Normalização de Vetores Integrada ao Cálculo do Produto Escalar Utilizando Tabelas de Dispersão

Rodrigo Ferreira da Silva
Departamento de Ciência e Tecnologia Aeroespacial
DCTA
São José dos Campos, Brasil
rodsilva23@yahoo.com.br

Glauco da Silva, Herivelto Tiago Marcondes dos Santos
Faculdade de Tecnologia de Guaratinguetá
FATEC
Guaratinguetá, Brasil
glauco@fatecguaratingueta.edu.br/hmarcondes@yahoo.com.br

***Resumo*— Este trabalho teve por objetivo otimizar o cálculo do produto escalar, base para qualquer modelo de iluminação. Foi utilizada para este fim, uma tabela de dispersão, que substitui as funções de raiz quadradas, e a integração da normalização de vetores ao cálculo do produto escalar. São apresentados os resultados em gráficos que comparam o desempenho em cada algoritmo e em imagens que mostram a qualidade e o erro obtido pelos mesmos. Conclui-se que, para fins de entretenimento, uma pequena tabela de dispersão é capaz de gerar um erro aceitável e que o algoritmo final conseguiu ter o dobro da eficiência.**

Palavras-Chave-Computação Gráfica; Otimização; Produto Escalar

***Abstract*— This study aimed to optimize the calculation of the scalar product, the basis for any lighting model. Was used for this purpose, a hash table that replace the functions of the square root and a vector normalization integrated in scalar product. It presented the results in graphs that compare the performance of each algorithm and images that show the quality and the error obtained by them. It follows that, for entertainment purposes, a small hash table is able to generate an acceptable error and the final code was two times more efficient.**

Keywords-Computer Graphics; Optimization; Escalar product

I. INTRODUÇÃO

Um dos grandes desafios da computação gráfica é simular corretamente a interação da luz com o ambiente tridimensional de forma realista e eficiente.

Muitas áreas como o cinema, a medicina, a arquitetura e a indústria de jogos eletrônicos tem o interesse de obter imagens de boa qualidade e com aspecto realista [2], mas os métodos existentes, para este fim, consomem muito tempo de processamento. As aplicações que exigem respostas em tempo real, como por exemplo, *games* e simuladores, simplificam a modelagem e utilizam métodos de renderização menos sofisticados para obterem um tempo de resposta favorável e que viabilize a percepção de movimento dos personagens e objetos da cena, o que gera resultados que, em muitos casos, ficam longe da realidade.

Diante desse cenário, este trabalho tem por objetivo otimizar o cálculo do produto escalar, base para qualquer modelo de iluminação. A partir da equação do produto escalar, substitui-se as funções de raiz quadradas por uma tabela de dispersão e, através da integração da normalização de vetores ao cálculo do produto escalar, diminui-se a quantidade de operações matemáticas necessárias. Todo cálculo é executado no processador, para que, futuramente, se possa utilizá-lo em métodos híbridos, que utilizam tanto o processador quanto a placa de vídeo.

Os métodos foram desenvolvidos em um contexto de iluminação local e de reflexão difusa para fins de simplificação do ambiente de desenvolvimento.

O restante do artigo está dividido da seguinte maneira: na seção II são apresentados os métodos utilizados na otimização. Na seção III são mostrados os resultados obtidos e ao final, na seção IV, são feitas as conclusões.

II. MÉTODOS UTILIZADOS

A base do cálculo da iluminação de uma cena tridimensional é o produto escalar entre o vetor normal da superfície do objeto e o vetor que representa a direção da luz. Este cálculo possui um alto custo computacional, principalmente pelo fato de se utilizar duas funções de raiz quadradas, como em (1).

$$\alpha = \frac{nx\ .rx + ny\ .ry + nz\ .rz}{\sqrt{nx\ .nx + ny\ .ny + nz\ .nz}\ .\sqrt{rx\ .rx + ry\ .ry + rz\ .rz}} \quad (1)$$

na qual:
α = produto escalar
nx, ny, nz = vetor normal à superfície
rx, ry, rz = vetor direção da luz

A. *Tabelas de Dispersão*

Tabela de dispersão é uma estrutura que pode ser utilizada para a busca de dados e tem como característica não utilizar comparações para se encontrar o dado desejado. Para a realização da busca na tabela utiliza-se uma chave em uma função, chamada função de dispersão, que gera um índice referente à posição àquela chave na tabela. Através

deste índice pode-se acessar diretamente o local onde o dado está armazenado, sem a necessidade de comparações de valores [3].

