Geração de terrenos com Diamond Square

Gustavo Ebbo* Ivo Paiva† Rodrigo Calábria‡

Universidade Federal Rural do Rio de Janeiro,
Departamento de Ciência da Computação, Brasil

Figura 1: Terreno gerado a partir do Diamond Square

RESUMO

Este artigo propõe uma implementação de geração de terrenos, utilizando o algoritmo *Diamond-Square*. A codificação faz uso das bibliotecas *OpenGL* e *GLUT*, amplamente utilizadas no desenvolvimento de aplicações gráficas, uma vez que trazem consigo inúmeras funções que facilitam o desenvolvimento de aplicações para computação gráfica.

Keywords: Radiosity, global illumination, constant time.

1 INTRODUÇÃO

Com o advento do avanço tecnológico, diversos algoritmos de computação gráfica têm ganhado cada vez mais espaço no dia a dia das pessoas, seja facilitando a verificação de tumores por meio da análise de imagens médicas, seja no aprendizado de comportamento de transito para carros autônomos.

Entre as inúmeras aplicações possíveis para a área, está o processo de geração de terrenos, que consiste no emprego de métodos de computação gráfica com o objetivo de formar um modelo tridimensional que se assemelhe a um terreno geográfico, utilizando para isso técnicas que mapeiam as diferenças de níveis do solo, que permitem a correta representação de montanhas, assim como erosões. Devido à alta popularidade dos jogos e da realidade virtual, foram desenvolvidos novos algoritmos de geração de terreno, aliando o alto desempenho necessário na indústria de jogos com alta qualidade gráfica.

2 CRIAÇÃO DE TERRENO A PARTIR DE HEIGHTMAPS

Para que seja criado um modelo tridimensional de um terreno, naturalmente é utilizado como base uma malha plana, normalmente formada por faixas de triângulos enfileiradas, onde é aplicado um *heightmap* (mapa de altura), que mapeia cada vértice dos triângulos definidos na malha a um valor referente a altura deste vértice no espaço tridimensional. Na área da computação gráfica, é comum o uso de imagens monocromáticas que representam um mapa de altura serem aplicadas para a distorção da malha inicial. Considerando que imagens, por definição, são matrizes bidimensionais de pixels, tais imagens podem ser traduzidas para matrizes durante a execução de um programa, onde para cada elemento da matriz seria atribuído um valor relativo ao tom de cinza deste pixel. Há também a possibilidade de se utilizar algoritmos para a geração de matrizes que se comportem como um *heightmap*, mas sem a utilização de imagens no processo, já que a matriz de alturas seria determinada diretamente. Há diversos algoritmos bem conhecidos já definidos na literatura que geram *heightmap* que conseguem representar mapas de relevos de forma realista. Alguns exemplos destes algoritmos são o *Perlin Noise*[3], *Fault Formation* e o *Square-Diamond*[1], utilizado neste projeto.

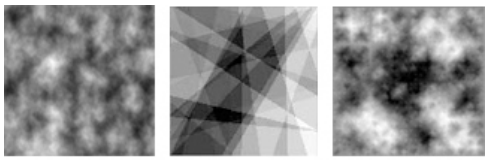

Figura 2: *Heightmaps* formados por diferentes algoritmos.

Há também métodos que podem ser aplicados a *heightmap* para simular efeitos naturais que superfícies geológicas sofrem, como efeitos de erosão causados pela chuva e pelo ar. Os algoritmos geradores de *heightmap*, na sua maioria, utilizam atribuições de valores randômicos e médias entre valores e posições da matriz base para formar as variações de alturas em seus elementos.

3 ALGORITMO DIAMOND-SQUARE

O algoritmo *Diamond Squares* é aplicado neste trabalho para criar o mapa de alturas do terreno. O algoritmo foi proposto por Fournier em 1982 [1]. O algoritmo começa em plano de duas dimensões e a

*e-mail: gustavoebbo@ufrrj.br
†e-mail: ivosantospaiva@ufrrj.br
‡e-mail: rodrigovcalabria@ufrrj.br

partir de um conjunto de passos chamados de passos de diamante e passos quadrados, preenche todo o plano de quadrados, gerando o terreno. O funcionamento do algoritmo é descrito abaixo [2]:

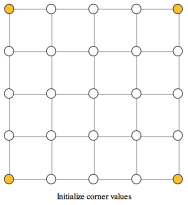

Figura 3: Primeira iteração

No primeiro passo, o algoritmo insere 4 deformações nas bordas do plano com valores aleatórios que variam dentro do limite de altura estabelecido para o problema. Após inserir as deformações, o algoritmo realiza um passo diamante, que consiste em inserir um vértice no meio do plano e sua altura é baseada na seguinte fórmula.

$$\frac{H1+H2+H3+H4}{4}+\Delta \quad (1)$$

Onde H_i representa a altura dos vértices ao redor do diamante e Δ representa o valor de ruído que deve ser somado à média aritmética das alturas.

