Um Escalonador para Geração Procedural de Terrenos Pseudo-Infinitos em Tempo-Real Utilizando Arquiteturas Heterogêneas GPU/CPU

Fábio Markus N. Miranda, Luiz Chaimowicz
Departamento de Ciência da Computação
UFMG
Email: fabiom@gmail.com, chaimo@dcc.ufmg.br

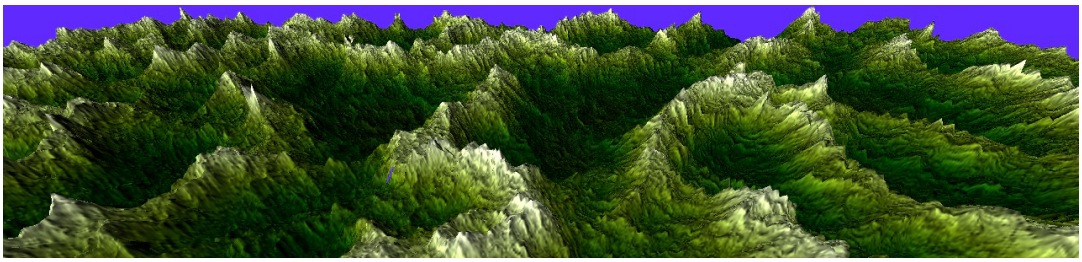

Figura 1. Terreno gerado proceduralmente.

***Resumo*—O rápido crescimento do poder de processamento das placas gráficas fez com que diversas tarefas migrassem da CPU para a GPU. Porém, as unidades de processamento gráfico podem ser vistas como aliadas da CPU, e não rivais. Este trabalho propõe um modelo que utilize tanto a CPU quanto a GPU para minimizar o tempo gasto com a geração procedural de terrenos e permitir uma navegação fluida através de um mundo pseudo-infinito gerado proceduralmente.**

Ao final, uma discussão é feita com base em testes com três modelos de geração (apenas CPU, apenas GPU, GPU e CPU), com o objetivo de expor suas vantagens e desvantagens.

***Keywords*-geração procedural; gpu; gpgpu; programação paralela**

I. Introdução

A geração procedural de modelos é uma área da Ciência da Computação que propõe que modelos gráficos tridimensionais (representação em polígonos de algum objeto) possam ser gerados através de rotinas e algoritmos. Tal técnica vem se tornando bastante popular nos últimos tempos, tendo em vista que, com o crescimento da indústria do entretenimento, há uma necessidade de se construir modelos cada vez maiores e com um grande nível de detalhe. A técnica de geração procedural vem então como uma alternativa à utilização do trabalho de artistas e modeladores na criação de modelos tridimensionais.

A geração de terrenos é uma das vertentes da geração procedural. Diversos algoritmos [1] foram desenvolvidos com o objetivo de gerar terrenos cada vez mais realistas Terrenos gerados proceduralmente são uma fatia

Outro fato também muito relevante atualmente são as GPUs, microprocessadores incorporados às placas de vídeo e especializados em processamento gráfico. O avanço da indústria de *games* fez com que as GPUs ficassem cada vez mais rápidas, tornando-as atraentes para outras áreas da computação.

O restante deste trabalho está organizado da seguinte maneira: na Seção II são apresentados os trabalhos relacionados. A Seção III mostra as principais contribuições deste trabalho. A Seção IV revisa alguns conceitos pertinentes. A Seção V apresenta a arquitetura proposta para a geração procedural utilizando tanto a CPU quanto a GPU e a Seção VI detalha a implementação de tal arquitetura. A Seção VII faz uma discussão sobre os testes realizados. Finalmente, a Seção VIII apresenta a conclusão e a perspectiva para futuros trabalhos.

II. Trabalhos relacionados

A base para a geração procedural de terrenos é o ruído Perlin [2], uma função pseudo-aleatória que, dado uma entrada (posição), retorna um valor que possui uma suave transição com os seus vizinhos. Em [3] foi apresentado um ruído Perlin otimizado, que buscou tornar o ruído mais amigável às novas arquiteturas (GPUs), melhorar as propriedades visuais e introduzir uma única versão do ruído que retornaria os mesmos valores independentemente da plataforma de *hardware* ou *software*.

Em [1] são apresentados alguns algoritmos que fazem uso do ruído Perlin e que são capazes de gerar terrenos de uma forma significativamente realista. Podemos citar o algoritmo *fBm*, *heterogenous terrain*, *hybrid multifractal* e *ridged multifractal*, sendo que este último foi o algoritmo utilizado neste trabalho.

A geração procedural utilizando a GPU foi explorada em [4] e [5]. O primeiro trabalho, faz uso de *geometry shaders* e está limitado às placas de vídeo com suporte a DirectX 10. O segundo trabalho, mais abrangente quanto as placas de vídeo suportadas, gera os terrenos na GPU com o uso de algoritmos multifractais (semelhante ao que é proposto aqui). Nenhum dos dois trabalhos, porém, faz uma comparação entre implementações de geração de terrenos utilizando a CPU e a GPU, e também não buscam uma plataforma que utilize as duas arquiteturas.

A geração procedural utilizando uma arquitetura

III. Contribuições

O trabalho apresentado aqui busca atingir os seguintes objetivos:

- Propor uma plataforma de geração procedural de terrenos que faça uso tanto da CPU quanto da GPU, de forma a não deixar
- Fazer um estudo quanto aos benefícios da geração procedural na CPU, na GPU, e em uma arquitetura heterogênea GPU/CPU.

IV. Conceitos básicos

A. Ruído Perlin

O ruído Perlin foi criado pelo Professor Ken Perlin [2], da *New York University* e é usado para simular estruturas naturais, como nuvens, texturas de árvores, e terrenos.

