Universidade Federal de Pernambuco
Graduação em Ciência da Computação
Centro de Informática

2007.2

EFEITOS DE ILUMINAÇÃO REALISTAS UTILIZANDO IMAGENS HDR

Trabalho de Graduação

Aluno – Guilherme de Sousa Moura, gsm@cin.ufpe.br
Orientadora – Judith Kelner, jk@cin.ufpe.br
Co-orientadora – Veronica Teichrieb, vt@cin.ufpe.br

29 de Janeiro de 2008

"Imagination is more important than knowledge..."
Albert Einstein (1879-1955)

RESUMO

Em Computação Gráfica, existe uma vertente de estudos que busca aprimorar as técnicas de síntese de imagens de forma a reproduzir com máxima fidelidade cenas realistas. Esta vertente é conhecida como renderização foto-realista. Tais estudos possuem aplicações diretas em jogos e soluções de realidade virtual e realidade aumentada.

Entre as abordagens utilizadas para alcançar o objetivo de produzir cenas sintéticas semelhantes às cenas reais, existe o estudo realizado sobre os efeitos decorrentes da visualização direta de fontes de luz. Por exemplo, ao focar diretamente um farol de carro ou uma lâmpada acesa, ocorre uma série de efeitos quando os raios luminosos atingem a lente receptora, podendo ser visualizados na forma de raios concêntricos, de um círculo gradiente ao redor da fonte luminosa, entre outros.

Em muitas aplicações 3D convencionais, a utilização destes efeitos é desconsiderada, ou muito simplificada, resultando em fontes de luz que são facilmente identificadas como artificiais. O foco de estudo deste trabalho foi a implementação de alguns destes efeitos, de forma a aumentar o nível de realismo de cenas sintéticas. Para alcançar este objetivo, foram estudados os efeitos *Bloom*, *Glare*, *Dazzling Reflection* e *Exposure Control*, que acontecem naturalmente em cenas reais. Os algoritmos de cada efeito foram escritos na linguagem de *shader*, de forma a ser executada diretamente nas placas gráficas, possibilitando assim a sua aplicação em tempo real. O resultado visual obtido foi satisfatório e o desempenho da aplicação demonstrou a possibilidade de utilizar os algoritmos em aplicações de realidade virtual, realidade aumentada e jogos.

Palavras-chave: Computação Gráfica, Simulação, Iluminação, Foto-realismo, GPU.

AGRADECIMENTOS

Agradeço em primeiro lugar a minha família, que me deu apoio durante todo a minha formação profissional, aderindo a minhas decisões pessoais e ajudando nos momentos mais difíceis.

Agradeço a minha orientadora, Judith Kelner, que me deu a oportunidade de trabalhar junto com ela, e dedicou seu tempo na minha formação e na elaboração deste trabalho.

Agradeço a minha co-orientadora, Veronica Teichrieb, pela orientação dada no desenvolvimento deste projeto e durante a escrita da monografia.

Agradeço a Saulo Pessoa e Pedro Leite pelo apoio no desenvolvimento do projeto, e pela orientação na escrita da monografia.

Agradeço a todos meus colegas que me incentivaram na escolha do projeto e durante o desenvolvimento do mesmo.

Agradeço finalmente a Sarah Costa pelo apoio e pela motivação especiais na fase final do desenvolvimento do projeto e da escrita da monografia.

ASSINATURAS

Este Trabalho de Graduação é resultado dos esforços do aluno Guilherme de Sousa Moura, sob a orientação da professora Judith Kelner, conduzido no Centro de Informática da Universidade Federal de Pernambuco. Todos abaixo estão de acordo com o conteúdo deste documento e os resultados deste Trabalho de Graduação.

Recife, 29 de Janeiro de 2008

Judith Kelner
Orientadora

Veronica Teichrieb
Co-orientadora

Guilherme de Sousa Moura
Graduando

ÍNDICE

1. Introdução

Em Computação Gráfica, ao sintetizar uma cena tridimensional existe a opção de representar os elementos da cena de forma artificial ou procurar simular os elementos presentes no mundo real de forma a obter um resultado visual realista. Esta última linha é conhecida como foto-realismo, e tem como objetivo produzir o mesmo resultado visual de uma cena real, sem necessariamente se preocupar com o comportamento físico real que originou aquele resultado visual [1].

O processo de síntese foto-realista envolve vários elementos do processo de síntese normal (ou não foto-realista), como texturização, animação ou iluminação. Este Trabalho de Graduação foca na iluminação da cena, mais especificamente nos efeitos causados pela visualização direta de fontes de luz presentes na cena virtual, buscando simular o resultado visual foto-realista.

1.1. Motivação

A busca por realismo em cenas sintetizadas por computador é um desafio constante em aplicações de computação gráfica. Entretanto, os algoritmos necessários para alcançar resultados visuais satisfatórios muitas vezes requerem um grande volume de processamento de dados, e aplicações em tempo real só foram possíveis com o advento das placas gráficas. Estas placas possuem atualmente poderosos microprocessadores dedicados ao processamento gráfico – chamados de *Graphics Processing Units* (GPUs) –, que abrem o leque de oportunidades para algoritmos que podem utilizar este recurso computacional extra.

Uma das aplicações que foram beneficiadas com este progresso foram os jogos, que muitas vezes precisam mostrar animações e efeitos gráficos em tempo real ao jogador. Os efeitos de iluminação, que a princípio eram bastante simplificados, começaram a se tornar mais sofisticados, algumas vezes respeitando o comportamento físico real. Com isso, os jogos produzidos atingiram um alto nível de realismo visual, como pode ser observado na Figura 1, onde os jogos World of Warcraft [2] e Half-Life 2 [3] implementam efeitos de iluminação realistas em tempo real.

Figura 1. À esquerda, uma cena do jogo World of Warcraft, da Blizzard; à direita, uma cena do jogo Half-Life 2, da Valve.

Além da indústria de jogos, aplicações de Realidade Virtual (RV) em geral e de Realidade Aumentada (RA) possuem requisitos de realismo em várias situações. Particularmente em RA, um dos grandes desafios é exibir informações virtuais integradas com cenas reais de forma que sejam indistinguíveis dos objetos reais contidos na cena, como ilustrado na Figura 2. Desta forma, o usuário poderia interagir mais naturalmente com a interface. Uma maneira de alcançar este objetivo é sintetizar as informações virtuais explorando as técnicas de foto-realismo, respeitando a iluminação do mundo real [4].

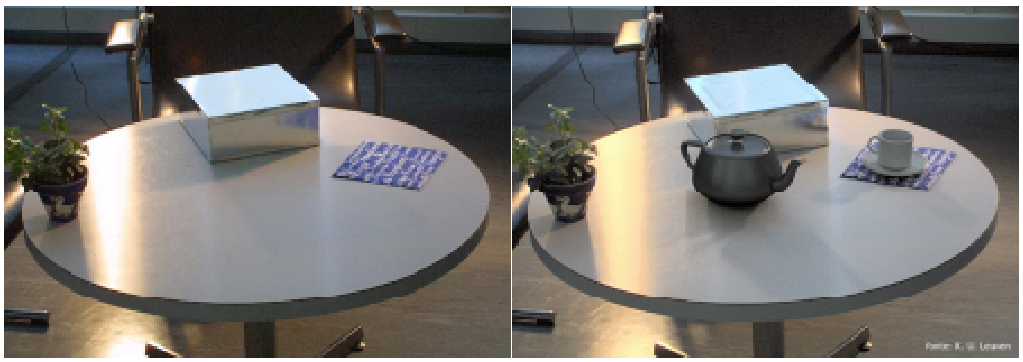

Figura 2. Exemplo de aplicação de RA onde os objetos à direita foram inseridos com a preocupação de manter o realismo da cena.

1.2. Objetivos

O objetivo deste trabalho envolve o estudo sobre os efeitos de iluminação mais comuns pesquisados pela comunidade acadêmica, seguido do desenvolvimento de um aplicativo interativo para demonstrar os resultados da implementação de um conjunto destes efeitos. O foco do estudo serão alguns efeitos causados pela visualização direta da fonte de luz, analisando, quando possível, o que ocorre fisicamente com o raio luminoso ao entrar na lente

receptora, de forma a gerar um resultado próximo do mundo real. Entretanto, devido às limitações computacionais, as simplificações necessárias serão avaliadas de acordo com a literatura para a escolha do modelo final. Dessa forma, o escopo do trabalho consiste em gerar o resultado visual foto-realista, sem a preocupação do comportamento físico real.