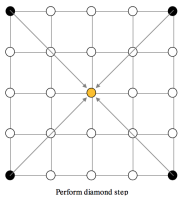

Figura 4: Passo diamante

Após a inserção do diamante central, o algoritmo realiza o passo quadrado, gerando deformações no ponto médio entre as bordas anteriores, gerando novos quadrantes.

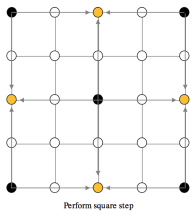

Figura 5: Passo quadrado

O algoritmo se repete até que o critério de parada seja satisfeito, seguindo o pseudocódigo listado abaixo.

3.1 Implementação do *Diamond-Square*

No nosso projeto o algoritmo do *Diamond-Square* foi alterado de forma que sua implementação utilize recursão ao invés de apenas o percorrimento de *loops* para realizar suas iterações. O algoritmo já apresenta uma natureza recursiva, pois define um ponto central para atribuir seu valor, de forma a dividir o plano bidimensional em quatro quadrantes (no topo à esquerda e a direita e na base à direita e esquerda do ponto central), e então repete o passo para cada quadrante criado (definindo um ponto central e o dividindo em mais

Algorithm 1 Diamond Square

```
function DIAMOND_SQUARE(Width, Height)
    A ← ArrayofWidth * Height
    Pre seed four corners of A with a Random value
    StepSize ← Width − 1
    r ← Rand(Range)
    while doStepSize > 1
        for Each point in A do
            Do DiamondStep for each square in A
        end for
        for Each point in A do
            Do SquareStep for each square in A
        end for
        StepSize ← StepSize/2
        Reduce Random range for r
    end while
end function
function DIAMONDSTEP(x,y,StepSize,r)
    avg ← average of square corners StepSize apart
    A[x + stepSize/2][y + stepSize/2] = avg + r
end function
function SQUARESTEP(x,y,StepSize,r)
    avg ← average of four corners of diamond
    A[x][y] = avg + r
end function
```

Output: HeightMap

quatro quadrantes, e assim por diante). Com isso, a mudança principal na implementação deste algoritmo de forma recursiva consiste na necessidade de mais duas entradas em sua função principal, que representam os pontos limitadores do quadrante em que será realizado o passo atual da recursão.

4 Outros processos da implementação

Além da definição das alturas dos vértices através do algoritmo *Diamond-Square*, outros processos são realizados na *mesh* para que sua remodelagem resulte em soluções mais agraveis e realistas. Após a aplicação de relevo à *mesh*, teremos apenas as informações de posição dos seus vértices, sem nenhuma informação de cor que possa ser renderizada e exibida. Para a coloração da *mesh*, a cada vértice criado é atribuído uma cor do sistema RGB respeitando as seguintes regras:

- Para vértices com altura de até 30% de ALT, aplica-se a cor amarela.
- Para os entre 30% e 50% de ALT, aplica-se a cor verde.
- Para os entre 50% e 70% de ALT, aplica-se a cor marrom.
- Para os acima de 70% de ALT, aplica-se a cor branca.

Onde ALT representa a diferença de altura do vértice mais alto ao vértice mais baixo da *mesh*. A escolha das cores teve como objetivo representar componentes de um terreno real (no caso areia, grama, rochedos e neve, respectivamente). Outro processo realizado após a definição das alturas dos vértices é o de cálculo dos vetores normais de cada um destes vértices. A forma em que é calculado estes vetores é definida depois da escolha do tipo de *shading* que será atribuído ao modelo. Um *shader* é responsável pela determinação dos níveis apropriados de luz, sombra e cor em cada ponto do modelo, de forma a induzir uma noção de profundidade ao observador. O *Flat Shading* e o *Smooth Shading*(ou *Gouraud Shading*), mostrados na Figura 6, são exemplos de tais.

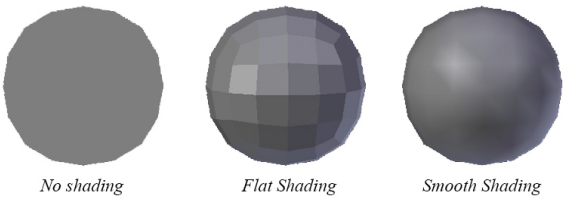

Figura 6: Exemplos de *shading*

5 SMOOTH SHADING

Como é possível perceber pela Figura 6, a utilização do *Smooth Shading* como método de sombreamento apresenta um resultado muito mais natural se comparado ao *Flat Shading*, por apresentar um gradiente de cores em suas faces. Isto se dá pela forma em que as normais dos vértices são calculadas. O *Flat Shading* apresenta um mesmo vetor normal para cara vértice de uma dada face, enquanto o *Smooth Shading* é criado modificando os vetores normais de cada vértice compartilhado por múltiplas faces, realizando a soma normalizada das normais destas faces e aplicando ao vértice.

