideal e completo para iluminação de nuvens, no entanto, deve levar em conta uma série de outros fatores não considerados aqui, tais como a luz provinda da atmosfera, a luz provinda da reflexão dos raios do sol na terra e raios de luz refletidos de outras partículas da nuvem. Para a descrição de um modelo mais específico para nuvens, ver [Nishita et al 96].

5.2 – Visualização

Não fosse a presença de volumes, o método mais adequado para visualizar cenas envolvendo elementos da natureza seria o tradicional *Ray-Tracing* [Glassner 89]. Contudo, deve-se realizar uma extensão para este modelo, de forma a suportar elementos volumétricos, gerados por exemplo pelas hipertexturas.

Propõe-se assim uma técnica híbrida entre o *Ray Tracing* e o *Ray Casting* primeiramente descrita por Sobierajski e Kaufman [Sobierajski et al 94].

Esta técnica consiste em lançar raios partindo de um observador em direção a um *pixel* de um *grid* definido. Inicialmente trata-se cada um dos volumes gerados como

cubos inseridos no espaço, independentemente do elemento que estes representam. Podem existir na cena outros objetos geométricos, tais como esferas, planos, etc. O algoritmo começa funcionando como se fosse um *Ray Tracing* tradicional. Ao encontrar uma interseção com um destes cubos que representam um volume, deve-se determinar o ponto de entrada e saída do raio sobre este objeto. A partir dai, o algoritmo passa a comportar-se como um *Ray Casting* e o raio passa a percorrer o interior do volume de forma discreta, ou seja, visitando cada *voxel* que possui interseção com o raio . Ao realizar isto, acumula-se uma cor para este raio, assim como o coeficiente de transparência, com base na definição do *voxel* em questão. Repete-se isto até atingir o segundo ponto de interseção do cubo, o qual corresponde à sua saída. Caso o raio encontre em algum instante uma nova superfície ainda dentro do volume, deverá lançar um raio recursivo referente à reflexão e outro referente à transmissão, caso haja transparência. O resultado de cada um destes raios será acumulado no *voxel* que lhes deu origem. Ao sair do volume, o algoritmo volta a comportar-se como um *Ray Tracing*.

A equação clássica da iluminação utilizada no *Ray Tracing* [Glassner 89] não se adapta perfeitamente a este algoritmo, por isto adota-se a equação descrita por [Sobierajski et al 94], que é apenas uma extensão da primeira:

$$I_l(x,\vec{w}) = I_{vl}(x,x',\vec{w}) + t_l(x,x')I_{sl}(x',\vec{w}) \tag{5.1}$$

$I_{sl}(x',\vec{w})$ é a intensidade de iluminação sobre a superfície, e pode ser calculado como no modelo tradicional do *Ray Tracing*, ou seja,

$$I_{sl}(x', \vec{w}) = I_{al} k_{dl} +$$

$$\sum_{i=1}^{nL} A_i I_{pli} [k_{dl} (\vec{N} \cdot \vec{L}_i) + k_{sl} (\vec{N} \cdot \vec{H}_i)^n] +$$

$$k_{sl} I_{sl}(x', \vec{w}_r) + k_{tl} I_{sl}(x', \vec{w}_t)$$

(5.2)

onde x' é o primeiro ponto de interseção encontrado pelo raio $\vec{w}$, nL é o número de fontes de luz da cena, I_{pli} é a intensidade da i-ésima fonte de luz, k_{dl}, k_{sl} e k_{tl} são respectivamente os coeficientes de difusão, reflexão e transmissão da superfície, $\vec{N}$ é a normal da superfície no ponto x', $\vec{L}_i$ é o vetor do raio de luz da i-ésima fonte de luz, A_i é a atenuação da i-ésima fonte de luz e n é o coeficiente de especularidade. Além disso, $I_{sl}(x', \vec{w}_r)$ e $I_{sl}(x', \vec{w}_t)$ são as contribuições sobre x' da luz provinda da direção do reflexo especular e da transmissão, respectivamente.

O termo $t_l(x, x')$ da equação 5.1 corresponde à atenuação da luz, devido aos volumes da cena. Este fator pode ser calculado da seguinte forma:

$$t_l(x, x') = e^{-\int_x^{x'} \sum_{i=1}^{nV} s_{al}(i,t) + s_{scl}(i,t) dt}$$

(5.3)

onde nV é o número de volumes na cena e $s_{al}(i,t)$ e $s_{scl}(i,t)$ correspondem à quantidade de luz absorvida e espalhada, respectivamente, para um volume i na posição t.