Para a implementação do aplicativo interativo será utilizada uma biblioteca de abstração gráfica de modo a facilitar o desenvolvimento do protótipo. O motor gráfico será o OGRE 3D [5]. Dessa forma, será possível concentrar os esforços na solução do problema, diminuindo o tempo de programação da aplicação final. O requisito desta aplicação é permitir ao usuário navegar livremente pela cena, de forma a analisar os efeitos implementados, mantendo o nível de interatividade, para comprovar que tais efeitos podem ser utilizados em jogos e aplicações de RV e RA em tempo real.

1.3. Organização do Documento

Este documento foi organizado em cinco capítulos para melhor entendimento do trabalho desenvolvido. Além do presente capítulo de introdução, os capítulos seguintes foram estruturados desta forma:

- No segundo capítulo, serão apresentados os principais conceitos relacionados ao trabalho. Os conceitos apresentados servirão de base para o entendimento da solução final. No decorrer do capítulo, serão apresentados os estudos de outros autores que de alguma forma contribuíram para o tema.
- No terceiro capítulo, a implementação da solução proposta será detalhada em vários passos, até chegar ao resultado integrado final. Também serão apresentadas as dificuldades encontradas no decorrer do desenvolvimento do trabalho.
- No quarto capítulo, será mostrado o resultado visual obtido com a solução final. Juntamente com os resultados visuais, serão apresentados os resultados de desempenho.
- No quinto capítulo, serão apresentadas as conclusões deste Trabalho de Graduação, destacando as suas contribuições. Também serão discutidas sugestões para aprimoramentos que possam ser realizados para melhorar seu resultado.

2. Conceitos Relacionados

A busca por maior realismo em imagens sintetizadas por computador produziu vários resultados ao longo do tempo. Várias inovações foram necessárias para melhorar o resultado visual, desde um modelo de representação mais fiel para o armazenamento das imagens até a subdivisão do problema em partes menores, como os efeitos de iluminação em particular.

Nas seções a seguir serão detalhados os conceitos mais importantes que surgiram destes estudos.

2.1. Imagens HDR

Para melhorar o realismo das cenas sintéticas foi necessário primeiramente adaptar o modelo de representação de imagens. A partir da década de 80 começaram a surgir implementações práticas do conceito de imagens conhecidas como *High Dynamic Range* (HDR).

Imagens convencionais, chamadas de *Low Dynamic Range* (LDR), possuem uma faixa de armazenamento para as cores que permanece sempre em 8 *bits* por cor. Ou seja, em uma imagem normal, os componentes Vermelho, Verde e Azul (RGB) podem assumir valores independentes entre 0 e 255.

Estes valores, apesar de conseguirem representar com fidelidade várias condições naturais de iluminação, são bastante limitados em situações onde a variação de luminosidade é muito grande. Por exemplo, uma situação onde seja necessário representar uma imagem de uma janela em um quarto. Nesse caso, a iluminação oriunda do exterior é muito mais intensa do que a iluminação no interior do ambiente, então todos os valores do exterior ficariam limitados pelo valor máximo de uma imagem LDR (R: 255, G: 255, B: 255), e toda a informação adicional sobre os elementos do exterior seria perdida. A Figura 3 mostra uma situação na qual este efeito pode ser visualizado.

Figura 3. Cena onde parte da informação sobre o ambiente externo é perdida.

Em contrapartida, imagens HDR podem armazenar 16 ou até 32 *bits* por componente de cor. Porém, por questões de memória, normalmente é utilizado o formato de 16 *bits*, ou seja, é possível atribuir para cada componente um valor entre 0 e 65535. Com isso, é possível representar com grande fidelidade o nível de luminosidade de uma cena, atribuindo valores elevados para fontes de luz intensas e valores pequenos para objetos pouco reflexivos. A Figura 4 ilustra a diferença entre LDR e HDR.

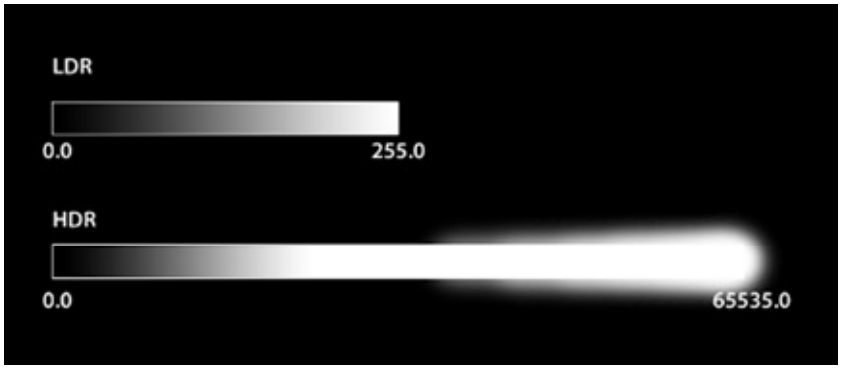

Figura 4. Diferença de faixa de representação entre imagens LDR e HDR.

Paul Debevec [6] foi um dos pioneiros a pesquisar um método de extrair informação HDR de fotografias convencionais através da composição da cena em vários níveis de exposição. Seu trabalho acabou tornando-se referência e as imagens geradas por ele utilizadas em muitas aplicações. Debevec chamou as imagens geradas por ele de *Light Probes*, e existem dois tipos de mapeamento disponíveis: esférico e cúbico, como pode ser observado na Figura 5. Neste

Trabalho de Graduação foram utilizados os mapas esféricos, porém existe a possibilidade de estendê-lo para utilização de mapas cúbicos.

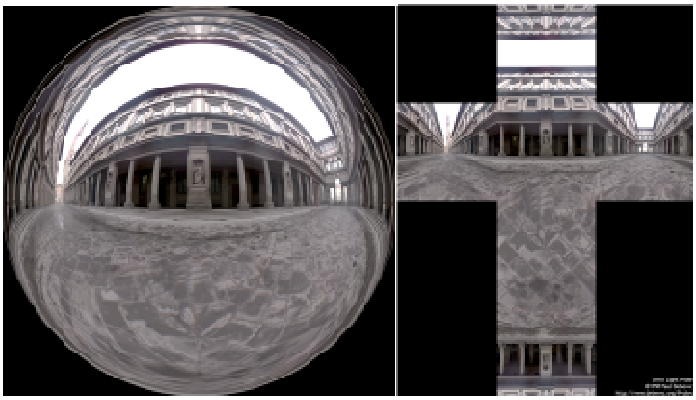

Figura 5. *Light Probes*: à esquerda, um exemplo de mapa esférico; à direita, um exemplo de mapa cúbico.

Para mapear os mapas esféricos (que são imagens bidimensionais) em uma estrutura tridimensional foi utilizada uma equação de mapeamento esférico, explorada em *Environment Maps* [7]. Esta equação é mostrada na Figura 6, onde o vetor correspondente a cada vértice (x, y, z) é projetado no plano (xt, yt).

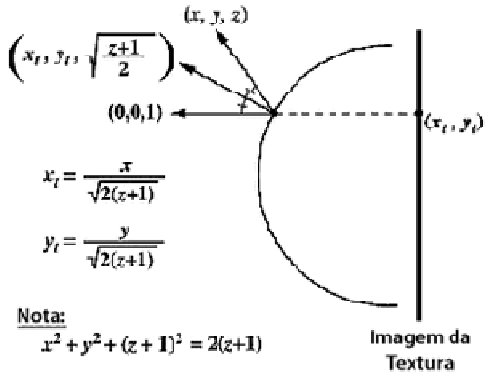

Figura 6. Equação de projeção de uma textura bidimensional em um mapa esférico.

O problema em tentar visualizar uma imagem HDR está na limitação dos monitores, já que estes não são capazes de mostrar todos os valores armazenados na imagem. Então, é necessário escolher uma faixa que será visualizada e utilizar métodos para comprimir a cor dentro de um espectro representável. Este procedimento será abordado com mais detalhes na Seção 3.4.

2.2. *Pipeline* Programável das GPUs

Certamente, um dos maiores avanços para a Computação Gráfica foi a utilização de placas gráficas para acelerar o processamento de elementos gráficos. Estas placas possuem processadores com instruções específicas para tratar estruturas comuns em aplicações 3D, como vértices e vetores. Para isso, foi desenvolvido um *pipeline* específico para este propósito que,

mesmo possuindo variações de implementação, geralmente pode ser dividido em: operações sobre vértices e operações sobre *pixels*. A Figura 4 ilustra como está estruturado um *pipeline* de uma GPU.

