

# Avaliação do impacto de efeitos com Deferred Shading

Bruno Duarte Corrêa    Thiago Dias Pastor    Ricardo Nakamura  
Departamento de Engenharia de Computação e Sistemas Digitais  
Escola Politécnica da Universidade de São Paulo, Brazil



Figura 1: *deferred shading* e conteúdo do *geometry buffer*

## Resumo

A viabilidade de se incorporar efeitos visuais e técnicas gráficas em um jogo depende de diferentes critérios como desempenho, estética e experiência do jogador. Esse artigo descreve uma metodologia de avaliação do impacto no desempenho de jogos a partir da adição de efeitos visuais em uma arquitetura de renderização baseada em deferred shading.

**Palavras chave:** deferred shading, análise de desempenho, shaders programáveis.

**Authors' contact:**

{bruno.duarte, thiagodiaspastor}@gmail.com  
ricardo.nakamura@poli.usp.br

## 1. Introdução

*Deferred shading* é uma técnica de renderização que recentemente vem ganhando muito espaço entre os desenvolvedores de jogos. Apesar de ter sido proposta inicialmente por Deering et al. [1988] há mais de vinte anos, somente com o hardware gráfico atual é possível utilizá-la em tempo real em computadores pessoais.

A proposta desse trabalho é analisar o impacto de efeitos visuais adicionados progressivamente sobre o desempenho de um renderizador baseado em *deferred shading*. O renderizador utilizado para a análise é implementado na plataforma XNA, de forma a ser compatível tanto com computadores PC como consoles Xbox.

Existem diversas classes de efeitos, algumas das quais causam significativa redução de desempenho e outros (como a técnica de *normal mapping*) que apresentam menor impacto. Uma métrica interessante na escolha de quais e quantos efeitos utilizar é a perda de desempenho oriunda da sua utilização combinada. Em função das particularidades de implementação dos efeitos, tem-se como hipótese que o impacto da combinação de técnicas não é a simples soma dos impactos individuais – a verificação de tal hipótese é um motivador deste trabalho.

Por questão de organização o trabalho foi dividido em quatro etapas: implementação do renderizador baseado em *deferred shading*; implementação dos efeitos visuais; elaboração do ambiente de teste e análise de desempenho.

## 2. Trabalhos relacionados

Na indústria de jogos, pode-se encontrar discussões sobre as possíveis escolhas de projeto e os impactos das mesmas na renderização [Valient 2010].

Em relação à análise de desempenho, Postma (2009) inicia uma discussão interessante sobre a influência do número de luzes no tempo total de um render deferred, porém com um foco voltado a comparação desta técnica com as mais tradicionais.

### 2.1. Deferred Shading

Em linhas gerais, a técnica deferred shading consiste em efetuar num primeiro passo a extração de alguns dados (como normais, diffuse color...) dos modelos de uma cena e salva-los em buffers, como ilustrado na Figura 1. A seguir efetua-se o processo de shading sobre estas texturas e depois combina-se as informações de iluminação com as do primeiro passo e obtêm-se uma imagem final. Por fim esta imagem é enviada para uma etapa de pós-processamento [Policarpo e Fonseca 2005].

Os principais benefícios do uso de *deferred shading* envolvem a eliminação de cálculos de iluminação desnecessários – o tempo para se calcular a iluminação de uma cena independe da sua complexidade em número de polígonos. Adicionalmente, a iluminação é calculada por pixel, permitindo-se o uso de modelos de iluminação de melhor qualidade. Por fim, as técnicas de *deferred shading* podem ser modularizadas, permitindo-se a combinação de diversos efeitos em uma cena.

Por outro lado, esta técnica apresenta também algumas desvantagens. Em primeiro lugar, destaca-se o alto consumo de banda de memória da placa de vídeo. O suporte a múltiplos *render targets* (resumidamente, a capacidade de gerar múltiplas imagens simultaneamente) também é um requisito que limita a aplicação de *deferred shading* no caso de hardware mais antigo – ou mesmo relativamente recente: placas sem suporte a DirectX 10, por exemplo, têm limitações que impedem o uso de anti-aliasing em hardware em múltiplos *render targets*. Além dessas limitações tecnológicas, as técnicas de *deferred shading* em geral não tratam corretamente transparência ou translucidez.

### 3. Implementação do renderizador

Para fins de organização, dividiu-se o renderizador em etapas, seguindo a arquitetura descrita por Policarpo e Fonseca [2005]. Desta forma, cada etapa realiza as seguintes funções:

- 1 Para cada objeto da cena, extrair informações dos e guardá-las em alguns buffers chamados coletivamente de *geometric buffer* (G-Buffer).
- 2 Para cada fonte de luz na cena, utilizar as informações do G-Buffer para calcular a sua influência em cada objeto e salvar estas informações em um buffer auxiliar.
- 3 Combinar as informações do buffer auxiliar mencionado no item 2 com as do G-Buffer e salvá-la no *frameBuffer*.
- 4 Realizar pós-processamento da imagem.

