Modelagem e Visualização de Ambientes Naturais em Volumes 3D

ESTEBAN WALTER GONZALEZ CLUA[1]
MARCELO GATTASS[1]

[1]PUC-RIO - Departamento de Informática
Rua Marquês de São Vicente, 255, 22443-900, Rio de Janeiro, RJ, Brasil
{esteban,gattass}@inf.puc-rio.br

Resumo: Existe ainda muito trabalho a ser desenvolvido sobre modelagem de ambientes naturais, assim como em simulação de fenômenos da natureza. Neste artigo exploramos a possibilidade de modelar alguns elementos naturais de forma volumétrica. Apresentamos também uma técnica de visualização adequada para este tipo de modelagem. Por fim, discutiremos brevemente sobre a possibilidade de realizar simulações de fenômenos naturais sobre estes modelos gerados.

Palavras-chave: *modelagem procedural, hipertexturas, volume rendering.*

1 Introdução

Encontramos na literatura inúmeras técnicas para modelar elementos da natureza. Alguns destes elementos , tais como terrenos e vegetação, podem receber um tratamento geométrico eficiente.

Existem, no entanto, alguns objetos cuja representação geométrica se torna inviável, como nuvens, gases e fogo. Procuramos, assim, uma técnica que nos possibilite modelar todos estes elementos numa única cena, de forma a criar modelos e imagens realistas, permitindo ao mesmo tempo a simulação de fenômenos naturais em que todos os elementos possam interagir entre si.

Inicialmente, poderíamos modelar toda a cena sobre um único volume. Contudo, isto seria inviável se estivermos trabalhando com cenas muito grandes, pois os volumes poderiam vir a adquirir proporções enormes e poderiam vir a requerer um alto tempo de processamento e uma grande quantidade de memória. Desta forma, exploramos a possibilidade de uma modelagem mista: alguns objetos são geométricos e outros volumétricos. Assim sendo, numa mesma cena, poderemos ter vários volumes delimitados por cubos e vários objetos geométricos. Neste trabalho iremos inicialmente propor técnicas para modelar relevos de forma volumétrica. A seguir apresentaremos a modelagem volumétrica de nuvens, separando-a em dois casos: camada de nuvens e nuvens 3D. Finalmente iremos discutir sobre a visualização para estas modelagens.

2 Modelagem Volumétrica

O processo de modelagem volumétrica consiste basicamente em preencher o volume inteiro com diversos atributos para cada região.

Neste trabalho estamos utilizando um volume regular. Para cada *voxel* do volume gostaríamos de definir uma estrutura da forma:

$$V_{ijk}=(t,\delta,m)_{ijk}$$

onde t corresponde ao tipo do elemento que está sendo modelado (solo, água, etc.), δ é a densidade correspondente ao *voxel* e m indica as propriedades do material deste elemento.

2.1 Terrenos

No caso de terrenos, a modelagem pode ser feita de forma procedural ou por mapas de altura. A modelagem procedural tem a vantagem de permitir o refinamento de detalhes, controle paramétrico e facilidade em animar atributos, porém possui a desvantagem de que gera terrenos aleatórios. Já o método por mapa de alturas possibilita a construção de terrenos especificados. Este artigo sugere um método misto entre ambos, combinando as vantagens que cada um oferece. Esta técnica pode ser utilizado para uma modelagem geométrica ou volumétrica.