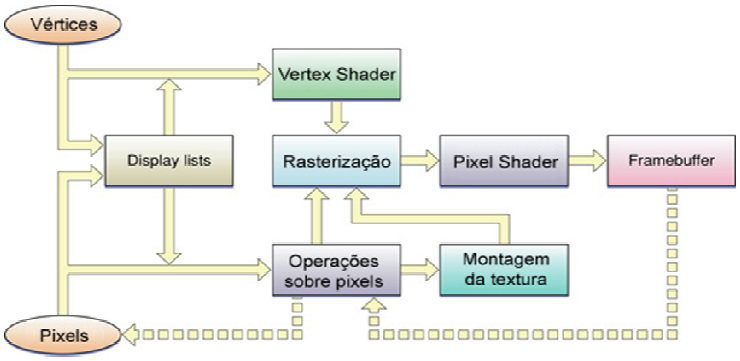

Figura 7. Exemplo de *pipeline* de uma GPU.

Com o avanço das placas, os desenvolvedores decidiram abrir sua arquitetura para possibilitar o desenvolvimento de programas específicos para serem executados nas GPUs. Estes programas foram chamados de *shaders*, e substituem o *pipeline* fixo (*vertex shader* e/ou *pixel shader*) das GPUs para realizar operações personalizadas. Esta flexibilidade, juntamente com o aumento do poder de processamento e da memória das placas, possibilitou que vários algoritmos fossem adaptados para serem executados na GPU e, conseqüentemente, tornando-se viáveis em tempo real.

Atualmente, existem três linguagens de *shader*. *High Level Shader Language* (HLSL) [8] é a linguagem desenvolvida pela Microsoft para ser usada em conjunto com sua biblioteca DirectX. HLSL é a mais utilizada, especialmente pela indústria de jogos, e também foi a linguagem escolhida para ser utilizada neste trabalho devido à facilidade em encontrar uma vasta biblioteca de exemplos, facilitando o aprendizado da linguagem. *OpenGL Shading Language* (GLSL) [9] é a linguagem desenvolvida pelo grupo OpenGL. E, por último, Cg [10] é a linguagem desenvolvida pela NVIDIA como iniciativa para unificar as linguagens HLSL e GLSL. Dessa forma, Cg possui compatibilidade com programas escritos em HLSL e em GLSL.

2.3. Efeitos de Iluminação

Existem vários caminhos pelos quais é possível atacar o problema de realismo em cenas sintetizadas por computador. É possível utilizar uma imagem para iluminar os objetos de forma natural, gerar sombras realistas, melhorar a textura do objeto, entre outras maneiras. A forma abordada neste trabalho foi simular os efeitos que ocorrem no olho humano ou em uma lente de câmera quando uma fonte de luz é observada. Para melhor entendimento do problema, estes efeitos de iluminação foram subdivididos em problemas menores.

O primeiro subproblema a ser abordado é um efeito conhecido como *Bloom* [11]. No mundo real, o *Bloom* ocorre quando um objeto é observado diretamente contra uma fonte de luz e acontece um fenômeno no qual a luz parece ultrapassar os limites da borda do objeto, ofuscando o observador (ver Figura 9a). Isso ocorre porque, ao entrar no olho humano (ou em uma lente), o raio luminoso se espalha, atingindo a retina em vários pontos [12]. Esses pontos, que seriam estimulados pela luminosidade natural do objeto, acabam estimulados pelo raio luminoso, como ilustrado pela Figura 8. Masaki Kawase [11] implementou com sucesso este e outros efeitos de iluminação em tempo real, obtendo um resultado visual realista.

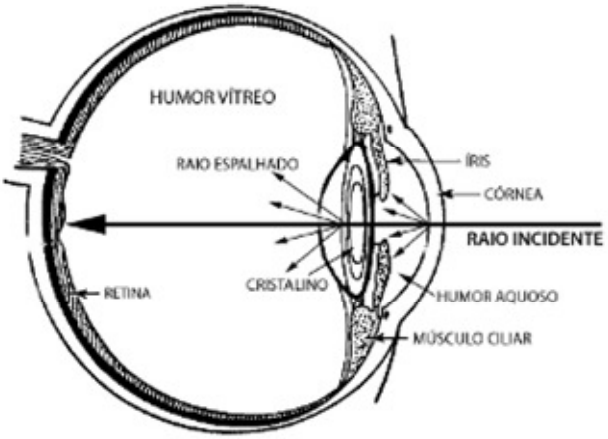

Figura 8. Representação do fenômeno que ocorre no olho humano.

O segundo subproblema ocorre por causa de um fenômeno onde fontes de luz ou reflexões intensas causam um espalhamento dos raios luminosos, principalmente nos olhos humanos. É percebido como um conjunto de arestas radiais em torno da fonte luminosa (ver Figura 9b) [13]. Este efeito é conhecido como *Glare*, e é de conhecimento geral que é causado pela difração e espalhamento em obstáculos perto do olho. Os raios luminosos são difratados primeiramente pelos cílios e algumas vezes pela borda das pálpebras [14]. Vários tipos diferentes de *Glare* podem ser observados em situações do mundo real. Dependendo do tipo da fonte de luz e do objeto que está capturando a imagem (olho, lente da câmera, etc.), os efeitos podem variar entre faixas horizontais, faixas verticais, estelares de quatro, seis ou várias pontas, entre outros. Alguns trabalhos realizados sobre este tópico buscam simular fisicamente o que ocorre no olho humano, como é o caso de Spencer *et al.* [12] e Kakimoto *et al.* [13]. Outros trabalhos procuram apenas aproximar-se do efeito visual real, sem levar em consideração a física que origina o efeito de iluminação, como é o caso de Kawase [11].

Em alguns casos, é possível observar também o surgimento do *Lenticular Halo* [12], um conjunto de círculos coloridos concêntricos, circundando a fonte luminosa.

Além de estes efeitos acontecerem na fonte primária de luz, eles também podem ocorrer em superfícies suficientemente reflexivas. Esta terceira situação foi nomeada como *Dazzling*

Reflection [11]. Quando um objeto reflete o ambiente, parte da intensidade luminosa que chega nele é atenuada, de acordo com o seu coeficiente reflexivo. Utilizando uma imagem HDR para simular fontes de luz, a informação de quão forte é o raio luminoso que atinge o objeto é muito mais precisa. Assim, a imagem refletida pode produzir os mesmos efeitos de ofuscamento mencionados anteriormente (ver Figura 9c).

Finalmente, o quarto e último subproblema é um fenômeno que ocorre naturalmente no olho humano (e é simulado pelas câmeras). Conhecido como *Exposure Control* [11], funciona como um filtro para a luminosidade que chega ao observador. Em cenas predominantemente claras, a tendência é diminuir a quantidade de luz que entra no dispositivo receptor, escurecendo a cena e diminuindo a visibilidade das partes menos iluminadas. Em cenas escuras, a tendência é dar prioridade para os objetos pouco iluminados, e eventuais focos de luminosidade acabam ofuscando o observador. Esse efeito é facilmente observado em situações como a saída de um túnel, onde a luminosidade varia de dentro do túnel para o exterior, e a visão do motorista precisa adaptar-se (veja outro exemplo na Figura 9d). Utilizando imagens HDR, é possível estimar com maior precisão a luminosidade que vêm da cena observada e filtrar as informações para gerar o resultado visual final. Dentro do problema de *Exposure Control*, também existe o trabalho de mapear a faixa de cores presentes em uma imagem HDR dentro dos limites dos monitores. Esse processo é conhecido como *Tone Mapping*, e vários autores estudaram algoritmos para solucionar este problema. Reinhard *et al.* [15] produziram um operador capaz de adaptar-se automaticamente às variações de cena em tempo real, e seu trabalho foi utilizado como base para este Trabalho de Graduação. Para os efeitos *Bloom*, *Glare* e *Dazzling Reflection* (citados acima), o trabalho de Masaki Kawase [11] foi utilizado como base para a implementação da solução proposta por este Trabalho de Graduação.

A Figura 9 mostra exemplos dos quatro efeitos citados neste capítulo e que serão abordados no presente trabalho: *Bloom*, *Glare*, *Dazzling Reflection* e *Exposure Control*.