Foi utilizada uma tabela de dispersão responsável pelo armazenamento de valores pré-calculados de raiz quadrada. O radicando da radiciação é a chave da função de dispersão que retorna o valor do índice que indica a posição na tabela onde o valor da raiz quadrada daquele radicando está armazenado, como pode ser observado na Tabela I.

$$i = f(h) \tag{2}$$

na qual:

i = índice de retorno
h = chave da função de dispersão
f = função de dispersão

TABELA I. TABELA DE DISPERSÃO COM RAÍZES PRÉ-CALCULADAS

Índice	Raiz
0	1.0
1	1.2
2	1.4
3	1.5
4	1.7

A função de dispersão utilizada, apenas transforma o valor da chave em um inteiro e utiliza um fator multiplicador para estabelecer a precisão do valor desse inteiro, como em (3).

$$f(h) = h \,.m \tag{3}$$

m = fator multiplicador múltiplo de dez

Se a chave da função de dispersão, por exemplo, possuir valores entre zero e um, e não for utilizado o fator multiplicador, o retorno será zero, para chaves menores que um, e um para chaves iguais a um. Com isso se tem apenas dois valores de índice para a tabela de dispersão. Com um fator multiplicador de valor dez, obtêm-se valores inteiros de zero a dez, ou seja, onze possibilidades de índices.

Para que haja a possibilidade da utilização de uma tabela que armazene valores pré-calculados de raiz quadrada, é necessário que se tenha uma quantidade limite de raízes, consequentemente é preciso que a função de dispersão gere uma quantidade limite de índices. Os vetores que representam a direção da luz podem adquirir qualquer valor, o que gera um problema com a utilização da tabela de dispersão empregada neste algoritmo, pois desta forma, não se tem um limite de valores de radicando, consequentemente não tendo um limite para o tamanho da tabela. A solução para este impasse é a normalização do vetor direção da luz. Com esta normalização, é gerado um vetor que representa da mesma forma a direção da luz, e cada componente deste vetor fica limitado a valores entre zero e um, garantindo a limitação da quantidade de radicandos através de uma aproximação, como pode ser observado em (4).

$$d = \sqrt{rx \,.\, rx + ry \,.\, ry + rz \,.\, rz}$$

$$rx = \frac{rx}{d} \qquad ry = \frac{ry}{d} \qquad rz = \frac{rz}{d} \tag{4}$$

$$\alpha = \frac{nx \,.\, rx + ny \,.\, ry + nz \,.\, rz}{vetor[f(nx \,.\, nx + ny \,.\, ny + nz \,.\, nz)].\, vetor[f(rx \,.\, rx + ry \,.\, ry + rz \,.\, rz)]}$$

na qual:

f(h) = função de dispersão
d = tamanho do vetor
vetor[ ] = vetor de dados que armazena os valores das raízes

Se por um lado ganhou-se desempenho com a substituição da função de raiz quadrada por uma tabela de dispersão, por outro, criou-se a necessidade de normalizar os vetores direção da luz, sendo ainda necessária a utilização de uma função de raiz quadrada, ou duas, no caso em que os dois vetores não estão normalizados. Uma forma de contornar esta situação é apresentada na próxima seção através do cálculo de normalização de vetores integrada ao cálculo do produto escalar.

B. Normalização de Vetores Integrada ao cálculo do Produto Escalar

O vetor que representa a direção da luz possui três componentes rx, ry e rz que correspondem à direção que o raio de luz atinge determinado ponto nos eixos x, y e z respectivamente.

É atribuída aos dois componentes de norma de menor valor a divisão entre a norma de cada um destes componentes pela de maior valor e mantido o sinal do valor inicial. Ao componente de norma de maior valor é atribuído o valor 1 e também mantido o sinal do valor inicial. Por exemplo, se a norma do valor de rx for maior que ry e rz, obtem-se os valores rx, ry, rz e α como pode ser visto em (5).

$$ry' = \pm \frac{||ry||}{||rx||} \qquad rz' = \pm \frac{||rz||}{||rx||} \qquad rx' = \pm 1 \tag{5}$$

$$\alpha = \frac{nx \,.\, rx + ny \,.\, ry + nz \,.\, rz}{vetor[f(rx' + ry' \,.\, ry' + rz' \,.\, rz')]}$$

na qual:

rx', ry' e rz' = valores do vetor direção da luz transformados em valores entre zero e um