A modelagem de terrenos por mapas de altura consiste apenas em utilizar os *pixels* uma imagem como o valor da altura para um terreno. Contudo, este método apresenta sérios problemas de *aliasing* quando o observador se aproxima do terreno. Isto ocorre porque um ponto do mapa de altura está sendo mapeado para muitos *pixels* na tela. Uma modelagem mista entre mapas de altura e funções procedurais pode ajudar a resolver este problema. No método proposto utiliza-se o gradiente de um ponto do mapa de altura h como um coeficiente para uma função procedural. Este gradiente pode ser facilmente estimado diretamente sobre o mapa de altura da seguinte forma:

$$\nabla h = D_u \otimes D_v$$

onde:

$D_u = (2, D_x, 0)$ e $D_v = (0, D_y, 2)$

e, caso h seja discreto (por exemplo, uma imagem digital de terreno ou de um mapa de isolinhas de altura constante),

$$D_x = \frac{\partial h}{\partial x} = h_{i+1,j} - h_{i-1,j}$$

$$D_y = \frac{\partial h}{\partial y} = h_{i,j+1} - h_{i,j-1}$$

Assim, o produto vetorial D_u x D_v corresponderá ao gradiente para a coordenada (u, v) do mapa de altura. A nossa proposta é que este valor seja acumulado numa variável que dá um coeficiente de escala para a função multifractal geradora de terrenos aleatórios sugerida em [Musgrave 1989]. Esta função consiste numa iteração fractal utilizando a função *noise* [Perlin 1985] sobre cada ponto do plano que irá formar o terreno:

```
altura = 0
incremento = 1
para i = 1 até f faça
    incremento = ((noise (ponto) + ∇h) x
        Fator_Limitante[f])x incremento
    incremento = incremento x altura
    altura = altura + incremento
    ponto = ponto x espaçamento
retorne (altura)
```

onde f é o número de iterações fractais, Fator_Limitante é uma tabela com valores entre 0 e 1, sendo Fator_Limitante[i] < Fatorl_Limitante[j], quando i < j, e espaçamento corresponde a um deslocamento sucessivo para o ponto.

2.2 Nuvens

A modelagem de nuvens ainda é um problema que permanece em aberto, embora alguns trabalhos interessantes tenham sido desenvolvidos, tais como [Gardner 1985, Ebert 1997]. Estas limitações devem-se principalmente ao fato de que as técnicas utilizadas até hoje procuram aproximar uma nuvem a um modelo geométrico.

Para modelar as nuvens sugerimos dois métodos diferentes, ambos utilizando hipertexturas [Perlin and Hoffert 1989]: um para criar camadas atmosféricas de nuvens e outro para criar uma nuvem 3D isolada.

O primeiro método utiliza a função *fractal Brownian motion*, fBm, descrita inicialmente por [Musgrave 1994] para sintetizar imagens 2D de nuvens. Apesar de Musgrave não ter utilizado esta função para criar nuvens 3D, ele sugere que para tal se utilizem hipertexturas junto com esta função. A fBm é uma iteração fractal utilizando a função *noise* sobre um ponto do espaço, sendo cada iteração delimitada por uma tabela de fatores limitantes, tal como descrita na função multifractal.

O método para definir uma camada de nuvens começa determinando uma região *fuzzy*, que é, no volume da cena, um retângulo alinhado com os eixos e definido pelos 2 vértices:

v1 = (0, Altura Inicial das nuvens, 0)
v2 = (largura, Altura Final das nuvens, profundidade)

Todas as nuvens estão dentro deste retângulo, que é dividido em três fatias horizontais. Chamamos a fatia inferior de zona nebulosa inferior, a fatia do meio de zona nebulosa central e a fatia superior de zona nebulosa superior.

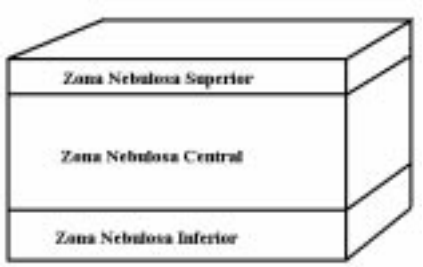

Figura 1 - Camadas de Regiões *fuzzy* para as camadas atmosféricas de nuvens.