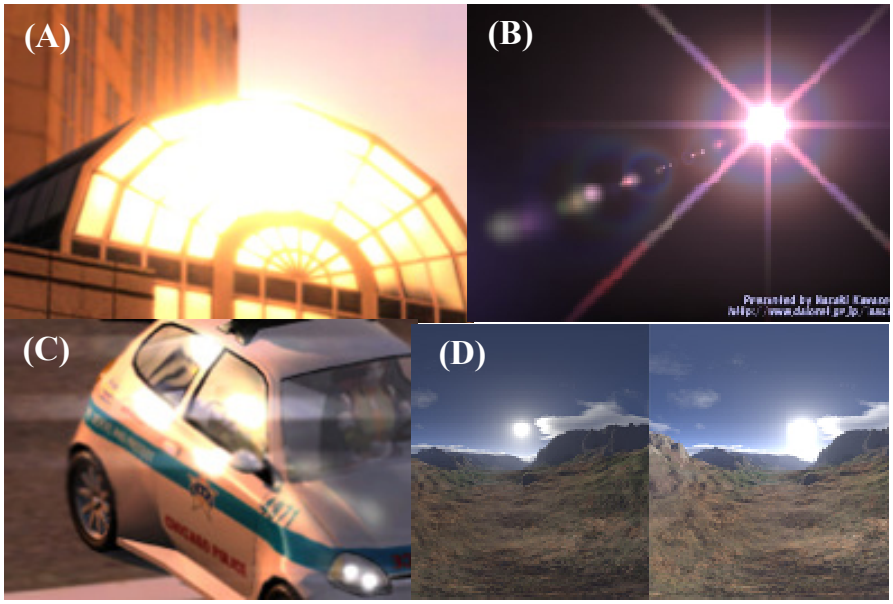

Figura 9. Efeitos de iluminação: (A) *Bloom*; (B) *Glare*; (C) *Dazzling Reflection*; (D) *Exposure Control*.

3. Estudo de Caso: Aplicação Interativa com Efeitos de Iluminação

Neste capítulo, serão explicados os detalhes de implementação de cada um dos efeitos de iluminação citados no capítulo anterior. Ao final, será explicado como estes efeitos foram integrados para funcionarem em conjunto numa aplicação interativa. A aplicação foi organizada em forma de passos, que são etapas de processamento na GPU. Cada passo realiza uma parte do trabalho de cada efeito. Um efeito é composto por um ou mais passos de *shader*.

3.1. *Bloom*

Implementar o efeito físico real do que ocorre no efeito *Bloom* é uma tarefa complexa, embora possível. Para uma aplicação em tempo real, geralmente opta-se por um cálculo mais simples, porém capaz de produzir um efeito visual semelhante.

Neste caso, o método encontrado para simular o efeito *Bloom* consiste em passos de pós-processamento da janela de visualização da aplicação [11]. O termo pós-processamento quer dizer que após toda a geometria da cena ser sintetizada, ela é submetida a procedimentos que irão alterar o resultado visual final.

A primeira etapa do processo é utilizar a informação HDR da cena para extrair áreas de alta luminosidade. Assim, a luminância (L_ω) de cada *pixel* é calculada através da equação:

$$L_\omega = (0{,}2125 \times R) + (0{,}7154 \times G) + (0{,}0721 \times B), \quad (1)$$

onde R, G e B são, respectivamente, os componentes Vermelho, Verde e Azul de cada *pixel*.

A partir de então, é necessário estabelecer um limiar que considera a partir de qual valor os valores de luminosidade serão processados. O resultado deste passo é uma imagem semelhante a uma imagem binarizada (ver Figura 10), onde os valores abaixo do limiar são convertidos para 0 (zero) e os valores acima do limiar mantêm o valor calculado por (1).

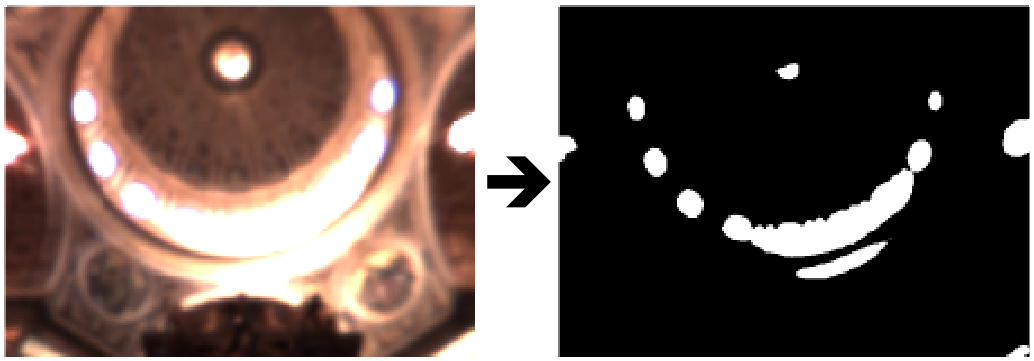

Figura 10. À esquerda, cena original contendo informação HDR; à direita, imagem contendo apenas os pontos com luminosidade acima de um limiar estabelecido.

A imagem calculada no passo anterior é passada como entrada para o próximo passo do algoritmo, que aplicará filtros gaussianos sucessivos na imagem. Aqui, existem duas etapas:

uma passada vertical e uma horizontal. Posteriormente, para um resultado mais suave, o resultado deste passo (horizontal e vertical) é submetido para uma nova função gaussiana, com um declive mais suave. Isto é necessário para simular o círculo gradiente ao redor da fonte luminosa. Entretanto, algumas otimizações podem ser realizadas antes de executar o processo citado. Primeiro a imagem original é sub-amostrada no passo em que a luminosidade de cada pixel é calculada, para uma resolução de 512x512, reduzindo assim o número de operações. Além disso, a função gaussiana é discretizada, como mostra a Figura 11, para simplificar os cálculos.

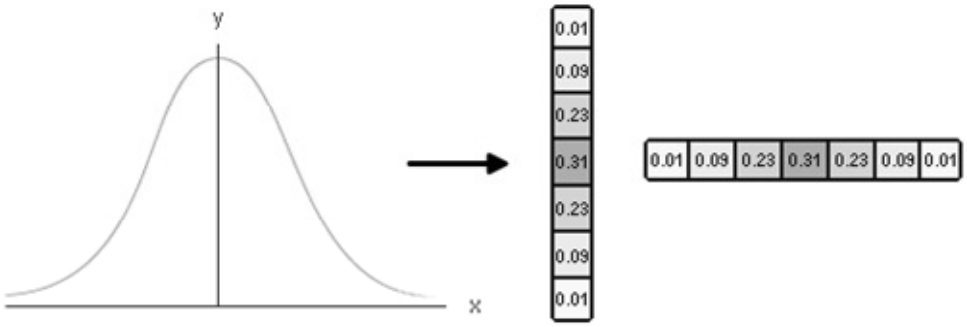

Figura 11. Discretização da função gaussiana em estruturas verticais e horizontais.

Para armazenar uma estrutura que simulasse a queda suave de uma função gaussiana, seria necessário alocar muita memória na placa de vídeo. Em vez de alocar várias funções gaussianas (gradualmente mais suaves), uma técnica de otimização utilizada é sub-amostrar as texturas onde o resultado será sintetizado. Isto produz o efeito gradiente desejado e, utilizando um filtro bilinear, o erro é imperceptível. Isto pode ser observado na Figura 12, onde a linha superior mostra as imagens antes do filtro bilinear, e a linha inferior mostra o resultado da composição. Dessa forma, na aplicação da primeira função gaussiana, a imagem é reduzida para a resolução 256x256. Na aplicação da segunda função gaussiana, a imagem é sub-amostrada para 64x64.

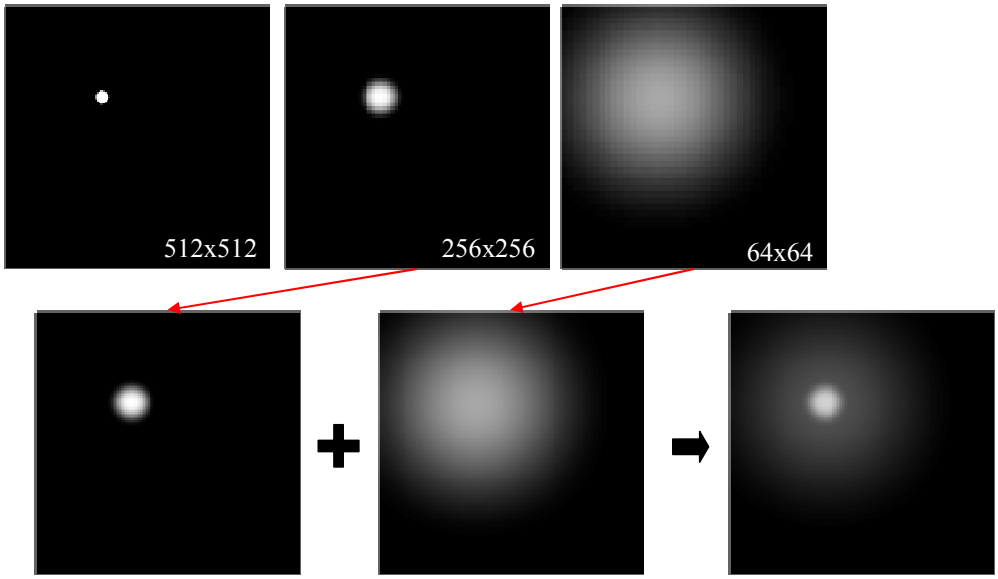

Figura 12. Resultado da aplicação de filtros gaussianos na imagem e o resultado final da composição.