#### 3.1. Geometric Buffer

O G-Buffer desta implementação contém quatro buffers distintos: *normal buffer*, *diffuse buffer*, *depth buffer* e *glow buffer*, descritos a seguir.

O *normal buffer* armazena a informação dos vetores normais dos objetos a serem renderizados, codificadas como cores em uma textura.

O *diffuse buffer* armazena as informações das cores dos objetos a serem renderizados.

O *depth buffer* armazena a profundidade dos objetos em relação à câmera.

O *glow buffer* armazena informações sobre *self-luminance* dos objetos. A utilização das informações contidas no *glow buffer* não será tratada neste artigo.

Para se gerar um G-Buffer, é necessário criar, configurar e inicializar adequadamente cada um dos *render targets* (abstração de uma região da memória da placa de vídeo). As placas de vídeo têm uma limitação importante em relação aos MRTs: todos devem utilizar o mesmo número de bits por pixel. Logo, é necessário uma análise cuidadosa antes de se escolher esta propriedade. Em geral, as três opções são 16, 32 e 64 bits por pixel, sendo que 32 bits é a solução de melhor compromisso entre precisão e consumo de memória. Deve-se observar que embora o número de bits por pixel de cada *render target* seja o mesmo, a informação representada em cada um é diferente. Por exemplo, um *diffuse buffer* de 32 bits armazena uma cor no formato RGBA com 8 bits por canal, enquanto o *depth buffer* armazena um único valor numérico por pixel.

#### 3.2. Iluminação

A entrada desta etapa consiste no G-Buffer (embora o *glow buffer* não seja usado neste ponto) gerado e de

informações de cada uma das fontes de luz da cena (tais como posição, cor e decaimento). A saída desta etapa é um buffer auxiliar contendo apenas informações sobre como as luzes afetam a cor dos objetos. Este buffer é comumente chamado de *LightRT*. Deve-se observar que nesta fase realiza-se apenas processamento de imagens, no qual as entradas e saídas são texturas 2D.

É importante salientar o desacoplamento entre as etapas do renderizador: não importa, por exemplo, se os vetores normais utilizados no cálculo da iluminação originaram-se de uma técnica de *normal mapping* ou foram carregados junto com o modelo 3D a ser renderizado.

O cálculo da influência de cada fonte de luz depende do tipo da mesma. Para luzes direcionais, realiza-se o processamento de todos os pixels da imagem. Consequentemente, é o tipo de fonte de luz que mais consome tempo de processamento da placa gráfica e seu uso deve ser limitado.

O cálculo do efeito de luzes pontuais pode ser otimizado, desenhando-se uma esfera na posição (na cena 3D) da fonte de luz e aplicando-se o cálculo de iluminação apenas na região da projeção 2D dessa esfera. O cálculo de luzes do tipo *spot* é bastante semelhante ao de luzes pontuais, utilizando-se um cone ao invés de uma esfera.

#### 3.3. Pós-processamento

Depois dos cálculos de iluminação, segue-se uma fase que combina as informações de iluminação com as dos modelos e compõe a cena final. Ela tem como entradas o G-Buffer e o LightRT (todos 2D) e como saída a imagem final (também 2D).

Por fim, esta imagem gerada pode ser submetida a efeitos de pós-processamento como, por exemplo, *bloom* ou *high dynamic range rendering* (HDRR) e a seguir vai para o *frameBuffer* da placa de vídeo.

A codificação do renderizador foi subdividida em duas partes. O objetivo da primeira etapa foi implementar uma estrutura central para suportar a comunicação entre as quatro etapas anteriores. Esse módulo consiste em interfaces e elementos de comunicação cujo papel é controlar quais informações vão trafegar entre os passos. O envio de dados para a GPU faz parte desta fase. A segunda etapa consiste na implementação das interfaces acima citadas, especificando a maneira como cada operação é realizada. A maioria das técnicas consiste em implementar uma dessas interfaces.

### 4. Seleção das técnicas a serem implementadas

Os efeitos implementados foram escolhidos frente à observação dos *game engines* Unreal, Unity e Leadwerks e de jogos que adotaram *Deferred Shading* como renderizador: Starcraft 2, KillZone 2, Stalker: Shadow of Chernobyl e Tabula Rasa. Procurou-se utilizar como referência algoritmos bem

conhecidos para a implementação dos efeitos, encontrados em [Zima 2010; Akenine-Moller, Haines e Hoffman 2008; Nguyen 2007; Pharr 2005].

Neste artigo, será discutida a avaliação de apenas dois dos efeitos selecionados: *Normal Mapping* e *Screen-Space Ambient Occlusion*.