$I_{vl}(x, x', \vec{w})$, da equação 5.1, corresponde à componente volumétrica de iluminação e baseia-se na teoria do transporte e propagação da luz citada em [Kruger 91]:

$$I_{vl}(x, x', \vec{w}) = \int_{x}^{x'} \sum_{i=1}^{nV} [I_{El}(i,t) +$$

$$s_{scl}(i,t) \int F_{scl}(\vec{w}', \vec{w}) I(t, \vec{w}') d\vec{w}'] t_{l}(x,t) dt$$

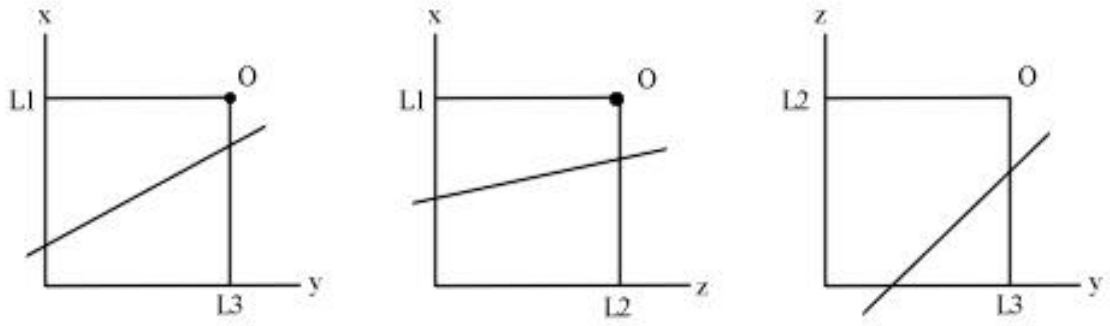
(5.4)

onde $I_{El}(i,t)$ corresponde à intensidade de luz emitida para a posição *t* pelo volume *i* e $F_{scl}(\vec{w}', \vec{w})$ é a função de espalhamento, indicando a porcentagem de luz provinda da direção $\vec{w}'$ que sai na direção $\vec{w}$.

Para percorrer o interior do volume com o raio traçado e avaliar os *voxeis* individualmente é necessário um algoritmo que percorra todos os valores discretos de uma reta do R^3. Descreve-se a seguir um algoritmo que gera a sequência de todos os *voxels* visitados por uma reta 3D [Cohen 94] e [Fujimoto et al 86]. Este algoritmo assemelha-se de certa forma ao de rasterização 2D descrito por Bresenham [Bresenham 65], sendo uma extensão para o espaço 3D. Também será utilizado para realizar traçamentos recursivos dos raios de ordens inferiores.

A idéia básica consiste em analisar qual é a face do *voxel* pelo qual o raio que o atravessa sai, de forma a determinar em qual dos *voxels* vizinhos o raio irá penetrar.

Assume-se que a direção da linha é positiva, sem perda de generalidade. Pode-se concluir portanto, que a linha vai ou para o *voxel* lateral, ou para o *voxel* de cima ou para o *voxel* de frente. Estas 3 faces compartilham de um mesmo vértice O, sendo que os 3 eixos que saem deste vértice são os que conectam as faces adjacentes. (veja figura 5.1)



Vista de frente Vista lateral Vista de cima

Figura 5.1 – Possíveis situações de uma reta atravessando um *voxel*.

Assim, realizando um teste entre a reta e as 3 linhas L1, L2, L3, pode-se descobrir qual será o *voxel* para o qual o raio irá. A equação implícita da projeção da linha sobre cada um dos 3 planos permitirá realizar este teste. Antes disto deve-se converter as coordenadas do ponto em que a reta entra e sai do volume em valores inteiros.

Desta forma, chama-se de e_{xy} , e_{xz} , e_{yz} a projeção da reta sobre cada um dos três planos. As equações implícitas destas retas projetadas serão:

$e_{xy} = y\Delta x - x\Delta y - c_{xy} = 0$ ➔ linha que vai de (x_0, y_0) a $(x_0 + \Delta x, y_0 + \Delta y)$

$e_{xz} = z\Delta x - x\Delta z - c_{xz} = 0$ ➔ linha que vai de (x_0, z_0) a $(x_0 + \Delta x, z_0 + \Delta z)$

$e_{zy} = y\Delta z - z\Delta y - c_{zy} = 0$ ➔ linha que vai de (y_0, z_0) a $(y_0 + \Delta y, z_0 + \Delta z)$

Assim, testando e_{xy} sobre o ponto O, é possível saber se esta cruza a linha L1 ou L3. Da mesma forma, testando e_{xz} sobre O, sabe-se se a reta projetada em xz cruza a

linha L1 ou L2 e testando e_{yz} sobre O, descobre-se se a reta projetada em yz cruza a linha L2 ou a linha L3. Caso a projeção e_{xy} cruze L1, então sabe-se que a reta original não irá passar para o *voxel* que está acima. Caso a projeção e_{xz} cruze L2, sabe-se que a reta original não irá passar para o *voxel* que está em frente ao *voxel* corrente e caso a projeção e_{yz} cruze L3, sabe-se que a reta não irá passar para o *voxel* lateral ao corrente.

Assim, fazendo apenas 2 testes determina-se qual será o *voxel* pelo qual a reta irá continuar seu caminho. Uma das grandes vantagens deste algoritmo é que utiliza apenas operações que envolvem números inteiros, sem utilizar operações trigonométricas.

Outra característica do algoritmo é o fato de que garante que todos os *voxels* que possuem alguma interseção não nula com a reta irão ser percorridos.