Normalmente, o processo de geração de nuvens deve ser o último passo da criação de um cena, como geralmente acontece quando se trabalha com hipertexturas, 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 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. Sugerimos dar o seguinte tratamento para a densidade δ *de* cada *voxel* desta região:

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

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 nos afastarmos 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 nos afastamos da zona central. Para corrigir este problema podemos acrescentar à equação um valor aleatório. Adequa-se bem a esta situação uma chamada à função *noise*. Assim, a equação resultará em:

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

Em geral, a parte inferior das camadas de nuvens tende a ser mais achatada que a parte superior. 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.

O segundo método, próprio para modelar uma nuvem 3D, é uma ligeira modificação do algoritmo descrito por [Ebert 1997] e consiste na utilização das funções *noise* e *turblence* [Perlin 1985] e de funções geradoras de superfícies implicitas.

O método consiste primeiramente na definição de uma região 3D, que envolve toda a nuvem. Note-se que se estivermos modelando apenas a nuvem de forma volumetrica, ou seja os demais objetos estão modelados geometricamente, esta região será um volume particular para a nuvem e pode ser definido na cena como um cubo. O algoritmo prossegue inserindo aleatoriamente alguns pontos no interior do volume. Estes são o centro de superfícies implícitas, que por sua vez estão definidas por:

$$F_i(r_i) = \begin{cases} 1 - 3\frac{r_i^2}{b_i^2} & 0 \le r_i \le \frac{b_i}{3} \\ \frac{3}{2}(1 - \frac{r_i}{b_i})^2 & \frac{b_i}{3} \le r_i \le b_i \\ 0 & b_i \le r_i \end{cases}$$

onde r_i é a distância de um ponto ao centro da i-ésima superfície implícita e b_i é o campo de atração da i-ésima superfície implícita.

A seguir, cada *voxel* V_{ijk} do interior da região definida deve ser perturbado no espaço da seguinte forma:

$$P_t(V_i) = V_i + Noise\ (V_i) + Turbulence\ (V_i)$$

Feito isto, calcularemos 2 componentes de densidade δ_1 *e* δ_2, aplicando as seguintes funções:

$$\delta_1(V_{ijk}) = \sum_{j=1}^{IS} F_j(r_t)$$

onde *IS* é o número de superfícies implicitas e $F_i(r_t)$ é a i-ésima função implícita, aplicada sobre a distância r_t do ponto perturbado, P_t, ao centro da i-ésima superfície implícita. Por sua vez, δ_2 é definido por:

$$\delta_2 = Turbulence(V_{ijk})$$

Por fim, a densidade D_i a ser dada como retorno para o *voxel* V_i será dada por:

$$\delta = ((B \text{ x } \delta_1 + (1\text{-}B) \text{ x } \delta_2) \text{x } M_\delta)^k$$

onde $B \in [0,1]$ e corresponde a um parâmetro de agrupamento para as duas densidades criadas, M_δ corresponde à máxima densidade que a nuvem pode ter e *k* é um parâmetro que permitirá ressaltar mais ou menos os contornos da nuvem.

3 Visualização

A utilização do *Ray-casting* tradicional não é suficiente para este trabalho, pois podemos ter modelos geométricos na cena, além dos volumes. O *Ray Tracing* também não se adapta por completo, devido a presença dos volumes.

Propomos assim uma técnica híbrida entre o *Ray Tracing* e o *Ray Casting* primeiramente descrita por [Sobierajski and Kaufman 94].

Esta técnica consiste em lançar raios partindo de um observador em direção a um *pixel* de um *grid* definido. Inicialmente trataremos cada um dos volumes gerados como cubos inseridas no espaço, independente do elemento que estes representam. Podem existir na cena outros objetos geométricos, tais como esferas, planos, etc. O algoritmo começa funcionando de forma similar a um *Ray Tracing* tradicional. Quando for encontrada uma interseção com um cubo que representa um volume, deve-se calcular e acumular o ponto de entrada e saída do raio sobre este objeto. A partir deste ponto, o algoritmo passa a comportar-se como um *Ray Casting* e o raio passa a percorrer pelo interior do volume de forma discreta, ou seja, de *voxel* para *voxel*. Ao se percorrer cada um dos *voxels*, acumula-se uma cor para este raio, assim como o coeficiente de transparência, com base na estrutura do *voxel* em questão. Fazemos isto até atingir o segundo ponto de interseção da caixa, que corresponde à saída do raio. 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 segue o comportamento normal de um *Ray Tracing*.