#### 4.1. Normal Mapping

Trata-se de um efeito que consiste em se utilizar vetores normais armazenados em uma textura ao invés de calculá-las usando informações dos vértices dos modelos. O principal benefício é a adição de detalhes sem a necessidade de um modelo com maior número de polígonos.

Seguindo o desacoplamento citado inicialmente, esta técnica consiste apenas em implementar uma interface e criar um shader, a parte de iluminação, por exemplo, não será alterada.

#### 4.2. SSAO (Screen Space Ambient Occlusion)

Esta é uma técnica bastante recente que consiste em uma maneira eficiente de simular oclusão de iluminação ambiente em tempo real, sendo utilizada pela primeira vez no jogo Crysis em 2007.

A implementação adotada é bastante semelhante à encontrada no game engine da empresa CryTek, consistindo em pós-processamento da imagem. Esta técnica é bastante custosa, pois exige muitas leituras de texturas (Pixel Shader Bounded) e muitas transformações entre espaços. Vale ressaltar que ela independe completamente da complexidade da cena.

### 5. Resultados

#### 5.1. Ambiente de testes

Inicialmente foi construído um *framework* para extrair informações estatísticas sobre a cena (número de polígonos por quadro renderizado, número médio de quadros por segundo, tempo que uma função demora em executar), tomando-se o cuidado de interferir o mínimo possível no desempenho do renderizador.

A seguir construiu-se um cenário bastante simples com poucos objetos, tendo ao todo 12703 polígonos.

O teste consiste na adição de geometrias e luzes Point simultaneamente a uma cena. A Figura 2 apresenta uma imagem do ambiente de teste em execução.

Para que o teste reproduzisse de forma mais próxima o ambiente de execução de um jogo, a cena conta com colisões e simulações físicas (JigLibX), entretanto, esse atraso adicionado pode ser ignorado para fins de comparação, uma vez que aparece igualmente em todos os testes.

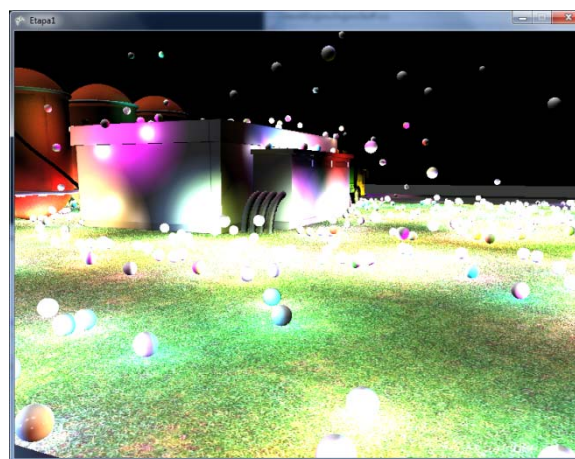


Figura 2 Ambiente de testes

Outra característica importante comum a todas as cenas é o fato de a câmera estar posicionada estaticamente e de uma maneira que abrangesse todos os objetos adicionados, evitando dessa forma que o culling interno interferisse nas medições. Foram realizadas três baterias de testes com as seguintes características:

- Esferas simples, com textura e iluminação padrão.
- SSAO. Esferas simples como o teste anterior
- Esferas com NormalMapping e textura, sem SSAO.

Os testes foram realizados em um PC HpTouchSmart com 4Gb de RAM, processador AMD Athlon II X2 2.7GHz com uma placa de video ATI HD3200.

#### 5.2. Análise dos dados

Na Figura 3, são exibidas o número de chamadas dos métodos Draw e Update por segundo e a soma dos dois valores. Com esse valores foram calculadas gráficos de tendência para suavizar o fato de que os objetos são adicionados de forma pontual no tempo. Os pontos iniciais devem ser desconsiderados pois sofrem a interferência do processamento de construção da cena.

A curva de cor verde representa o acumulado das outras duas variáveis e pode ser utilizado como métrica principal uma vez que os outros parâmetros têm comportamento parecido.

O gráfico de SSAO apresenta um comportamento condizente com a maioria dos efeitos de pós-processamento, tendo um valor inicial consideravelmente menor, entretanto a taxa com que o desempenho cai é próxima à taxa do gráfico sem efeito, o que não acontece no exemplo com *normal mapping*. Este apresenta um comportamento inicial bastante semelhante ao exemplo sem efeitos, entretanto tem uma taxa de queda de desempenho mais acentuada sendo visivelmente, linearmente dependente do número de objetos na cena.

É importante salientar nos testes em questão, a cada objeto adicionado, também é adicionada uma

fonte de luz pontual; o efeito dela é semelhante ao desenho de mais uma esfera na cena (conforme descrito anteriormente).