Utilizando este algoritmo, pode-se realizar a varredura dos raios sobre um *grid* criado com as definições próprias da camera. Inicialmente deve-se tratar o volume como sendo um cubo 3D inserido no espaço. Apenas depois de encontrar o ponto de entrada e saída do raio sobre este objeto é que passa-se a encarar este cubo como um volume.

5.3 – Texturas Volumétricas

Esta é uma nova técnica, elaborada ao longo desta pesquisa, com a função de visualizar elementos da natureza modelados volumetricamente em tempo real. Criar objetos através de volumes numa cena acarreta num encarecimento muito grande do processamento, podendo em alguns casos inviabilizar o projeto. Como exemplo pode-se citar uma aplicação em que se exija uma visualização rápida ou até em tempo real, o que é muito comum em aplicações que utilizem elementos da natureza (como em jogos ou visualização de terrenos).

O principal fator que torna esta visualização demorada consiste no fato de que, pelas técnicas tradicionais de *rendering*, ao gerar uma parte da imagem que contenha um pedaço do objeto volumétrico, deve-se penetrar neste volume e realizar um processamento (cálculo de densidade/transparência) dedicado para cada *voxel* do caminho percorrido. Suponha-se, por exemplo, que se deseje visualizar uma cena utilizando o método proposto na seção 5.2 e com um objeto volumétrico de uma resolução de 1000x1000x1000 *voxels*. Neste caso, para cada pixel da imagem a ser visualizado e que contenha o volume, em média 1000 *voxels* deverão ser percorridos um a um (este número será maior ou menor, dependendo do ângulo e do local de entrada do raio sobre o volume). Caso a visualização global esteja sendo feita com o Ray-tracing, isto pode agravar-se mais ainda, já que para cada pixel visualizado, devido às recursões, pode ser necessário penetrar mais do que uma vez no volume.

As texturas volumétricas, como foram chamadas neste trabalho, consistem numa tentativa de acelerar este cálculo, realizando um pré-processamento sobre o volume.

Este pré-processamento inicia-se através da definição de 6 matrizes, uma para cada face do volume que compõe o objeto em questão (a resolução destas matrizes será a mesma da face a ela associada).

Cada componente desta matriz irá corresponder a um *voxel* de entrada do volume. Calcula-se a seguir o valor de densidade acumulado para um raio que entra no volume por cada um dos *voxels*. Apenas isto, no entanto, não é suficiente, já que o percurso deste raio irá depender do seu ângulo de entrada sobre o volume. Deve-se, na verdade, pré-calcular diversas densidades acumuladas, variando gradativamente este ângulo de entrada para cada componente da matriz, como pode ser visto na figura 5.2 e ir armazenando

estes valores numa tabela, que em alguns casos armazenará também alguma outra informação necessária para a visualização, tal como a normal do elemento para aquele ângulo de entrada correspondente.

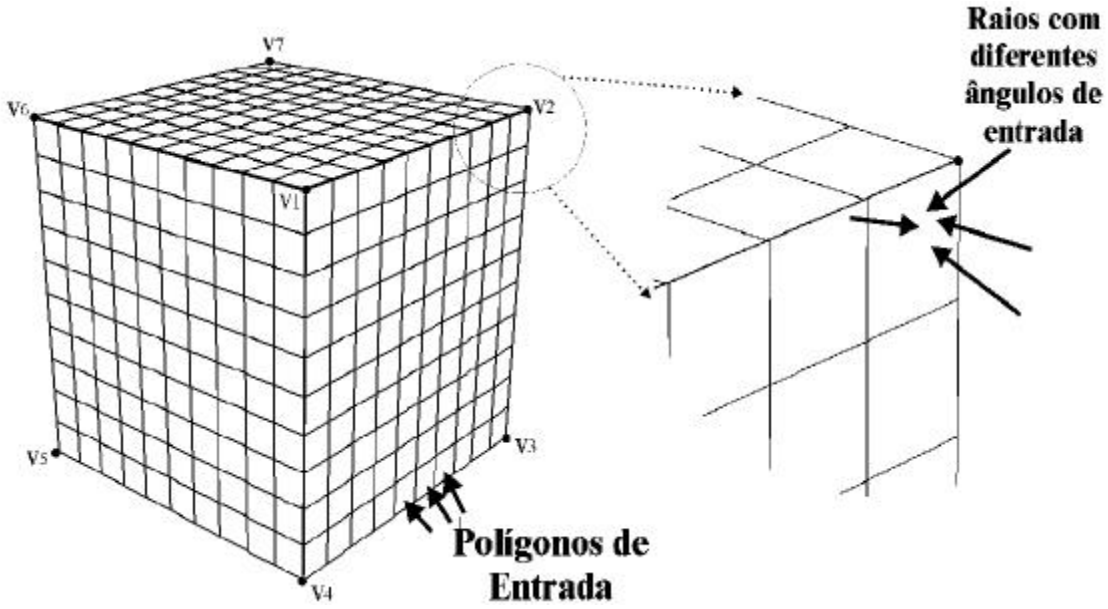


Figura 5.2 – O volume é dividido em polígonos de entrada. Para cada polígono há uma tabela para diferentes ângulos de entrada, onde será armazenada a densidade total para um raio que entre por este polígono com um determinado ângulo.