As imagens resultantes de cada passo do filtro gaussiano são posteriormente compostas em um resultado final, e este é adicionado na cena original para simular o efeito *Bloom* desejado.

Em RA, esse efeito é importante para conferir realismo à cena, pois um objeto virtual que não seja afetado pela iluminação do ambiente no qual ele está inserido é notado pelo observador como algo artificial. Em algumas aplicações, é desejado que o usuário não perceba esta diferença.

3.2. *Glare*

Existem duas abordagens principais que buscam implementar o efeito *Glare*: uma que procura simular o resultado físico real e outra que se preocupa apenas com o resultado visual.

Na linha de trabalhos que estudam o comportamento físico, existe o desafio de implementar um algoritmo que seja viável em tempo real, para ser utilizado em aplicações interativas. Uma técnica recente utilizada por Kakimoto *et al.* [13] utiliza a Transformada Rápida de Fourier (FFT) para gerar os raios concêntricos característicos deste efeito. Este algoritmo propõe o uso de imagens de anteparo, que servirão para simular a oclusão que ocorre no olho humano, causado pelos cílios e pela pupila, como observado na Figura 13. Desta forma, é possível computar a distorção que ocorre com os raios luminosos, levando-os para o domínio da freqüência. Para possibilitar a utilização em tempo real, a FFT foi implementada na GPU [16].

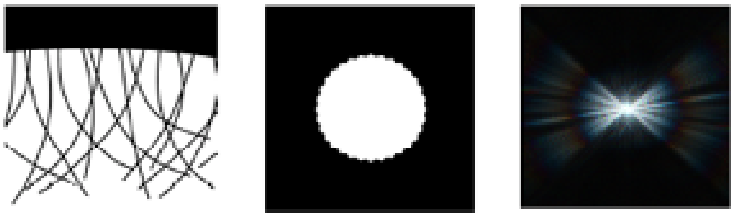

Figura 13. Imagens que simulam a oclusão natural no olho humano; à esquerda uma representação dos cílios; ao centro, uma representação da pupila; à direita o resultado da FFT.

O escopo deste trabalho consiste em reproduzir o efeito visual realista, sem se preocupar com o comportamento físico. Assim, a técnica utilizada para gerar o efeito *Glare* consistiu nos passos de pós-processamento descritos a seguir [11].

Primeiramente, os pontos de alta luminosidade são extraídos da cena. De forma análoga ao primeiro passo do efeito *Bloom*, descrito na subseção anterior, através da equação (1), a luminosidade de cada *pixel* é calculada. A diferença consiste no limiar utilizado, que neste caso é um pouco mais alto para filtrar somente os pontos de luminosidade intensa. A resolução utilizada para as imagens deste efeito foi 128x128.

Em seguida, serão utilizados filtros sucessivos para criar os raios concêntricos. Cada raio será gerado em uma direção específica e será composto de 3 passos no *shader*. Em cada passo, os pontos de luminosidade intensa da cena serão submetidos a um filtro atenuante (ver Figura 14) que gradualmente reproduzirá o efeito do raio luminoso. Assim, para o n-ésimo passo:

- A textura utilizada como entrada no primeiro passo é o resultado da equação (1), e nos passos seguintes é a saída do passo $n-1$ do algoritmo de *Glare*.
- A cor de cada *pixel* será definida por uma estrutura atenuante, descrita pelos seguintes pesos:

$$peso[s] = a^{(b \times s)}, \tag{2}$$

  onde a é o fator atenuante (entre 0,85 e 0,95) que diz o quanto o raio será dissipado com a distância; s é o estágio de amostragem, que diz o quão distante o *pixel* está da fonte luminosa, e b é um fator multiplicador que amplia o fator atenuante e é definido pela seguinte equação:

$$b = 4^{(n-1)}, \tag{3}$$

  onde n é o n-ésimo passo do algoritmo.
- O ponto onde o *shader* estará computando a cor será influenciado pelos seus *pixels* vizinhos, de acordo com a equação:

$$TexCoord[s] = (pto.x + (b \times s), pto.y + (b \times s)), \tag{4}$$

onde pto é o ponto atual onde o *shader* está sintetizando a imagem; b é o raio de vizinhança para pontos luminosos, definido pela equação (3), e s é o estágio de amostragem. Assim, a cada passo, o valor de b é atualizado, e o ponto a ser sintetizado será influenciado por *pixels* cada vez mais distantes.

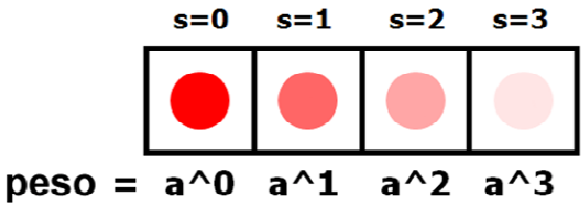

Figura 14. Exemplo de filtro atenuante do efeito *Glare* para o 1º passo ($\boldsymbol{b = 1}$); $\boldsymbol{s}$ são os estágios de amostragem, e $\boldsymbol{a}$ o fator atenuante.

Dessa forma, ao final do terceiro passo, o resultado final será uma imagem que contém um raio luminoso proveniente do efeito *Glare*. Porém, para produzir um efeito visualmente realista, é necessário variar o raio luminoso em 2, 4, 6 ou mais direções. Assim, este procedimento precisa ser repetido quantas vezes forem necessárias, de forma a gerar o número de raios desejado. Para isso, basta modificar a equação (4) para variar a direção na qual o raio é gerado. A Figura 15 ilustra o processo de geração dos raios do efeito *Glare* descrito acima.

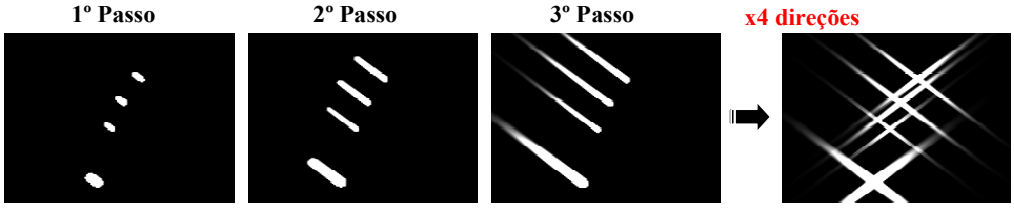

Figura 15. Para cada direção, o raio luminoso é gerado executando-se 3 passos; para obter o resultado realista desejado, basta variar o procedimento para as demais direções.

A imagem final gerada é então adicionada à cena original, para gerar o efeito desejado. É importante observar que a inclusão de mais raios luminosos aumenta também o volume de processamento do algoritmo. Dependendo da configuração utilizada, a aplicação pode se tornar inviável em tempo real. Este tópico será abordado com mais detalhes no Capítulo 4.

3.3. *Dazzling Reflection*

Para gerar o efeito de *Dazzling Reflection*, é necessário mapear o ambiente HDR na superfície dos objetos inseridos na cena [17]. Este procedimento é semelhante a traçar raios que atinjam a superfície do objeto e reflitam na direção do observador.

Dessa forma, como a imagem HDR do ambiente é diretamente mapeada nos objetos, a informação de iluminação contida nelas também é preservada. Conseqüentemente, é possível

estimar a luminosidade oriunda da reflexão, e aplicar os mesmos efeitos causados pela fonte de luz original.

Para simular este efeito, é necessário extrair a luminância de cada *pixel* do objeto através da equação (1). Isto é feito naturalmente quando a cena é observada como um todo, onde o ambiente e objetos estão presentes na janela de visualização. Assim, ao tratar do efeito presente no ambiente, os valores refletidos nas superfícies dos objetos também são tratados.

Ao utilizar o efeito *Dazzling Reflection*, é possível variar os efeitos Bloom e Glare decorrentes da reflexão para produzir resultados diferentes da fonte de luz original. Neste Trabalho de Graduação, apenas uma pequena variação foi inserida no efeito gerado na reflexão, que consiste na intensidade com a qual a luminosidade proveniente do ambiente é refletida. A cor dos objetos inseridos na cena foi definida através de duas constantes: um coeficiente difuso (Od) e um coeficiente especular (Ks). Od define diretamente a cor difusa da superfície do objeto, ou seja, usando o valor 0,0 (zero) o resultado é um objeto completamente escuro e usando o valor 1,0 (um) o resultado é um objeto completamente branco (embora as regiões não iluminadas possam aparentar mais escuras). Ks define o percentual de luz refletida proveniente do ambiente, ou seja, um valor 0,0 (zero) representa uma superfície que não reflete nada do ambiente, enquanto um valor 1,0 (um) é uma superfície totalmente reflexiva (espelhada). A cor final do objeto é definida então pela equação:

$$corObj = Od + (mapaAmbiente \times Ks), \tag{5}$$

onde $mapaAmbiente$ representa a cor proveniente da imagem HDR do ambiente.