A função ruído retorna, para um dado domínio e as mesmas sementes (*seeds*), números entre 0 e 1; dessa forma, em uma segunda execução, com as mesmas entradas, teremos os mesmos números entre 0 e 1. Cada valor retornado é o resultado do seguinte produto interno:

$$G \cdot (P-Q)$$

Onde P é a posição do ponto que está sendo calculado o valor do ruído, Q é a posição de um de seus vizinhos, e G é o valor de um vetor gradiente pseudo-aleatório. Os resultados do produto interno dos vizinhos é então interpolado, garantindo assim que haverá uma suave transição entre todos os valores retornados.

O resultado, como pode ser visto na Figura 2, apresenta transições suaves, diferentemente do ruído aleatório.

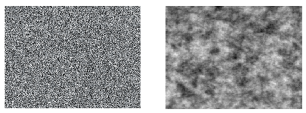

Figura 2. *Esquerda:* Ruído aleatório. *Direita:* Ruído Perlin.

As características fundamentais do ruído Perlin são então a sua aparente aleatoriedade (ao menos para o olho humano); sua capacidade de ser reproduzido, dado os mesmos valores dos gradientes; e sua transição suave entre valores.

B. Fractais

Fractais podem ser descritos, segundo [1], como objetos geométricos complexos, na qual a complexidade surge da repetição de uma forma em uma extensão de escalas. Um exemplo simples pode ser visto na Figura 3:

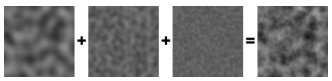

Figura 3. Exemplo de um fractal a pertir de ruído Perlin.

Os três ruídos Perlin estão em escalas diferentes e, uma vez somados, formam um fractal, segundo a definição citada. Multifractais já são um subgrupo caracterizado pela variação de sua dimensão fractal ao longo de sua localização.

C. Ridged Multifractal Noise

O *Ridged multifractal noise* é uma variação do ruído Perlin, e foi apresentado em [1]. O principal ponto do algoritmo é que ele captura a *heterogeneidade de terrenos em grande escala*, apresentando montanhas, planaltos e crateras. Os seguintes parâmetros são levados em consideração na execução do algoritmo:

- **Octaves:** Número de iterações (e, consequentemente somas) feitas sobre a função de ruído.
- **Amplitude:** Máximo valor adicionado ao valor total do ruído.
- **Frequency:** Número de valores de ruídos definidos entre dois pontos (quanto maior a frequência, maior o disturbio da textura resultante).
- **Lacunarity:** É um termo usado no cálculo de fractais, e dita o espaço entre sucessivas frequências, aumentando ou diminuindo a densidade do resultado final.
- **Offset:** Fator multifractal.
- **Tamanho:** Tamanho do mapa de altura que será salvo o resultado da geração procedural.

O tempo de execução é dependente apenas do tamanho do mapa e o número de octaves.

V. Proposta

O terreno geral é dividido em terrenos menores (chamados *patchs*), como mostra o *grid* da Figura 4. Dessa forma, apenas *patchs* de interesse do usuário (que estão mais próximos, por exemplo) precisarão ser gerados.

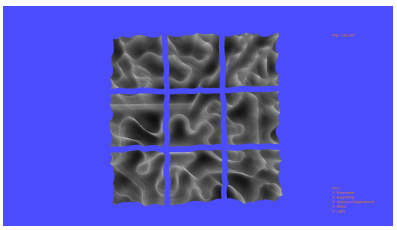

Figura 4. *Patchs* exibidos em um *grid*.

Considerando o usuário inicialmente localizado no *patch* central, ao mover-se para um *patch* vizinho, o sistema irá requisitar a geraçaõ de novos *patchs*, vizinhos a aqueles que estão na borda do grid. O número de vizinhos gerados, bem como a quantidade de vizinhos do *patch* central são variáveis do sistema, podendo ser adaptadas, pelo usuário, de acordo com o poder de processamento de sua máquina.

Para garantir uma visualização fluida do terreno, minimizando as interrupções com a geração, o sistema proposto decide qual arquitetura (GPU ou CPU) será utilizada na geração dos *patchs*. Considerando **n**

Tal métrica foi escolhida pelas seguintes razões:

- Obter um FPS mais fluido durante a navegação é um dos principais objetivos deste trabalho.
- A mesurização do tempo gasto com trabalho relacionado à parte gráfica não é algo trivial utilizando OpenGL. A extensão **GL_EXT_timer_query** [6], por exemplo, só está disponível em placas NVidia, algo que anularia a possibilidade da execução deste trabalho em placas ATI.

VI. Implementação

VII. Testes

VIII. Conclusão e Trabalhos Futuros

Referências

[1] D. S. Ebert, F. K. Musgrave, D. Peachey, K. Perlin, and S. Worley, *Texturing and Modeling: A Procedural Approach*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2002.

[2] K. Perlin, "An image synthesizer," *SIGGRAPH Comput. Graph.*, vol. 19, no. 3, pp. 287–296, 1985.

[3] ——, "Implementing improved perlin noise," in *GPU Gems*, R. Fernando, Ed. Addison Wesley Professional, March 2004, ch. 13.

[4] R. Geiss, *GPU Gems 3*. Addison-Wesley Professional, 2007, ch. 1 Generating Complex Procedural Terrains Using the GPU, pp. 7–37.

[5] J. Schneider, T. Boldte, and R. Westermann, "Real-time editing, synthesis, and rendering of infinite landscapes on GPUs," in *Vision, Modeling and Visualization 2006*, 2006.

[6] "Gl_ext_timer_query." [Online]. Available: http://www.opengl.org/registry/specs/EXT/timer_query.txt