O tamanho da tabela irá influenciar na idéia de profundidade do objeto volumétrico. Se for pequena, quando o observador se movimentar em torno do elemento, a face do volume irá variar pouco, dando a impressão de uma imagem mapeada sobre uma caixa.

A visualização deste volume resume-se em calcular a interseção do raio de visualização com a caixa que envolve o volume, e a seguir realizar uma busca na tabela correspondente a essa posição da matriz, de forma a saber qual a densidade acumulada para este raio lançado. Ao consultar esta tabela, pode-se ao invés de simplesmente buscar

o ângulo sólido mais próximo ao raio lançado, realizar uma interpolação com os valores de densidade para os ângulos vizinhos. Desta maneira garante-se uma transição suave na densidade quando o observador se mover em torno do volume.

Esta técnica adequa-se bem em casos de objetos volumétricos com poucos detalhes geométricos (como é o caso de nuvens, gases e fumaça), mas perde qualidade quando os objetos são muito detalhados (visualização médica, por exemplo), já que para não perder definição, acaba sendo necessário criar tabelas muito grandes para cada voxel de entrada, de forma a pré-calcular densidades para variações pequenas do ângulo sólido.

O tamanho destas texturas volumétricas será proporcional ao tamanho da tabela de ângulos. Supondo-se que esta tabela contenha valores de densidade para variações de 30^o nos ângulos de entrada, então esta tabela terá 25 posições. Tomando um volume com a resolução de 1000 x 1000 x 1000, o tamanho de cada matriz será de 25.000.000. Como são 6 matrizes para representar o volume completo, então o tamanho total será de 150.000.000, enquanto que o volume original possui um tamanho de 1.000.000.000. Neste caso, desconsiderando possíveis compressões, além das texturas volumétricas proporcionarem uma aceleração grande na visualização, irão requerer um espaço de armazenamento menor que o volume, já que uma vez construídas, o volume pode ser totalmente descartado (na verdade, os volumes nem sequer precisam ser construídos, já que as matrizes podem ser montadas utilizando diretamente as funções procedimentais).

Para a implementação desta otimização, é necessário desenvolver uma técnica de consulta rápida para a tabela de ângulos, assim como uma técnica eficiente para realizar a interpolação do valor das densidades. Além disso, podem-se desenvolver técnicas para

realizar uma compressão no armazenamento dos valores de densidade, já que em muitos casos irá repetir-se para vários ângulos de entrada em cada componente das matrizes.

5.4 - Refinamento de texturas 2D utilizando funções procedimentais

Esta técnica foi criada neste trabalho, e apresentou resultados interessantes no processo de visualização de uma cena contendo terrenos e texturas associadas [Clua et al 99a].

Ao realizar-se o mapeamento de uma textura discreta, normalmente uma imagem, sobre uma superfície 3D, surgem problemas relacionados à amostragem desta textura, pelo fato da superfície estar inserida num espaço contínuo enquanto a textura está num espaço discreto. Este problema é conhecido na computação gráfica como *aliasing*, e existe uma série de métodos para tentar suavizá-lo, já que corrigi-lo é impossível uma vez que não se conhece os valores para todos os pontos que se deseja mapear. Pode-se expressar este problema da seguinte forma: para um dado conjunto de pontos $P = \{ P_1, P_2, ..., P_i \}$, $i > 1$, pertencentes a uma superfície geométrica *S*, contínua no espaço 3D, de forma que todos estes pontos estejam dentro de uma esfera de raio ε, a função discreta *f* (por exemplo, o mapeamento de uma imagem) irá retornar um mesmo resultado para todos os elementos contidos em *P*, causando o surgimento de *jaggies*. Este problema torna-se bem visível quando se mapeia sobre uma superfície uma imagem de resolução pequena.

Uma primeira tentativa para suavizar este efeito consiste no método de interpolação. No entanto, o método do refinamento, ao invés de simplesmente procurar

interpolar *pixels* para diminuir o efeito dos *jaggies*, procura acrescentar dados a uma imagem, por meio de uma função procedimental.

Para realizar este refinamento, em primeiro lugar deve-se analisar que função se adequa ao conjunto de texturas que se deseja corrigir. Este é o principal fator que irá determinar se o refinamento irá produzir resultados convenientes ou não. Esta escolha pode ser feita automaticamente pelo programa ou pode ser realizada manualmente. A escolha automática pode não ser tão precisa mas em muitos casos será necessária. Tome-se como exemplo uma cena de uma paisagem com diversos elementos da natureza. Neste caso, provavelmente se desejaria uma função para refinar as texturas das rochas, outra para as da terra, outras para as da água, etc. Esta escolha pode ser baseada em vários critérios. Poder-se-ia, por exemplo, analisar o padrão de um pedaço da textura que compõe um elemento e tentar encontrar uma função adequada. Contudo, este seria um método caro em termos de processamento. Uma forma mais simples consiste simplesmente em realizar a escolha da função mais adequada através da cor do *pixel* que se deseja refinar. Se o *pixel*, por exemplo, for azul, procura-se a função apropriada para água, se for verde, uma função adequada para grama e assim por diante.