Utilizando este algoritmo, é possível amenizar o efeito de *Dazzling Reflection*, apenas variando as constantes citadas.

3.4. *Exposure Control*

Para gerar o efeito conhecido como *Exposure Control*, primeiramente é utilizado um algoritmo que calcula a luminância global da cena, conhecido como *Tone Mapping*. O problema de *Tone Mapping* pode ser definido como mapear a luminância medida/simulada da cena na luminância da tela e produzir uma imagem satisfatória [15]. O algoritmo consiste nos passos abaixo.

Como primeiro passo, é necessário varrer a imagem (janela de visualização da cena) para estimar a média logarítmica da luminância global ($\bar{L}_\omega$), calculada por:

$$\bar{L}_\omega = \frac{1}{N} exp\left(\sum_{x,y} log\left(\delta + \frac{L_\omega(x,y)}{maxLum}\right)\right), \tag{6}$$

onde $L_\omega(x,y)$ é a luminância de cada *pixel*, definida por (1), N é o número total de *pixels* da imagem, δ é uma constante pequena para impedir que a equação seja anulada caso exista um *pixel* preto na imagem, e $maxLum$ é o valor mais alto de luminância presente na imagem.

Normalmente, a imagem é sub-amostrada antes de ser submetida a esta fórmula, por questões de otimização. O valor utilizado neste trabalho foi 64x64. O valor final de $\bar{L}_\omega$ é escrito em uma textura 1x1 para ser enviada para o próximo passo no *shader*.

No segundo passo, é necessário calcular a luminância escalada (L), definida por:

$$L(x,y) = \frac{a}{\bar{L}_\omega} L_\omega(x,y), \tag{7}$$

onde a é uma constante chamada valor-chave, que está relacionada com o quão clara ou escura é a cena original e como ela vai parecer no final do algoritmo. Geralmente, esta constante assume valores entre 0,18 e 0,72 [15].

Finalmente, o valor da luminância final é calculado através da equação:

$$L_d(x,y) = \frac{L(x,y)}{1+L(x,y)}. \tag{8}$$

É importante observar que L_d é calculado para cada *pixel* (x,y) da imagem. O valor final é então multiplicado por seu *pixel* correspondente, para chegar à cor final resultante. Este algoritmo visa alterar o valor de todos os *pixels* da imagem, diminuindo a sua luminosidade, se necessário. Isto provoca um efeito que, ao mover a câmera para áreas da cena que contêm muitas áreas luminosas, a cena é escurecida, simulando o efeito *Exposure Control*. Um exemplo de cena pode ser observado na Figura 16, onde ao mover a câmera na direção do céu, a luminosidade média aumenta e a cena é escurecida.

Figura 16. Exemplo de cena onde o efeito *Exposure Control* pode ser observado.

3.5. Solução Integrada

Para o desenvolvimento da aplicação interativa, foi utilizado o Microsoft Visual Studio 2005 [18]. A linguagem de programação escolhida foi C++. Para simplificar o desenvolvimento dos elementos gráficos da aplicação, foi utilizada uma biblioteca de abstração gráfica chamada

OGRE 3D [6]. Esta biblioteca permite a criação rápida de vários elementos característicos de aplicações tridimensionais, e abstrai a biblioteca gráfica utilizada. Dessa forma, foi possível concentrar os esforços de desenvolvimento no problema dos efeitos de iluminação.

Para desenvolver os programas em *shader*, existem duas ferramentas mais utilizadas: FX Composer [19], desenvolvido pela NVIDIA, e RenderMonkey [20], desenvolvido pela ATI. Ambas as ferramentas estão em estado inicial de desenvolvimento, e não possuem vários recursos disponíveis em ferramentas de desenvolvimento para outras linguagens. Entretanto, já representam um avanço onde antes não existia nenhum auxílio. Depois de algum tempo experimentando as duas ferramentas mencionadas, o RenderMonkey foi escolhido como melhor opção para este Trabalho de Graduação, devido à forma como ele organiza os passos do *shader*. Para configurar uma cena de testes dentro da ferramenta, enquanto o FX Composer utiliza trechos de código de configuração em meio ao código referente ao *shader*, o RenderMonkey separa a parte de configuração da cena e a parte do programa em *shader* de forma mais clara, através de uma estrutura em árvore. Esse fator foi decisivo para a escolha do RenderMonkey para esta solução.

O OGRE suporta o carregamento direto de *shaders*, através de *scripts* chamados *Compositors* e *Materials*. Utilizando estes *scripts*, foi possível integrar cada programa (correspondente aos efeitos citados nas subseções anteriores) em uma solução unificada.

A primeira etapa para criar o aplicativo foi gerar uma estrutura para representar o ambiente. A estrutura escolhida foi o *skydome*, um objeto tridimensional semelhante a um hemisfério que envolve todo o ambiente. A textura deste objeto é o mapa esférico HDR, e para mapear as coordenadas tridimensionais do *skydome* na imagem bidimensional foi utilizada a equação citada na Seção 2.1.

Em seguida, foi criado um objeto tridimensional para servir de estudo para os efeitos de iluminação. O objeto escolhido foi uma cabeça de ogro, já presente na biblioteca OGRE. Este objeto deveria ser capaz de gerar o efeito *Dazzling Reflection*, então um *script* correspondente à textura do objeto foi criado, implementando o *shader* respectivo a este efeito. Por isso, a execução deste efeito acontece no momento da síntese do objeto, ao contrário dos outros efeitos que ocorrem no pós-processamento.

Diante deste fato, os outros efeitos foram organizados na seguinte ordem:

- Primeiramente, a cena é sintetizada normalmente. Como todos os efeitos (exceto o *Dazzling Reflection*) são criados no pós-processamento, a cena original é gerada, para que os efeitos sejam posteriormente sobrepostos a ela.
- Um passo do *shader* calcula a média logarítmica da luminância global, necessária para o efeito *Exposure Control*. Este resultado é guardado em uma textura 1x1.
- Em outro passo, os valores de alta luminosidade são calculados em uma textura sub-amostrada para 512x512. Este é o primeiro passo necessário para o efeito *Bloom*.

- Em seguida, um outro passo do *shader* aplica o filtro gaussiano horizontalmente na imagem do passo anterior, sub-amostrada para 256x256.
- Outro passo é necessário para aplicar o filtro gaussiano verticalmente.
- Em seguida, a imagem é sub-amostrada novamente (para 64x64) para a aplicação do filtro gaussiano horizontalmente, pela segunda vez. Como explicado anteriormente, esse passo simula o efeito de utilizar um filtro gaussiano mais suave, ganhando no desempenho.
- Finalmente, outro passo é necessário para aplicar o filtro gaussiano verticalmente na imagem sub-amostrada em 64x64, finalizando o efeito *Bloom*. O resultado final é guardado em duas texturas (uma 256x256 e outra 64x64) para utilização posterior.
- Agora, um passo é necessário para extrair os pontos de luminosidade intensa, necessário para o efeito *Glare*. A imagem é salva em uma textura 256x256.
- Em seguida, começam os passos para gerar o efeito *Glare*. Em texturas de 128x128, são executados os três primeiros passos para gerar o raio luminoso na direção Noroeste.
- Depois, são executados mais três passos consecutivos para gerar o raio luminoso na direção Nordeste.
- Depois, são executados mais três passos para gerar o raio na direção Sudeste.
- Finalmente, são executados os três últimos passos para gerar o raio na direção Sudoeste.
- No passo final do efeito *Glare*, é necessário mais um passo do *shader* para realizar a composição das quatro direções do efeito.
- Como etapa final, é necessário um passo final do *shader*, que receberá como entrada a textura proveniente da cena original, as duas texturas do efeito *Bloom*, a textura para finalizar o efeito *Exposure Control* e a textura final do efeito *Glare*. Este passo realiza os cálculos finais referentes ao efeito *Exposure Control* e em seguida compõe as texturas para gerar o resultado final que será mostrado na tela, de acordo com a equação:

$$corFinal = (cor \times lum) + \big((bloom1 + bloom2) \times 0{,}2\big) + (glare \times 0{,}2), \qquad (9)$$

onde cor é a cor da cena original, lum é o valor calculado da luminância final, $bloom1$ e $bloom2$ são as texturas provenientes do efeito *Bloom* e $glare$ é a textura proveniente do efeito *Glare*.

A organização deste procedimento pode ser visualizada na Figura 17, abaixo.