A equação clássica da iluminação utilizada no *Ray Tracing* não se adapta perfeitamente a este algoritmo, por isto adotamos a equação descrita por [Sobierajski

and Kaufman 94], que é apenas uma extensão da primeira:

$$I_\lambda(x,\vec{w}) = I_{v\lambda}(x,x',\vec{w}) + \tau_\lambda(x,x')I_{s\lambda}(x',\vec{w})$$

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

$$I_{s\lambda}(x',\vec{w}) = I_{a\lambda}k_{d\lambda} + \sum_{i=1}^{nL} A_i I_{p\lambda i}[k_{d\lambda}(\vec{N}\cdot\vec{L}_i) + k_{s\lambda}(\vec{N}\cdot\vec{H}_i)^n] + k_{s\lambda}I_{s\lambda}(x',\vec{w}_r) + k_{t\lambda}I_{s\lambda}(x',\vec{w}_t)$$

onde x' é o primeiro ponto de interseção encontrado pelo raio $\vec{w}$, *nL* é o número de fontes de luz da cena, $I_{p\lambda i}$ é a intensidade da i-ésima fonte de luz, $k_{d\lambda}$, $k_{s\lambda}$ e $k_{t\lambda}$ 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, $I_{s\lambda}(x',\vec{w}_r)$ e $I_{s\lambda}(x',\vec{w}_t)$ são as contribuições sobre x' da luz provinda da direção do reflexo especular e da transmissão, respectivamente.

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

$$\tau_\lambda(x,x') = e^{-\int_x^{x'}\sum_{i=1}^{nV}\sigma_{a\lambda}(i,t)+\sigma_{sc\lambda}(i,t)dt}$$

onde *nV* é o número de volumes na cena e $\sigma_{a\lambda}(i,t)$ e $\sigma_{sc\lambda}(i,t)$ correspondem à quantidade de luz absorvida e espalhada, respectivamente, para um volume *i* na posição *t*.

A componente $I_{v\lambda}(x,x',\vec{w})$ corresponde à componente volumétrica de iluminação e baseia-se no modelo da teoria do transporte da propagação da luz citada em [Kruger 1991]:

$$I_{v\lambda}(x,x',\vec{w}) = \int_x^{x'}\sum_{i=1}^{nV}[I_{E\lambda}(i,t) + \sigma_{sc\lambda}(i,t)\int F_{sc\lambda}(\vec{w}',\vec{w})I(t,\vec{w}')d\vec{w}']\tau_\lambda(x,t)dt$$

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

4 Conclusão e Trabalhos Futuros

As idéias apresentadas aqui estão sendo implementadas em um programa de modelagem e visualização, *World Generator*. Até a data do Simpósio, esperamos apresentar os primeiros resultados desta investigação.

Uma vez que se tem este sistema bem implementado, apontamos a possibilidade de se realizar algumas simulações de fenômenos naturais com bastante precisão. Considere por exemplo o fenômeno do encontro de duas massas de ar: cada massa pode ser considerado como um volume separado, formado por *voxels* que representariam partículas com determinadas propriedades. Quando duas destas partículas com propriedades diferentes ocuparem o mesmo espaço físico, ou seja, ocorreu um encontro das massas de ar, algum evento pode vir a ser associado (por exemplo, o surgimento de uma nova partícula, representando a água da chuva).

5 Bibliografia

[Musgrave 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.

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

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

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

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

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

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