Como se viu anteriormente, o problema do *aliasing* na aplicação de texturas surge pelo fato de que para um conjunto de pontos *P* de uma dada superfície, o valor de um mapeamento de textura utilizando uma imagem é o mesmo, ou seja, um mesmo *pixel* de uma textura está sendo mapeado para mais de um ponto pertencente a uma superfície geométrica. Assim sendo, inicialmente realiza-se uma chamada para uma função, utilizando como parâmetro de entrada as coordenadas espaciais do ponto da superfície que está sendo visualizado. Este ponto tem como propriedade, numa situação normal, o

fato de que na cena apenas ele possui esta coordenada, uma vez que as superfícies são contínuas no espaço e nunca mais do que um elemento pode ocupar o mesmo espaço ao mesmo tempo.

No entanto, fazer isto para cada ponto de uma dada superfície pode implicar num encarecimento do processamento, já que uma chamada a uma função procedimental costuma exigir um processamento extra razoavelmente grande. Apresenta-se mais adiante uma forma de otimizar este processamento.

Uma vez calculado um valor para um ponto de um elemento utilizando uma função procedimental, deve-se realizar uma composição com a textura original.

Esta composição nada mais é do que uma combinação, que pode ser feita de diversas formas. O caso ideal seria substituir por completo a cor da textura mapeada pelo valor encontrado no cálculo da função. Isto no entanto poderá ser feito apenas quando a função descrever a textura que está sendo mapeada de forma adequada (por exemplo, o mapeamento de uma textura de areia sobre uma superfície, já que é conhecida uma função f_a capaz de simular de maneira adequada o padrão de textura de areia). Neste caso é possível substituir por completo a textura pela função, eliminando por completo os problemas de *aliasing*. Na maioria das vezes isto não acontece, logo deve-se aplicar outra estratégia, de forma a combinar a cor da textura mapeada para o ponto na superfície 3D com o valor encontrado pela função procedimental para este. Esta combinação pode ser feita utilizando a seguinte equação:

$$T(P) = Textura(P) * (1 - coef) + f(P) * coef \quad (5.5)$$

onde *f* é a função procedimental que está sendo utilizada e *coef* é um coeficiente de mesclagem, ou seja a porcentagem de contribuição da textura.

Para realizar esta composição pode-se acrescentar outra função procedimental, fazendo com que a mesclagem não ocorra de maneira uniforme, o que pode vir a ser útil para determinados tipos de padrões, como por exemplo, cristas de ondas numa superfície de mar, pedaços de mato sobre um solo arenoso, etc. Neste caso, a equação 5.5 ficará da seguinte forma:

$$T(P) = Textura(P) * (1 - coef*f2(P)) + f(P) * coef*f2(P) \tag{5.6}$$

onde *f2* é a função procedimental aplicada à mesclagem.

A equação geral para o refinamento de textura, utilizando a interpolação fractal, semelhante ao que foi apresentado na seção 4.1.1, para um dado ponto P eqüivale a:

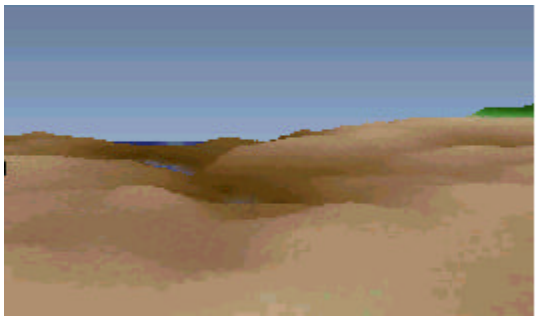
$$C = T(U_1)\cdot r(U_1)\cdot\frac{m_1}{m_1+m_2+...+m_k} + T(U_2)\cdot r(U_2)\cdot\frac{m_2}{m_1+m_2+...+m_k} + ... + T(U_k)\cdot r(U_k)\cdot\frac{m_k}{m_1+m_2+...+m_k} \tag{5.7}$$

onde $T(U_1)$, $T(U_2)$, ..., $T(U_k)$ correspondem aos resultados obtidos para o cálculo da equação de composição aplicados aos pontos válidos que cercam o ponto P.

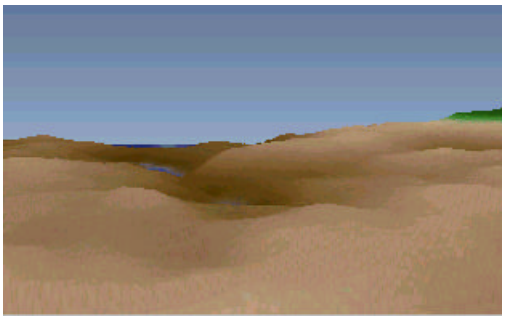
5.4.1 - Otimização

O cálculo de um refinamento para um dado ponto P pode acabar sendo muito demorado, principalmente pelas chamadas às funções $f(\mathrm{P})$, $f2(\mathrm{P})$ e $r(U_i)$.