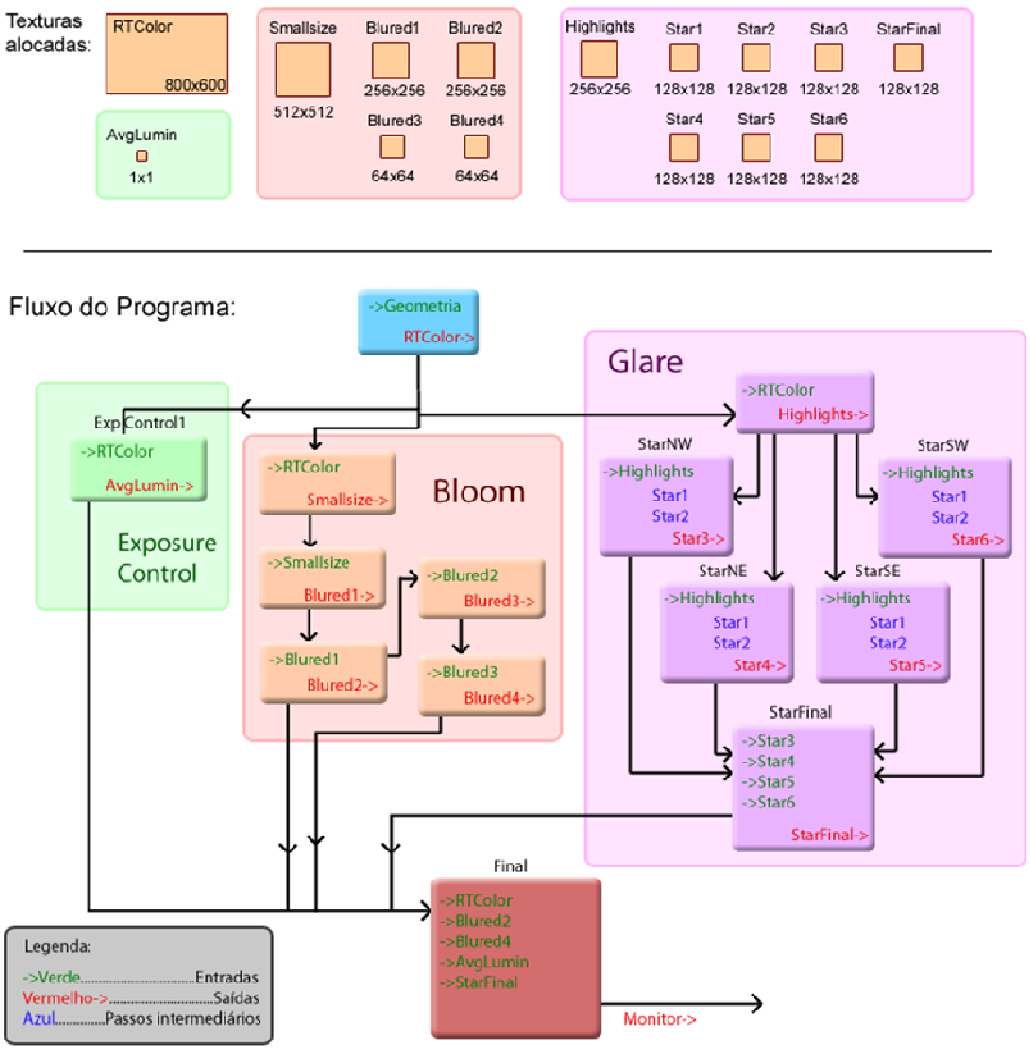

Figura 17. Organização do *shader*: acima, as texturas utilizadas; abaixo, o fluxo do programa, destacando os três efeitos: *Exposure Control*, *Bloom* e *Glare*.

3.6. Dificuldades Encontradas

Durante o desenvolvimento deste trabalho, várias dificuldades foram encontradas no estudo das tecnologias envolvidas. Primeiramente, foi necessário um estudo prévio sobre a linguagem de *shaders*, para entender o paradigma de programação para o *pipeline* das placas gráficas. A mudança de paradigma para o processo em paralelo característico das GPUs levou a um início lento no desenvolvimento. Ao mesmo tempo, os programas de apoio ao desenvolvimento (IDE) disponíveis para *shaders* ainda estão em fase inicial, com poucas funcionalidades e muitas vezes contendo problemas. No caso da ferramenta escolhida (RenderMonkey [20]), não era possível definir variáveis de entrada para os *shaders*, resultando em códigos menos flexíveis antes da transferência dos programas para o OGRE. Além disso, a IDE não suporta carregamento de arquivos externos, ou seja, não era possível escrever uma função comum entre vários passos do

shader, e carregar o arquivo para cada programa. Isto resultou em código replicado, e maior atenção para alterações de forma a não causar inconsistências.

Outro problema encontrado foi na utilização de mapas esféricos para representação do ambiente HDR. Ao utilizar estes mapas, a informação contida nas bordas da imagem acaba sendo fortemente distorcida, resultando em perda de dados. Isto pode ser visualizado na aplicação final como um ponto do mapa que mostra estas distorções (ver Figura 18), por não conseguir representar com fidelidade a informação do ambiente.

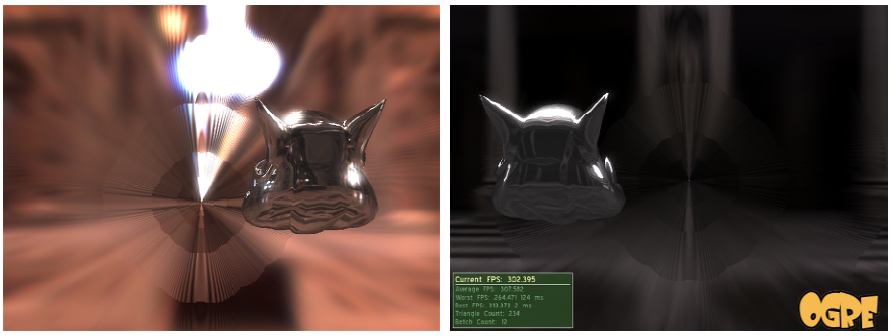

Figura 18. Distorção do ambiente causada pela perda de informação nas bordas do mapa esférico.

Além disso, durante a implementação do efeito *Glare*, foi cogitada a possibilidade de implementar o algoritmo proposto por Kakimoto [13], que leva em consideração o comportamento físico dos raios luminosos no olho humano. Porém, devido ao tempo disponível e à complexidade do problema, não foi possível implementar o passo de adaptar a FFT para execução na GPU. Dessa forma, foi escolhido o algoritmo proposto por Kawase [11] para simular os raios luminosos, que não leva em consideração o comportamento físico.

4. Resultados

O aplicativo desenvolvido neste Trabalho de Graduação visou melhorar o realismo de cenas sintetizadas por computador através da simulação de efeitos de iluminação causados por fontes de luz. Analisando visualmente os resultados do aplicativo, pode-se notar que eles são satisfatórios para a utilização em jogos eletrônicos e aplicações de RV e RA.

Para os testes de desempenho, foi utilizada uma máquina com a seguinte configuração técnica: AMD Atlhon 64 X2 4800+ a 3,0GHz, NVIDIA GeForce 8800 GTX com 768MB VRAM e 1GB de memória RAM. A resolução de tela utilizada em todos os testes foi 800x600. Como a placa de vídeo utilizada foi um modelo avançado, o desempenho da aplicação foi mantido em níveis ótimos. Com todos os efeitos ativos simultaneamente, a média de desempenho foi em torno de 310 quadros por segundo (FPS). Utilizando uma placa mais simples (NVIDIA Quadro FX 3450 com 256MB VRAM) o desempenho da aplicação caiu para 30 FPS, mantendo ainda a interatividade com o usuário. As imagens de resultados podem ser vistas a seguir.

A Figura 19 demonstra os resultados do efeito *Bloom*. Foram utilizados vários ambientes HDR para ilustrar a melhor o efeito. É possível observar que, ao ficar contra a luz, a silhueta do ogro acaba sendo sobreposta pela luz proveniente do ambiente.

Figura 19. Resultados do efeito *Bloom* em vários ambientes HDR. A taxa de desempenho média ficou em torno de 740 FPS.

A Figura 20 demonstra os resultados do efeito *Glare*. É possível observar que apenas os pontos de luminosidade mais intensa provocam o surgimento do efeito na cena. Foram utilizados raios luminosos em quatro direções para simular este efeito.

Figura 20. Resultados do efeito *Glare* em vários ambientes HDR. A taxa de desempenho média ficou em torno de 450 FPS.

A Figura 21 demonstra os resultados do efeito *Dazzling Reflection*. É possível observar que a superfície do ogro reflete o ambiente HDR, provocando os mesmos efeitos de *Bloom* e *Glare*, porém com menor intensidade. É possível controlar o coeficiente reflexivo da superfície alterando as propriedades do objeto.

