USO DE *SHADERS* COMO FERRAMENTA DE AUXÍLIO AO PROCESSAMENTO DE ALGORITMOS PARA ILUMINAÇÃO E EFEITOS VISUAIS

Tiago Sanches da Silva[1], Roberto Scalco[2]

Abstract *— This work presents the auxiliary graphical library development that aims the illumination and effect algorithms implementation using the OpenGL Shading Language, directed to graphical processing into Graphic Processor Unit (GPU), in contrast to other libraries that implement its instructions in the Central Processor Unit (CPU). With the processing of shaders into GPUs there is a performance gain, once the graphic architecture cards especially is developed to work with the vectorial data. The library that is in development phase will help 3D graphics programmers to create scenes with illumination and shade, normal vectors manipulation for visual effect as bump mapping, apart from creation of reflexive surfaces using the algorithm ray tracing. Also it will be possible shape surfaces re-covered with coats or wires of hair, whose reflection is anisotropic.*

Index Terms — Computer Graphics, Photorealistic Render, Ray-Tracing, Shader.

INTRODUÇÃO

Esse trabalho versa sobre a confecção de uma biblioteca que permite ao programador elaborar aplicativos que utilizam recursos de Computação Gráfica para tornar uma cena, ou modelo tridimensional, visualmente mais atrativo aos olhos de um usuário final.

Para a confecção da biblioteca proposta, será utilizada a tecnologia *Shader*, que consiste em uma maneira simples para acessar diretamente as instruções da *Graphic Processor Unit* (GPU), permitindo que processamentos complexos e de alto custo computacional possam ser concluídos mais rapidamente e com melhor qualidade do que se fosse utilizada a *Cental Processor Unit* (CPU).

Componentes de uma Cena

Em um aplicativo de Computação Gráfica, pode-se dizer que existem três componentes fundamentais para que a imagem ou animação gerada possua boa qualidade: o observador, representado pela câmera virtual; os modelos geométricos constituídos por vértices e faces, além da iluminação e efeitos visuais.

Na fase de modelagem são definidas as formas geométricas, bem como suas posições e proporções. Ou seja, o espaço virtual é confeccionado considerando, principalmente, o dimensionamento físico dos objetos que constituem a cena.

Após a modelagem, torna-se necessário aplicar um conjunto de algoritmos que melhorem a impressão visual da cena, ou seja, deve-se aplicar um acabamento final para que a imagem possua características foto-realísticas, aproximando o ambiente virtual do real. Dessa maneira, há sensação de imersão de um usuário em um jogo [1] ou em um aplicativo gráfico qualquer (daí a necessidade de um bom posicionamento e movimentação da câmera virtual). Para criar esse realismo são utilizados algoritmos de textura como *bump mapping* [2], *parallax mapping* [3] e *displacement maping* [4]. Além disso, também podem ser utilizados algoritmos de iluminação global, como radiosidade ou *ray-tracing* [5], ou até mesmo o acréscimo de efeitos na iluminação local, como técnicas para gerar sombras, refração e outros.

Esta última etapa é essencial para obter uma alta qualidade na cena, porém o custo computacional é muito alto dificultando a criação de projetos que necessitam de tonalização em tempo real.

O Problema do Processamento

Em muitas situações, um programador deve decidir entre qualidade visual ou desempenho quando aplica um determinado algoritmo.

A aplicação do modelo de iluminação *ray-tracing* é um desses casos, uma vez que essa técnica consiste em determinar a intersecção de uma reta com os objetos da cena. Inicialmente, a reta é definida a partir do observador (representado pela câmera virtual) e passa através de cada um dos *pixels* da imagem. Após definir a intersecção entre a reta e um objeto na cena, o processo deve ser repetido para uma reta cuja direção é calculada pela mesma equação que define a reflexão de um raio de luz sobre um ponto de uma superfície. Esse processo deve ser repetido até que o raio de luz atinja uma fonte de luz ou até um determinado número de iterações, definido pelo programador.

Depois disso, o processo reverso é aplicado, ou seja, o modelo de iluminação deve ser aplicado novamente,

[1] Tiago Sanches da Silva, Centro Universitário do Instituto Mauá de Tecnologia – Escola de Engenharia Mauá, Praça Mauá, 1, sala G-02, 09580-900, São Caetano do Sul, SP, Brasil, tiago.ssilva@ceun.maua.br
[2] Roberto Scalco, Centro Universitário do Instituto Mauá de Tecnologia – Escola de Engenharia Mauá, Praça Mauá, 1, sala G-02, 09580-900, São Caetano do Sul, SP, Brasil, roberto.scalco@maua.br

adicionando à cor da luz refletida, uma parcela da cor do objeto que a refletiu, até atingir novamente o *pixel* da tela, onde a cor será exibida.