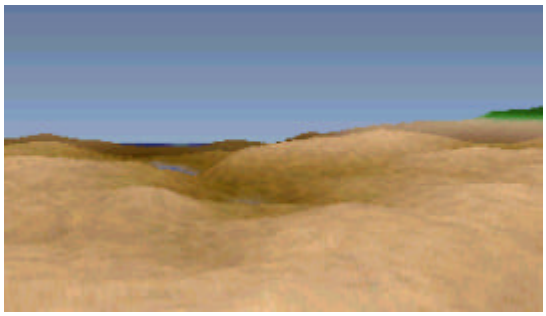
Uma maneira de se otimizar este refinamento de texturas consiste em pré-calcular valores destas funções e a seguir armazená-los numa tabela. Esta otimização pode trazer como consequência resultados com uma qualidade inferior, principalmente se a tabela for pequena, pois haverá repetição de padrões. Por outro lado trará um acréscimo alto na performance da visualização, pois para cada ponto, o cálculo será limitado a uma consulta numa tabela. Neste caso, os parâmetros de entrada da tabela devem ser as coordenadas espaciais do ponto que está sendo visualizado. Para gerar as imagens apresentadas na figura 5.3 foi utilizado este método de otimização.



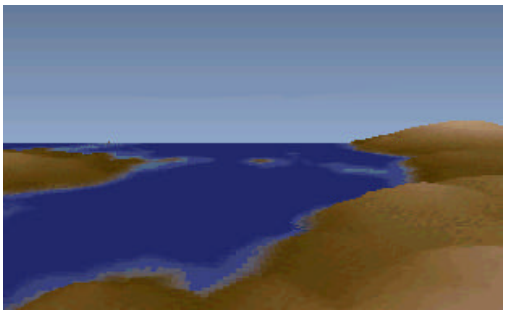
(a)



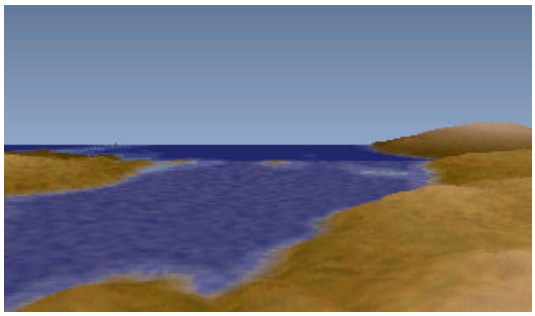
(b)



(c)



(d)



(e)

Figura 5.3 - (a) Terreno original, sem interpolação e sem refinamento de textura (b) Terreno com interpolação fractal (c) Terreno com refinamento de textura (d) Textura de lago apenas com interpolação linear (e) Textura de lago, com refinamento de textura. (figuras extraídas de [Clua et al 99b])

Capítulo 6

Conclusão e Trabalhos Futuros

Este trabalho teve como primeiro objetivo realizar um estudo sobre o estado da arte com relação à modelagem dos principais elementos da natureza. Viu-se que há várias abordagens que se podem dar ao criar estes objetos, sendo que aqui se enfatizou a modelagem através de métodos procedimentais. Apesar dos bons resultados obtidos, há ainda muito trabalho a ser desenvolvido neste campo.

As principais contribuições feitas neste trabalho, além de uma ampla revisão bibliográfica, consistem em variações com relação à implementação das funções multifractais para o processo de criação de terrenos, desenvolvimento e implementação da teoria das camadas de texturas fractais para simulação de ondas, desenvolvimento e implementação da teoria sobre as camadas tridimensionais de nuvens, apresentação da teoria sobre a interpolação fractal, apresentação da teoria sobre as texturas volumétricas e apresentação e implementação da teoria sobre o refinamento de texturas.

Um dos principais desafios está em buscar métodos procedimentais que se aproximem mais dos modelos reais. Isto permite, além de adquirir mais realismo na produção final, poder realizar simulações com mais facilidade e menos "truques".

Com relação à visualização, há também muito o que ser explorado. Para criar-se objetos através de hipertexturas, deve-se utilizar *volume-rendering*. Contudo, como foi apresentado, este método é muito demorado e exige muito espaço de memória. Assim, propôs-se uma solução que consiste nas texturas volumétricas. Este método pode ainda ser otimizado tanto no aspecto de processamento como no de estrutura de dados. Outra abordagem para evitar a presença de volumes, é o de construir superfícies geométricas juntamente com texturas a partir de um volume, uma vez que para algumas aplicações viu-se que é inviável a presença deste tipo de objeto na cena.

Os modelos de iluminação podem ainda ser estendidos, de maneira a levar em conta mais fatores atmosféricos. Apresentou-se na seção 5 uma solução que é específica para gases. Deve-se buscar modelos que sejam mais genéricos.

Para a iluminação de superfícies formadas por água, como é o caso de oceanos, deve-se desenvolver modelos que levem em conta a iluminação ambiente e os fenômenos de *caustics*, que ocorrem em meios formados por água.

Com relação à modelagem de terrenos, deve-se aperfeiçoar o método que mescla a modelagem procedimental com a modelagem baseada em mapas. O método proposto pode deformar a geografia desejada, dependendo da extensão do mapa. Além disso, há ainda muitas formações geológicas que ainda não são bem simuladas procedimentalmente, tais como terrenos com desgaste erosivo fluvial ou glacial. Deve-se também implementar a teoria proposta sobre interpolação fractal para mapas de altura.