O dividendo é calculado normalmente, utilizando os novos valores de rx, ry e rz. No divisor é excluído o cálculo

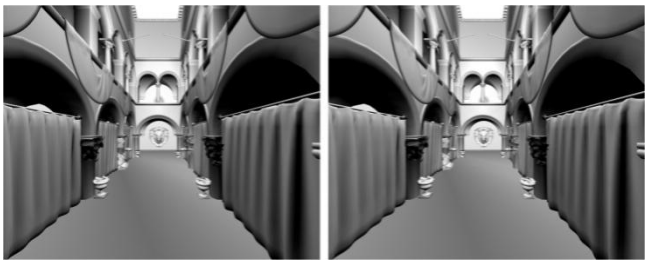
da raiz quadrada da soma dos quadrados dos componentes do vetor normal do vértice.

III. RESULTADOS

Para os testes foi utilizado o modelo 3D Sponza [1], usado frequentemente em trabalhos acadêmicos, como em [4], [5] e [6]. A partir do modelo obtido no formato obj, gerou-se um arquivo em linguagem C++ para ser utilizado nos algoritmos. O modelo 3D completo possui 117.784 polígonos de três e quatro vértices, com um total de 124.178 vértices.

Utilizou-se para os testes um computador *desktop* com processador Core 2 Duo E7400 de 2.80 Ghz e 3MB de *cache*, 4GB de RAM de 800 Mhz em *dual channel*, placa de vídeo GeForce GTX 550 TI de 1GB, *front side bus* de 1066 MHz e sistema operacional Windows 7 de 32 bits. Apenas um núcleo do processador foi utilizado, pois não se implementou o algoritmo com *threads*.

Um aspecto importante na otimização de um algoritmo é determinar o erro deste em relação à forma original. Como pode ser observado na Figura 1, o erro gerado pela utilização de uma tabela de dispersão de 21 posições é praticamente imperceptível ao olho humano.



(a) (b)

Figure 1. Comparação entre as imagens geradas pelo algoritmo (a) sem otimização e (b) otimizado com uma tabela de disperção de 21 posições.

Cada faixa de cor em RGB pode ter um valor entre 0 e 255. Considerando esta distância como 100%, pode-se determinar o valor em porcentagem do menor erro possível, como em (6).

$$P = \frac{1}{255} = 0{,}0039 \qquad (6)$$

na qual:

P = valor da menor diferença entre dois valores em uma faixa de cor RGB

Calculou-se então a diferença entre as imagens comparando cada faixa de cor de cada ponto da imagem, encontrando o erro entre eles em cada faixa de cor RGB. A diferença em cada faixa de cor é somada e dividida por três vezes a quantidade de pontos da imagem. O valor resultante da diferença para as imagens geradas pelo algoritmo sem otimização com o que utilizou uma tabela de 2001 posições foi de 0,0066%. Com uma tabela de 21 posições o erro foi de 0,71%.

O que causa o erro na imagem são os valores de raízes aproximados das tabelas de dispersão. Na Figura 2 pode-se observar onde ocorre o erro para as imagens geradas utilizando uma tabela de raízes com 21, 201 e 2001 posições. Os pontos brancos na imagem representam a ocorrência de algum erro em relação à imagem original, mesmo sendo uma mínima diferença.

Em relação ao desempenho, o algoritmo que utiliza tabelas de dispersão, apresenta-se um pouco inferior ao não otimizado, como pode ser observado na Figura 3. Isto ocorre pelo fato de que para a utilização de tabelas de dispersão neste algoritmo, é necessário que o vetor normal do vértice e o vetor direção da luz estejam normalizados, sendo que, além de ter de utilizar uma função de raiz quadrada para o cálculo de normalização do vetor, utiliza-se para cada um dos vetores, uma função de dispersão para determinar o índice na tabela com raízes quadradas pré-calculadas, o que gera maior perda de desempenho.

Também pode ser observado na Figura 3 que o algoritmo que integra o cálculo de normalização do vetor ao cálculo do produto escalar, consegue atingir até o dobro do desempenho em relação aos algoritmos anteriores.

IV. CONCLUSÂO

Com a evolução constante dos processadores e a limitação de arquitetura que as placas gráficas possuem para cálculo de propósito geral, torna-se oportuna a combinação destes dois componentes de hardware para se obter melhores resultados em termos de tempo de processamento e qualidade de imagem.