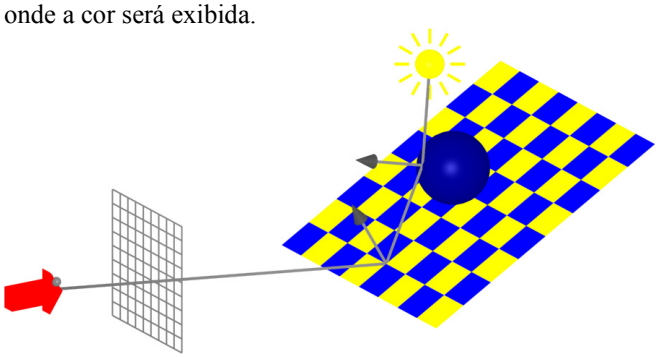

FIGURA. 1
REPRESENTAÇÃO GRÁFICA DO ALGORITMO RAY-TRACING

Esse algoritmo é normalmente aplicado em ambientes onde existam muitos fragmentos (pequenas faces que compõem superfícies e sólidos) com alto fator reflexivo, como objetos metálicos e superfícies polidas ou espelhadas. Pelo fato de ser recursivo exige um alto esforço computacional, pois há uma grande quantidade de operações matemáticas necessárias para calcular todas as intersecções e reflexões do ambiente, para cada um dos raios de luz que gerou a imagem.

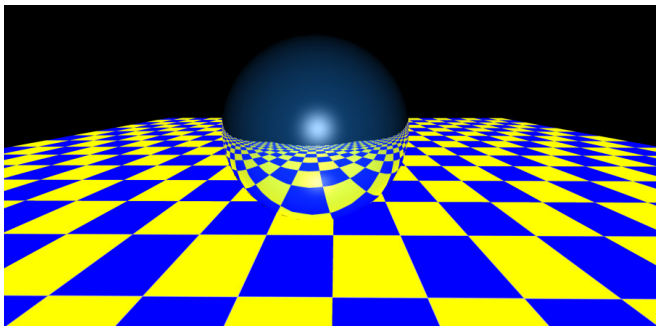

FIGURA. 2
RESULTADO DA APLICAÇÃO DO ALGORITMO RAY-TRACING

CPU × GPU

O exemplo do algoritmo *ray-tracing* mostra que um grande número de operações aritméticas devem ser calculadas. Essas operações normalmente são realizadas na CPU, que não é dedicada a operações aritméticas matriciais e vetoriais como a GPU. O processador gráfico não necessita de um gerenciamento complexo de controle de fluxo e de *caching* de dados como a CPU. Devido a esta diferença de arquitetura o poder de processamento de uma GPU é muito maior que o de uma CPU. A figura 3 mostra claramente que a medida que os anos passam esta diferença só tende a aumentar [6].

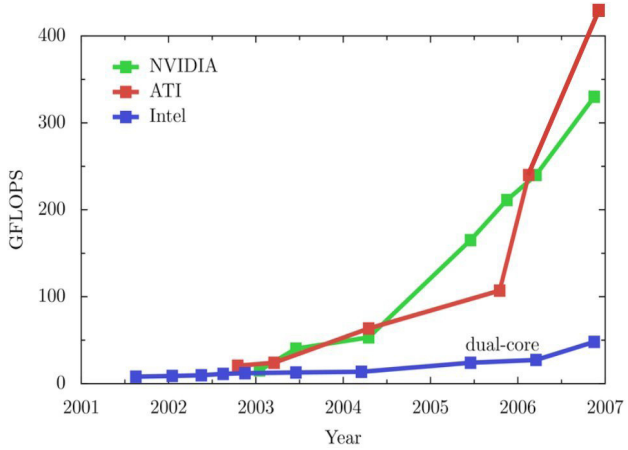

FIGURA. 3
NÚMERO DE OPERAÇÕES POR SEGUNDO COM VALORES PONTO-FLUTUANTE ENTRE DIFERENTES UNIDADES DE PROCESSAMENTO

Isso mostra que a implementação de algoritmos com uma grande quantidade de operações matemáticas na GPU, como é o caso do algoritmo *ray-tracing*, representa uma maneira de minimizar as conseqüências de perda na qualidade da cena causada pelo baixo desempenho do aplicativo. Entretanto, para enviar instruções para a GPU é necessário utilizar a tecnologia *Pixel Shader*, atualmente em sua versão 4.0, presente nas placas de vídeo mais recentes.

Shaders são *scripts* utilizados pela placa gráfica para melhorar a qualidade da tonalização de modelos tridimensionais. Os efeitos gráficos são aplicados sobre uma cena, inicialmente criadas em uma linguagem destinada à modelagem, permitindo maior detalhamento e aumento do realismo da imagem gerada.

Esses *scripts* (*shaders*) são executados diretamente na GPU, o que proporciona um desempenho muito maior comparado ao da CPU. Isso ocorre devido o fato do projeto da arquitetura interna da GPU permitir que cálculos matemáticos e processamento paralelo sejam intensamente executados, devido a maior parte dos transistores serem destinados para o processamento aritmético matricial e vetorial, limitando o seu número destinado ao *caching* de dados e controle de fluxo, ao contrario da CPU que utiliza a maior parte dos transistores para o controle de fluxo e o *caching* de dados [6].