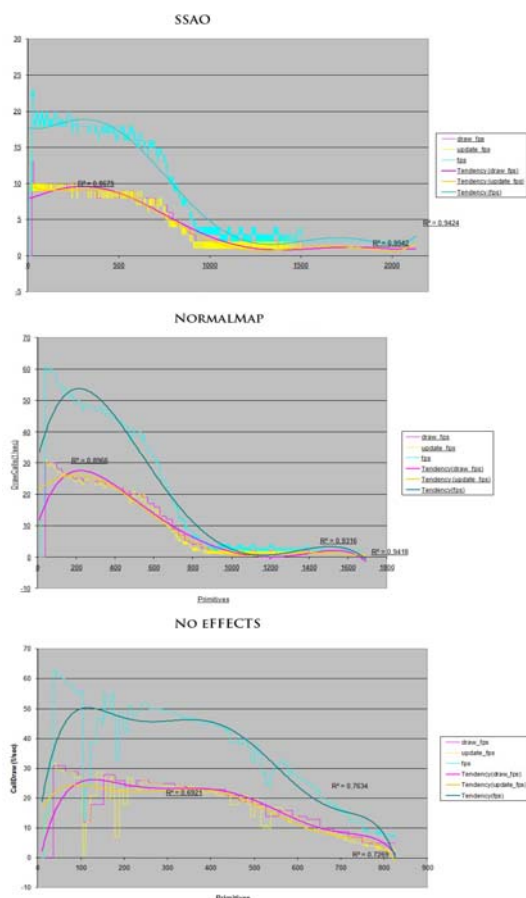


Figura 3: Resultados do teste de desempenho

## 6. Conclusões

Comprovou-se que a arquitetura deferred abre um novo leque de possibilidade em relação à iluminação. Nesses testes foi possível adicionar cerca de 800 luzes.

O gerenciamento de efeitos, que inclui a adição e a manutenção dos mesmos é bastante facilitada, uma vez que as alterações são bastante localizadas e respeitam interfaces bem definidas.

### 6.1. Problemas encontrados

A configuração dos buffers que compõem o G-Buffer, que inclui tanto a quantidade de bits quanto a maneira como serão alocados em canais foi um dos grandes problemas.

O fato de as placas de vídeo atuais exigirem que todos os buffers de saída de um MRT tenham a mesma quantidade de bits aliado ao compromisso de qualidade/ desempenho forçou a escolha de uma configuração de 32 bits.

Um segundo problema foi a definição do número de buffers do G-Buffer. Devido a restrições de performance do Xbox utilizou-se somente quatro *render targets*.

## 6.2. Trabalhos futuros

Como trabalhos futuros, pretende-se identificar a relação de outros efeitos individualmente e entre si, agrupando-os em classes definidas pela sua principal dependência em termos de desempenho.

Até o momento pudemos identificar algumas classes principais: Dependentes do número de objetos (*Normal Map*, *Specular Map*, *Glow Map*), dependentes do número de luzes e geometria (*Shadow Map*, Reflexão dinâmica) e dependentes da resolução da tela (em geral técnicas de pós-processamento).

## Referências

- AKENINE-MOLLER, T, HAINES, E. E HOFFMAN, N., 2008. Real-time Rendering. Wellesley: AK Peters.
- DEERING, M; WINNER, S; SCHEDIWIY, B.; DUFFY, C. AND HUNT, N. *The Triangle Processor and Normal Vector Shader: A VLSI System for High Performance Graphics*. In: Anais da 15th annual conference on computer graphics and interactive techniques (SIGGRAPH 88). 22, 4. New York: The ACM Press, 1988. p. 21-30.
- NGUYEN, H. (ED.), 2007. GPU Gems 3. Addison-Wesley.
- POLICARPO, F. E FONSECA, F., 2005. *Deferred Shading Tutorial* [online]. Disponível em: [http://bat710.univ-lyon1.fr/~jciehl/Public/educ/GAMA/2007/Deferred\\_Shading\\_Tutorial\\_SBGAMES2005.pdf](http://bat710.univ-lyon1.fr/~jciehl/Public/educ/GAMA/2007/Deferred_Shading_Tutorial_SBGAMES2005.pdf) [Acesso em 4 Agosto 2010].
- PHARR, M. (ED.), 2005. GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation. Addison-Wesley.
- POSTMA, B., 2009. *An Evaluation of Deferred Shading Under Changing Conditions*. Student thesis, University of Groningen.
- VALIENT, M. *Deferred Rendering in Killzone 2* [online]. Disponível em: <http://www.guerrilla-games.com/publications/>
- ZIMA, C., 2010. *Catalin's XNA Experiments* [online]. Disponível em: <http://www.catalinzima.com/tutorials/deferred-rendering-in-xna/> [Acesso em 4 Agosto 2010].