A otimização apresentada demonstrou o dobro de eficiência em relação ao algoritmo não otimizado.

O erro da imagem gerada pela aproximação das raízes quadradas nas tabelas de dispersão de 21 posições se mostrou satisfatório para aplicações que não requerem imensa precisão, como é o caso de animações e *games*.

REFERENCIAS

[1] CRYTEK. Modelo tridimensional sponza_obj.rar (3,70 MB). Disponível em <http://crytek.com/cryengine/cryengine3/downloads>. Acessado em 2010.

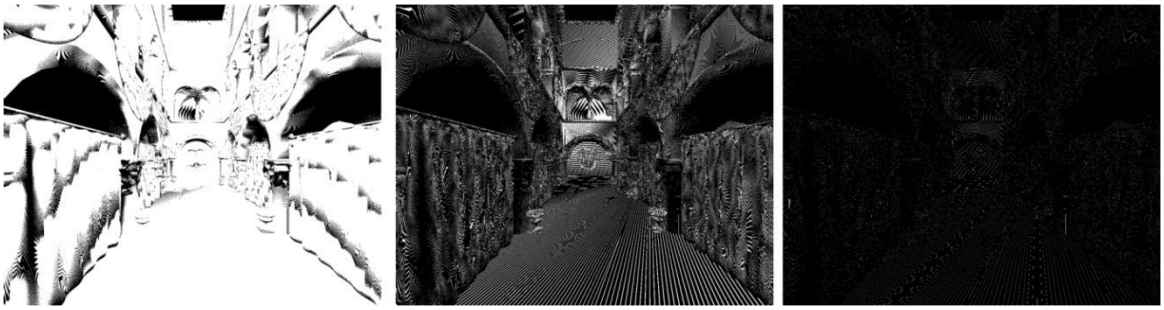
[2] DAMEZ, C. Dmitriev, K Myszkowski, K. Global Illumination for Interactive Applications and High-Quality Animations. Saarbrücken: Eurographics, 2002.

[3] DROSDEK, Adam. Estrutura de dados e algoritmos em C++. Tradução Luiz Sérgio de Castro Paiva – São Paulo: Cengage Learning, 2009.

[4] FABIANOWSKI, B.; Dingliana, J. Interactive Global Photon Mapping. Saarbrücken: Eurographics, 2009.

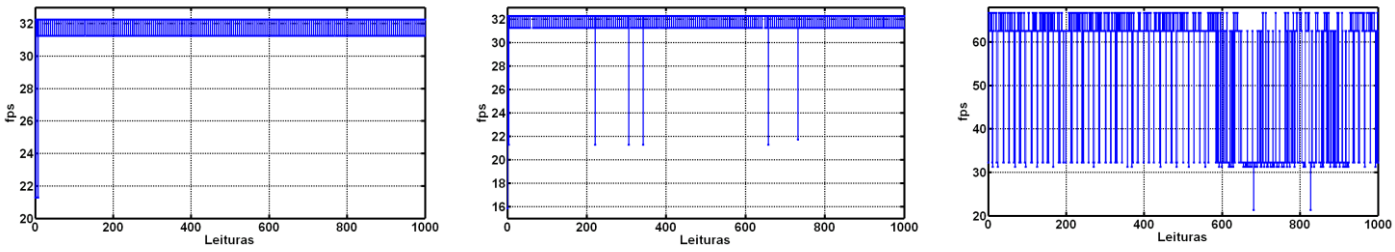
[5] GAUTRON, Pascal. et al. Radiance Cache Splatting:A GPU-Friendly Global Illumination Algorithm. Dublin: Eurographics, 2005.

[6] KAPLANYAN, Anton. Light Propagation Volumes in CryEngine 3. SIGGRAPH, New Orleans: ACM, 2009.



(a) (b) (c)

Figure 2. Diferença entre a imagem gerada sem otimização com a que utiliza uma tabela de dispersão de (a) 21 posições, (b) 201 posições e (c) 2001 posições. Onde é branco existe uma mínima diferença em relação a imagem original.



(a) (b) (c)

Figure 3. O eixo horizontal demonstra o número de leituras realizadas e o eixo vertical o valor identificado em cada leitura em fps (quadros por segundo). Os testes foram realizados com o algoritmo (a) sem otimização, (b) utilizando tabela de dispersão e (c) com integração da normalização de vetores ao cálculo do produto escalar.