Figura 21. Resultados do efeito *Dazzling Reflection* em vários ambientes HDR. A taxa de desempenho média ficou em torno de 370 FPS.

A Figura 22 demonstra os resultados do efeito *Exposure Control*.Nas imagens da esquerda a cena foi observada de um ponto onde a luminosidade não é muito forte. Nas imagens da direita, a cena foi observada de um ponto em que a luminosidade atinge o observador com maior intensidade. É possível observar que nas imagens da direita, todos os pontos da cena foram ligeiramente escurecidos, simulando o controle de exposição natural.

Figura 22. Resultados do efeito *Exposure Control*. As imagens que estão lado a lado são referentes ao mesmo ambiente, porém com níveis de exposição diferente. O desempenho ficou em torno de 390 FPS.

A Figura 23 ilustra todos os efeitos implementados simultaneamente. É possível observar um objeto que interage com a iluminação do ambiente aumenta a sensação de realismo e a integração de todos os efeitos de iluminação faz com que o objeto pareça naturalmente inserido na cena.

Figura 23. Todos os efeitos integrados. As imagens lado a lado mostram o efeito *Exposure Control*. A taxa de desempenho média ficou em torno de 310 FPS.

5. Conclusão

Até algum tempo atrás, aplicações gráficas precisavam ser bastante simplificadas para possibilitar a sua execução em tempo real, ou pelo menos em taxas que possibilitassem a interação do usuário. Com o avanço das placas gráficas, as simulações puderam ser melhoradas para alcançar níveis de realismo sem precedentes.

O objetivo deste Trabalho de Graduação foi desenvolver um aplicativo capaz de simular alguns efeitos de iluminação presentes no mundo real e observar o ganho de realismo visual no resultado final.

Observando os resultados obtidos, é possível concluir que os efeitos gerados foram capazes de melhorar o realismo da cena e a integração entre o objeto e o ambiente. Também é possível concluir que, de acordo com os resultados de desempenho obtidos, esta solução pode ser utilizada em jogos eletrônicos e aplicações de RV e RA, de forma a melhorar a experiência visual do usuário.

No caso particular de RA, é interessante ressaltar a importância de o objeto virtual ser integrado com o ambiente que o circunda, para atingir o objetivo de mesclar objetos sintéticos com objetos reais da cena, sem que o usuário perceba claramente a diferença entre os dois mundos.

Como contribuições deste trabalho podem-se mencionar o estudo realizado sobre a simulação de efeitos de iluminação realistas e suas respectivas implementações diretamente no *pipeline* da placa gráfica. Também foi possível comprovar que mesmo utilizando estes efeitos, o desempenho obtido ainda é adequado para a utilização destes algoritmos em aplicações inerentemente mais complexas, como jogos eletrônicos, RV e RA. Além disso, este trabalho deu origem a uma publicação no Workshop de Realidade Virtual e Aumentada em 2007 e uma submissão sob avaliação ao X Symposium on Virtual and Augmented Reality em 2008.

Cabe ressaltar que o presente Trabalho de Graduação está inserido no contexto de um projeto chamado *Real-Time Photorealistic Rendering of Synthetic Objects into Real Scenes* (RPR-SORS), que tem por objetivo o desenvolvimento de uma solução capaz de sintetizar em tempo real objetos virtuais em cenas reais o mais realisticamente possível, isto é, eles devem se parecer com objetos do mundo real.

5.1. Trabalhos Futuros

Embora o trabalho tenha sido concluído, existem várias melhorias que podem ser feitas para tornar o resultado da solução aperfeiçoado. A maioria das melhorias sugeridas diz respeito ao resultado visual, que embora seja satisfatório, pode ser aperfeiçoado em cada cena.

O primeiro trabalho a ser mencionado é referente ao erro causado pela utilização de mapas esféricos. Neste caso, a proposta é dar suporte também à utilização de mapas cúbicos, para verificar se o erro é corrigido e qual a sua relação com o desempenho da aplicação.

Em relação ao efeito *Bloom*, existe a possibilidade de estudar a queda de desempenho ao utilizar mais passos de suavização, para obter um resultado mais natural.

Outro trabalho a ser realizado é em relação ao efeito *Glare*, onde é necessário um estudo minucioso sobre a adição de raios em mais direções e sua relação com o desempenho da aplicação. Outra possibilidade é dar continuidade ao estudo sobre o efeito físico através da FFT.

Em relação ao efeito *Dazzling Reflection*, um trabalho a ser realizado é o estudo de novas variações do efeito, possivelmente alterando a forma do *Glare* gerado pela reflexão.

Finalmente, em relação ao efeito *Exposure Control*, é necessário otimizar o código implementado para que ele seja mais flexível aos diferentes ambientes HDR. Atualmente, é necessário definir manualmente o valor da luminosidade máxima, porém é provável que esse cálculo possa ser realizado automaticamente. Além disso, é necessário implementar um atraso para o tempo de adaptação da cena. Atualmente, a adaptação ocorre instantaneamente, porém para um efeito natural, o ideal é que a abertura da lente seja adaptada gradualmente.

Referências Bibliográficas

[1] Ferwerda, J.A. Three Varieties of Realism in Computer Graphics. In: SPIE Human Vision and Electronic Imaging, 2003, pp. 146-158.

[2] World of Warcraft. Disponível: World of Warcraft Community site. URL: http://www.worldofwarcraft.com/, visitado em 8 de Janeiro de 2008.

[3] Half-Life 2. Disponível: Valve site. URL: http://www.valvesoftware.com/games.html, visitado em 8 de Janeiro de 2008.

[4] Fournier, A.; Gunawn, A.S.; Romanzin, C. Common Illumination between Real and Computer Generated Scenes. In: GI, 1993, pp. 254-262.

[5] OGRE. Disponível: OGRE 3D: Open source graphics engine site. URL: http://www.ogre3d.org/, visitado em 8 de Janeiro de 2008.

[6] Debevec, P.E.; Malik, J. Recovering High Dynamic Range Radiance Maps from Photographs. In: SIGGRAPH, 1997, pp. 369-378.

[7] Haeberli, P.; Segal, M. Texture Mapping as a Fundamental Drawing Primitive. In: EG Workshop in Rendering, 1993, pp. 259-266.

[8] HLSL. Disponível: DirectX Developer Center site. URL: http://msdn2.microsoft.com/en-us/library/bb509561.aspx, visitado em 10 de Janeiro de 2008.

[9] GLSL. Disponível: OpenGL Shading Language site. URL: http://www.opengl.org/documentation/glsl/, visitado em 10 de Janeiro de 2008.

[10] Cg. Disponível: NVIDIA Developer Zone site. URL: http://developer.nvidia.com/page/cg_main.html, visitado em 10 de Janeiro de 2008.

[11] Kawase, M. Practical Implementation of High Dynamic Range Rendering. In: GDC, 2004. Disponível: Masaki Kawase's Home Page. URL: http://www.daionet.gr.jp/~masa/, visitado em 11 de Janeiro de 2008.

[12] Spencer, G.; Shirley, P.; Zimmerman, K.; Greenberg, D.P. Physically-based Glare Effects for Digital Images. In: SIGGRAPH, 1995, pp. 325-334.

[13] Kakimoto, M.; Matsuoka, K.; Nishita, T.; Naemura, T.; Harashima, H. Glare Generation Based on Wave Optics, Computer Graphics Forum, vol. 24, no. 22, 2005, pp. 185-193.

[14] Minnaert, M.G.J. (Translated and revised by Seymour L.). Light and Color in the Outdoors. New York: Springer-Verlag, 1993. 417p.

[15] Reinhard, E.; Stark, M.; Shirley, P.; Ferwerda, J. Photographic Tone Reproduction for Digital Images. In: Annual Conference on Computer Graphics and interactive Techniques, ACM, New York-USA, 2002, pp. 267-276.

[16] Moreland, K; Angel, E. The FFT on a GPU. In: Graphics Hardware, 2003, pp. 112-119.

[17] Realistic Natural Effect Rendering. Disponível: GameDev.net site. URL: http://www.gamedev.net/reference/articles/article2138.asp, visitado em 18 de Janeiro de 2008.

[18] Visual Studio 2005. Disponível: Visual Studio site. URL: http://msdn2.microsoft.com/en-gb/vstudio/default.aspx, visitado em 18 de Janeiro de 2008.

[19] FX Composer. Disponível: NVIDIA Developer Zone site. URL: http://developer.nvidia.com/object/fx_composer_home.html, visitado em 18 de Janeiro de 2008.

[20] RenderMonkey. Disponível: ATI Developer site. URL: http://ati.amd.com/developer/rendermonkey/index.html, visitado em 18 de Janeiro de 2008.