Com relação à água, apenas estudou-se o movimento de casos mais simples, como o das ondas de um lago ou oceano. Há no entanto muito o que investigar com relação a movimentos mais complexos, como é o caso de corredeiras em rios ou quedas de água

em cachoeiras. Muitos passos foram dados neste campo através de sistemas de partículas, mas não há praticamente nenhuma abordagem procedimental. Mesmo no caso do mar, há fenômenos mais complexos como a quebra de uma onda ou o encontro de várias ondas que não estão resolvidos. Conhece-se muito pouco também sobre a iteração de objetos sólidos com a superfície da água.

Este trabalho não abarcou um estudo sobre a modelagem procedimental de plantas, já que estas possuem um padrão fractal diferente do utilizado para os elementos aqui expostos. Deve-se, portanto, estudar e construir uma série de outras funções básicas, capazes de simular com realismo estes objetos.

Referências Bibliográficas

[Aguilar 98] Carolina L. Aguiler. Técnicas Procedimentais de Texturização. *Dissertação de Mestrado*, Depto. de Informática, Puc-Rio, 1998.

[Birtel 98] Ricardo Birtel M. de Freitas. Texturas Aplicadas ao Método da Radiosidade. *Dissertação de Mestrado*, Depto. de Mecânica, Puc-Rio, 1998.

[Blinn et al 76] James F. Blinn e E. Newell. Texture and Reflection in Computer generated images. *Communications of the ACM*, 19, pp 542-546, 1976.

[Blinn 77] James F. Blinn. Models of light reflection for Computer Synthesised Pictures. *Computer Graphics (SIGGRAPH'77 Proceedings)*,11(2), 1977.

[Blinn 78] James F. Blinn. Simulation of wrinkled surfaces. *Computer Graphics (SIGGRAPH'78 Proceedings)*, volume 11, pp. 192-198, August 1978.

[Blinn 82] James F. Blinn. Light Reflection Functions for Simulation of Clouds and Dusty Surfaces. *Computer Graphics*, 16 (3), 21-29, 1982.

[Bloomenthal 97 et al] J. Bloomenthal, C. Bajaj, J. Blinn, Cani-Gascuel, M. P., A. Rockwood, B. Wyvill, G. Wyvill. Introduction to Implicit Surfaces. Morgan Kaufman Publishers, 1997.

[Bresenham 1965] J. E. Bresenham. Algorithm for Computer Control of a Digital Plotter. *IBM Systems J.*, 25-30, January 1965.

[Catmull 74] Edwin E. Catmull. A subdivision Algorithm for Computer Display of Curved Surfaces. PhD thesis, Department of Computer Science, University of Utah, December 1974.

[Cavalcante 94] Jandênia Cavalcante de Oliveira. Perturbação e Texturas em superfícies NURBS. D*issertação de Mestrado*, Depto. de Matemática, Puc-Rio, 1994.

[Cohen 94] Daniel Cohen, in Graphic Gems IV, pag. 366-369, 1994.

[Cornette 92 et al] W. M. Cornette, J. G. Shanks. Physical reasonable analytic expression for single-scattering phase function. *Applied Optics*, Vol. 31, n. 16, pp 3152-3160, 1992.

[Clua et al 98] Esteban Gonzalez Clua, Marcelo Gattass. Modelagem e visualização de ambientes naturais em volumes 3D. *Sibgrapi'98 Electronic Publication*, October 1998.

[Clua et al 99a] Esteban Gonzalez Clua, Marcelo Dreux, Marcelo Gattass. Texture Refinement using procedural functions. *Proceedings of Information Visualization*, pp. 452-456, July 1999.

[Clua et al 99b] Esteban Gonzalez Clua, Marcelo Dreux, Marcelo Gattass, Flávio Szenberg. A Terrain Visualization Algorithm. *Technical vídeo*, *Sibgrapi'99*, October 1999.

[Ebert et al 94] Davis Ebert, F. Kenton Musgrave, Darwyn Peachey, Ken Perlin, Steve Worley. Texture and Modeling: A procedural Approach. Academic Press, 1994.

[Ebert 97] David Ebert. Procedural Volumetric Modeling and Texturing, *SIGGRAPH 97*, course 14, notes, chapter 6, August 1997.

[Ebert 99] David Ebert. Procedural Volumetric Cloud Modeling and Animation. *SIGGRAPH 99, course notes*. August 1999.

[Evertsz et al 92] C. J. G. Evertsz e B. B. Mandelbrot. Multifractal Measures, Appendix B, pp 921-953. Springer-Verlag, New York, 1992.

[Faria 96] José L. Faria Júnior. Técnicas para Geração de Texturas em um Sistema Ray-Tracing. D*issertação de Mestrado*, Depto. de Mecânica, Puc-Rio, 1996.

[Foley et al 96] James D. Foley, Andries van Dam, Steven K. Feiner, John F. Huges. Computer Graphics - principles and pratice. Second Edition in C, Addison-Wesley Publishing Company, 1996.

