SCAN PARALELO EM ARQUITETURA CUDA PARA JOGOS 3D

Aluna: Patricia Zalmon Rosenberg
Orientador: Bruno Feijó

Introdução

Jogos 3D requerem altíssimo desempenho de tempo real que só pode ser alcançado através de algoritmos paralelos. Existem blocos de construção de algoritmos paralelos que são usados em um grande número de situações. Um dos mais simples e úteis destes blocos de construção é a operação de somas acumulativas de prefixos (*all-prefix-sums operation*) também chamada de **soma de prefixos** ou simplesmente **scan**. O scan recebe um operador associativo binário $\oplus$ com identidade I e um vetor de n elementos $[a_0, a_1, \cdots, a_n]$ e retorna o vetor $[I, a_0, (a_0 \oplus a_1), (a_0 \oplus a_1 \oplus a_2), \cdots, (a_0 \oplus a_1 \oplus a_2 \oplus \cdots \oplus a_{n-2})]$. O scan pode ser empregado para implementar vários algoritmos, tais como *quicksort*, *quickhull*, compactação de *stream*, *summed-area tables*, ... O primeiro passo na pesquisa de scan paralelo em jogos 3D é o domínio do scan sequencial e de sua aplicação em computação gráfica.

Objetivos

O primeiro objetivo é estudar Computação Gráfica e OpenGL [1] [2] e usar a soma de prefixos sequencial (scan sequencial) na técnica de *Summed Area Table* para uma aplicação típica de renderização 3D: profundidade de campo (*depth field*). Depois, o objetivo é indicar a direção para a segunda etapa da pesquisa que consiste em usar o scan paralelo.

Metodologia

O *Summed-area table* (**SAT**) [4] de uma imagem corresponde à chamada **imagem integral**, onde cada pixel tem um valor de intensidade que é igual à soma das intensidades de todos os pixels que o precedem acima e à esquerda (e mais a dele próprio). Obtemos a imagem SAT aplicando a soma scan para todas as linhas (Figura 1b) da imagem original (Figura 1a), seguida da soma scan de todas as colunas do resultado. A Figura 1c é a imagem integral da Figura 1a (*e.g.* o pixel i=3 e j=2 tem valor 11 na Figura 1c, pois é igual a 2+3+2+3+0+1 na Figura 1a). A soma de todas as intensidades de um conjunto de pixels no retângulo $m \times n$ (chamado de **máscara de filtro**) da Figura 1d pode ser obtido pela seguinte equação:

$$T_R = s_{lr} - s_{ur} - s_{ll} + s_{ul} \qquad (1)$$

onde s_{lr} é o valor de intensidade do pixel mais embaixo e à direita da máscara de filtro (*lower-right*) na imagem integral (*i.e.* no SAT), s_{ul} é o valor do pixel mais acima e à esquerda (*upper-left*) da máscara e assim por diante. Na Equação (1), s_{ul} corresponde à soma das intensidades dos pixels da região R_{ul} na imagem original. Esta soma é subtraída duas vezes quando usamos os termos s_{ur} e s_{ll} na Equação (1). Por esta razão, somamos s_{ul} no final desta equação (ver subtração das áreas na Figura 1e). O valor final da intensidade do pixel é dado pelo valor médio da intensidade dos pixels cobertos pela máscara; portanto, o valor da aplicação deste filtro (chamado de *box filter* e que é um filtro de convolução) é dado por $s_{filtro} = T/m \times n$. No exemplo da Figura 1d, a máscara é 5×5. Os **efeitos de borda** ocorrem quando precisamos aplicar o filtro em pixels próximos às bordas da imagem. Neste caso, as opções são inúmeras e destacamos duas: ***constant pad*** (onde definimos todos os pixels fora da imagem original como tendo um valor constante, que pode ser 0) e ***clamp*** (onde repetimos os pixels das bordas indefinidamente).