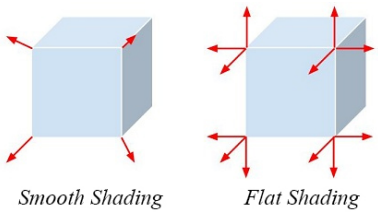

Figura 7: Exemplos de orientações de normais

Na utilização do *Smooth Shading*, com a realização de seus cálculos, toda face apresenta uma variação de sombreamento que é continuada por suas faces vizinhas, de forma a não acentuar as arestas que as separam.

5.1 Cálculo das normais

No *Smooth Shading*, é realizado para cada vértice a soma dos vetores normais das faces que as interceptam, e então realizado a normalização deste vetor resultante, provindo um vetor normal para este vértice.

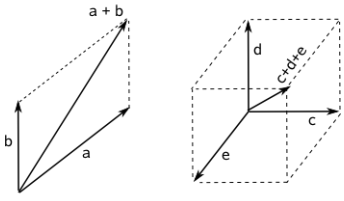

Figura 8: Exemplo de soma de vetores

Sabendo disso, é possível perceber que o *Smooth Shading* utiliza os vetores que definem as normais do *Flat Shading* em seus cálculos.

6 RESULTADOS OBTIDOS

Para obter o seguinte resultados, executamos o código com as seguintes configurações: Tamanho do plano $100x100$ e limites de altura $[-100, 100]$.

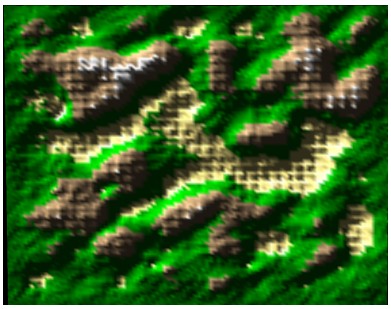

Figura 9: Terreno gerado visto de cima

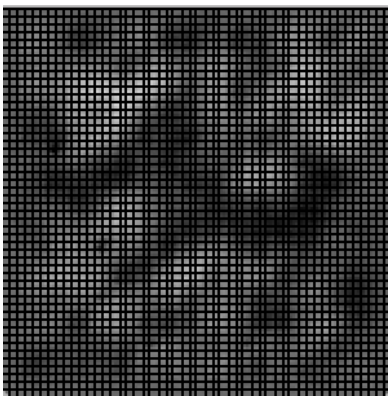

Figura 10: mapa de altura

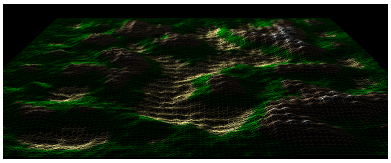

Figura 11: Malha de poligonos

Com a implementação dos algoritmos desenvolvidos e com ajustes de parâmetros para o algoritmo *Diamond-Square*, foi possível obter resultados satisfatórios de geração de terrenos. Algumas amostras de terrenos gerados são mostradas a seguir.

7 CONCLUSÃO E TRABALHOS FUTUROS

O trabalho apresentou resultados satisfatórios. O algoritmo de *Diamond Squares* gerou bons mapas de altura como o esperado, no entanto foram obtidos resultados com falhas. De acordo com Gavin[4], o algoritmo gera resultados falhos, por conta dos grandes vincos verticais e horizontais introduzidos por causa da grande perturbação na grade retangular. Como trabalho futuro, serão implementados outros modelos de *Shaders* para reduzir o aspecto pontiagudo do terreno, assim como a otimização dos métodos implementados durante o trabalho com o uso de bibliotecas paralelas. Outros algoritmos geradores de mapa de altura também serão implementados visando um comparativo entre a qualidade e defeito dos principais algoritmos de geração de terreno.

REFERÊNCIAS

[1] A. Fournier. An image synthesizer. *SIGGRAPH Comput. Graph.*, (19):287–296, 1985.

[2] LightHouse3d, Inc. *Terrain Tutorial*, January 2002.
[3] K. Perlin. Computer rendering of stochastic models. *ACM SIGGRAPH Computer Graphics*, 20(4):39–48, 1982.
[4] K. Perlin. The definition and rendering of terrain maps. *Communications of the ACM*, 6(25):371–384, 1986.