[Foster et al 99] Nick Foster, Dimitri Metaxas. Realistic Animation of Liquids. *SIGGRAPH 99, course notes*. August 1999.

[Fourneir et al 86] Alain Fournier and William T. Reeves. A simple Model of Ocean Waves. *Computer Graphics*, vol. 20, n. 4, pp 75-84. 1986.

[Fournier 94] Alain Fournier. The Modelling of Natural Phenomena. *SIGGRAPH 94, Course Notes.* August 1994.

[Fujimoto et al 86] A. Fujimoto, K. Iwata. Accelerated Ray-Tracing. *IEEE CG e A.* v. 6, pp. 16-26, April 1986.

[Gardner 84] Geoffrey Y. Gardner. Simulation of natural scenes using textured quadric surfaces. In Hank Christiansen, editor, *Computer Graphics* (*SIGGRAPH'84 Proceedings)*, volume 18, pp 11-20, July 1984.

[Gardner 85] Geoffrey Y. Gardner. Visual Simulation of clouds. In B. A. Barsky, editor, *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19, pp. 297-303, July 1985.

[Gardner 94] Geoffrey Y. Gardner. Modelling Amorphous Natural Features. *SIGGRAPH'94 Course notes*, August 1994.

[Glassner 89] A. Glassner. An Introduction to Ray Tracing. Academic Press, 1989.

[Kajiya et al 84] J. T. Kajiya and T. L. Kay. Ray Tracing Volume Densities, *Computer Graphics (SIGGRAPH '84 Proceedings),* July 1984.

[Kruger 91] W. Kruger. The Applicaions of Transport Theory to Visualization of 3D Scalar Data Fields. *Computer in Physics*, pp. 397-406, July/August 1991.

[Mandelbrot 82] B. Mandelbrot, The Fractal Geometry of Nature, W. H. Freeman and Co, 1982.

[Miyata 86] H. Miyata. Finite-Difference Simulation of Breaking Waves. *Journal of Computational Physics* 65 pp. 179-214, 1986.

[Musgrave et al 89] F. Kenton Musgrave, Craig E. Kolb and Robert S. Mace. The synthesis and rendering of eroded fractal terrains. *Computer Graphics (SIGGRAPH '89 Proceedings)*, volume 23, pp. 41-50, July 1989.

[Musgrave 99] F. Kenton Musgrave. Procedural Fractal Terrain. *SIGGRAPH '99 Course Notes*. August 1999.

[Neyret 97] Neyret, F., Qualitative Simulation of Convenctive Cloud Formation and Evolution. *Eight International Workshop on Computer Animation and Simulation, Eurographics*, September 1997.

[Nishita et al 96] Tomoyuki Nishita, Yoshinori Dobashi, Eihachiro Nakamae. Display of Clouds Taking into Account Multiple Ansiotropic Scattering and Sky Light., *Computer Graphics (SIGGRAPH '96 Proceedings)*, volume 30, pp 379-386, August 1996.

[Peachey 85] D. Peachey, Solid texturing of complex surfaces .In B. A. Barsky, editor, *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19, pp. 279-286, July (1985)

[Peitgen et al 86] H. O. Peitgen and P. H. Richter, *The Beauty of Fractals*, Springer-Verlag, 1986.

[Perlin 85] Ken Perlin. An image synthesizer. In B. A. Barsky, editor, *Computer Graphics (SIGGRAPH ’85 Proceedings)*, volume 19, pp. 287-296, July 1985.

[Perlin et al 89] Ken Perlin and Eric M. Hoffert. Hypertexture. In Jeffrey Lane, editor, *Computer Graphics (SIGGRAPH’89 Proceedings)*, volume 23, pp. 253-262, 1989.

[Phong 75] Phong Bui-Thong. Illumination for Computer-generated Pictures. *Comm. ACM*, 18(6), June 1975.

[Prusiniewicz et al 90] Przemyslaw Prusiniewicz and Aristid Lindenmayer, The Algorithmic Beauty of Plants, Springer-Verlag, 1990.

[Sobierajski et al 94] L. M. Sobierajski and A. E. Kaufman. Volumetric Ray Tracing. Proceedings of the Symposium on Volume Visualization, pp.11-18, 1994.

[Stevens 74] P. S. Stevens, Patterns in Nature, Little Brown and Co, 1974.

[Szenberg 97] Flávio Szenberg. Um Algoritmo para Vizualização de Terrenos com Objetos.D*issertação de Mestrado*, Depto. de Informática, Puc-Rio, 1997.

[Tessendorf 99] Jerry Tessendorf. Simulating Ocean Water. *SIGGRAPH ’99 Course Notes*. August 1999.

[Thompson 61] D. W. Thompson, On Growth and Form, Cambridge University Press, 1961.

[Wyvill et al 86] G. Wyvill, C. McPheeters, B. Wyvill. Data Strucuture for Soft Objects, *The Visual Computer*, vol. 2, pp 227-234, January 1986.

[Worley 96] Steve Worley. A Cellular Texture Basis Function, *Computer Graphics (SIGGRAPH ’96 Proceedings)*, volume 30, pp 291-294, August 1996.