Os *scripts* que serão executados na GPU devem ser escritos nas linguagens HLSL, caso a interface com o *hardware* gráfico seja o DirectX, ou GLSL quando a OpenGL for utilizada.

UMA BIBLIOTECA DE ALGORITMOS IMPLEMENTADOS EM *SHADER*

Com base nas tecnologias expostas, foi iniciada a confecção de uma biblioteca para auxiliar programadores da área da Computação Gráfica na criação de seus aplicativos, facilitando a inserção de elementos gráficos auxiliares que dão à cena uma melhor impressão visual.

Os *scripts* da biblioteca em desenvolvimento utilizam a linguagem GLSL e, conseqüentemente, seu suporte é realizado pela biblioteca gráfica OpenGL. O ambiente de programação utilizado é o CodeGear RAD Studio – Delphi 2007.

Inicialmente, a biblioteca deve contemplar algoritmos simples de iluminação local, considerando suas componentes ambiente, difusa e especular, sendo que o cálculo dessa última componente pode ser escolhido pela técnica apresentada em [7] ou [8].

A partir da ferramenta será possível aplicar efeitos de sombra, uma vez que os modelos de iluminação local consideram apenas a interação da luz diretamente sobre os elementos das superfícies, sem considerar se existe algum objeto entre a fonte de luz e o trecho analisado.

Um passo seguinte será implementar uma técnica de manipulação de vetores normais a partir de uma determinada textura. Essa técnica é denominada *bump mapping* e permite gerar relevos aparentes sobre uma superfície, sem alterar a sua geometria.

O tratamento de superfícies reflexivas também é o foco da biblioteca, que deve fazer uso do algoritmo *ray-tracing*, exposto anteriormente, para permitir ao programador criar elementos como espelhos ou superfícies metálicas polidas.

CONCLUSÕES

Embora a biblioteca esteja em seus primeiros passos de desenvolvimento, a equipe está satisfeita com os resultados teóricos da comparação entre o processamento de um algoritmo gráfico no processador do computador (CPU) ou da placa de vídeo (GPU).

Os primeiros testes mostram que, embora a linguagem GLSL possua apenas instruções para definir a iluminação de modo menos automático quando comparado à biblioteca gráfica OpenGL, são simples de compreender, uma vez que são definidos por equações conhecidas da Computação Gráfica.

Em versões futuras da biblioteca, pretende-se analisar a viabilidade da elaboração de algoritmos de sistemas de partículas, que permitem a representação de elementos naturais, como chuva, fogo, nuvens e outros.

AGRADECIMENTO

Ao Centro Universitário do Instituto Mauá de Tecnologia pela concessão da bolsa ao aluno-estagiário envolvido nesse projeto.

REFERÊNCIAS

[1] TORTELLI, D. M., WALTER, M. "Implementação da técnica de *Displacement Mapping* em *Hardware* Gráfico". *In: VI Simpósio Brasileiro de Jogos para Computador e Entretenimento Digital*. São Leopoldo, 2007. v.1, p.1-4. Disponível em: <www.inf.unisinos.br/~sbgames/anais/shortpapers/35439>. Acesso em: 09 out. 2008.

[2] BLINN, J. F., "Simulation of wrinkled surfaces". *In: SIGGRAPH 78*, Chicago, 1978. Computer Graphics. v.12 (ago), p. 286-292.

[3] KANEKO, T., *et al*. "Detailed Shape Representation with Parallax Mapping". *In: 11th International Conference on Artificial Reality and Telexistence*. Tokyo, 2001. p. 205-208. Disponível em: <http://www.vrsj.org/ic-at/papers/01205.pdf>. Acesso em: 09 out. 2008.

[4] COOK, R. L. "Shade trees". *In: SIGGRAPH 84*, 1984. Computer Graphics. v. 18 (jul). p. 223 – 231.

[5] WHITTED, T, "An Improved Illumination Model for Shaded Display". Communications of ACM (jun), v. 23, n. 6, p. 343-349, 1980.

[6] CUNO, A., VIANA, J. R. M., "Conceitos de programação em GPU". Laboratório de Computação Gráfica – UFRJ. Rio de Janeiro. Disponível em: <http://www.lcg.ufrj.br/Cursos/GPUProg/gpuintro>. Acesso em 10 out. 2008.

[7] BLINN, J. F, "Models of light reflection for computer synthesized pictures". *In: SIGGRAPH 77*, San Jose, 1977. Computer Graphics. v. 11 (jul). p. 192-198.

[8] PHONG, B. T, "Illumination for computer generated pictures". Communications of ACM (jun), v. 18, n. 18, p. 311-317, 1975.