Para a geração do efeito de profundidade de campo (*depth field*) numa imagem, usamos o valor da profundidade (dada pelo *depth buffer*) para calcular um fator de desfocagem e estabelecer o tamanho da máscara. Desta maneira, para a maior distância da

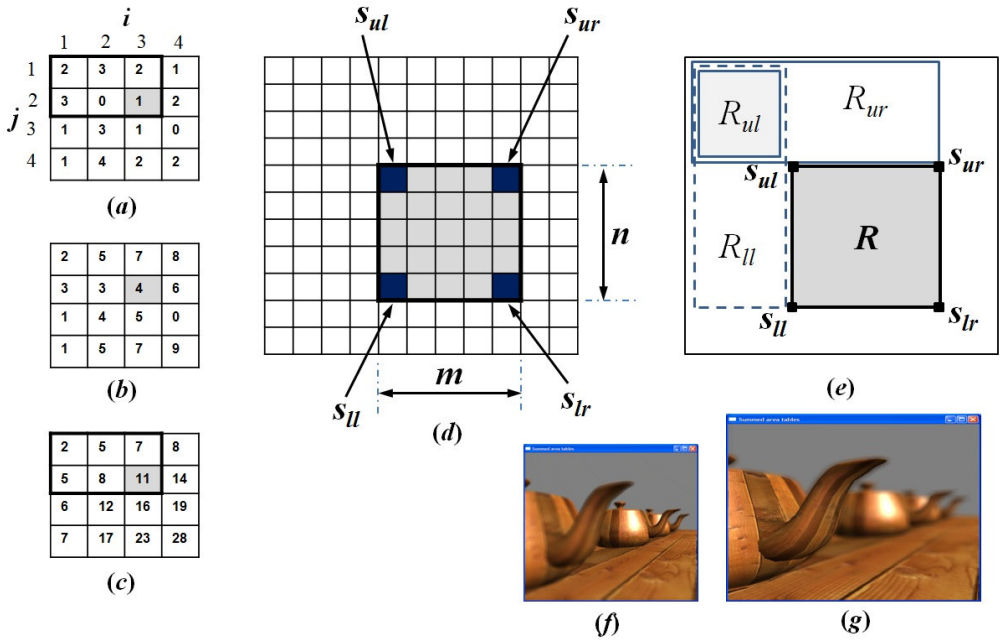

Figura 1: Processo de calcular o SAT e a soma de intensidades na região R em (f). Resultado em (g).

câmera na cena, estabelecemos como sendo o fator de desfocagem máximo igual a 1.0 (e, para este valor, arbitramos o maior valor ímpar de máscara *m=n=M*). Então, para cada pixel, lemos a profundidade correspondente e calculamos um tamanho de máscara proporcional. Na prática, geramos 3 matrizes de imagens integrais, uma para cada canal de cor (R, G, B).

Resultados

A Figura 1f é a cena gerada pelo OpenGL e a Figura 1g é o resultado do *depth field*.

Conclusões

O processo investigado revelou que o scan sequencial é lento, mas muito apropriado para ser paralelizado. A qualidade da profundidade de foco não ficou tão boa, mas isto depende de ajustes no cálculo do tamanho da máscara. O caminho para a segunda etapa pode ser encontrado em [3], [5] e [6].

Referências

[1] WRIGHT Jr, R. S., LIPCHAK, B., HAEMEL, N. **OpenGL SuperBible – Comprehensive Tutorial and Reference**, 4th Ed., Addison-Wesley, 2007.

[2] HILL Jr, F.S., KELLY, S.M. **Computer Graphics Using OpenGL**, 3rd. Ed., Prentice Hall, 2006.

[3] HARRIS, M., SENGUPTA, S., OWENS, J.D. **Parallel Prefix Sum (Scan) with CUDA**. In: H. Nguyen (Ed.), GPU Gems 3, Addison-Wesley, Chapter 39, p. 851-876, 2007.

[4] LANSDALE, R.C. **Texture Mapping and Resampling for Computer Graphics**. MSc Thesis, University of Toronto, Canada, January 1991.[disponível em: http://citeseer.ist.psu.edu/viewdoc/download?doi=10.1.1.57.4266&rep=rep1&type=pdf] [acessado em 2/07/2011].

[5] HENSLEY, J., SCHEUERMANN, T., COOMBE, G., SINGH, M. & LASTRA, A. **Fast summed-area table generation and its applications**, *Computer Graphics Forum*, 24(3), p. 547-555, 2005.

[6] GPGPU.ORG. **CUDPP – CUDA Data Parallel Primitives Library**. Disponível em http://gpgpu.org/developer/cudpp [acessado em 29/04/2010].