Universidade Federal Fluminense

Felipe Pedrosa Martinez

Método para Distribuição Não Uniforme de Raios em Arquitetura Paralela de Traçado de Raios em Tempo Real na GPU

NITERÓI

2014

Universidade Federal Fluminense

Felipe Pedrosa Martinez

Método para Distribuição Não Uniforme de Raios em Arquitetura Paralela de Traçado de Raios em Tempo Real na GPU

Dissertação de Mestrado submetida ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do título de Mestre. Área de concentração: Computação Visual.

Orientador:
Prof. D.Sc. Leandro Augusto Frata Fernandes

NITERÓI

2014

Método para Distribuição Não Uniforme de Raios em Arquitetura Paralela de Traçado de Raios em Tempo Real na GPU

Felipe Pedrosa Martinez

Dissertação de Mestrado submetida ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do título de Mestre em Informática.

Aprovado por:

Prof. D.Sc. Leandro Augusto Frata Fernandes / IC-UFF
(Orientador)

Prof. D.Sc. Esteban Walter Gonzalez Clua / IC-UFF

Prof. Ph.D. Diego Fernandes Nehab / IMPA

Prof. Ph.D. Rodrigo Penteado Ribeiro de Toledo / DCC-UFRJ

Niterói, 01 de Setembro de 2014.

Dedicado à minha família e amigos.

Agradecimentos

Este trabalho foi fruto de uma árdua jornada tanto profissional quanto pessoal. Um tempo de mudanças onde aprendi bastante sobre mim mesmo, sobre minhas dificuldades e sobre minha motivação e força de vontade para superar obstáculos.

Certamente, não trilhei por este caminho sozinho e gostaria de não só agradecer, mas também dedicar este trabalho à minha família e meus amigos próximos que estiveram comigo nestes últimos anos.

Agradeço aos meus pais Manolo e Geisa por terem plena confiança na minha capacidade e no meu sucesso, mesmo quando eu os coloquei em dúvida. Agradeço ao meu irmão Vitor, por ser um grande motivador dessa conquista também. Agradeço aos meus amigos que estando perto ou longe, sempre me deram forças nos momentos onde o cansaço por vezes falou mais alto. Agradeço também por todas as vezes que precisei distrair minha cabeça e tive um ombro amigo pra me fazer companhia. Agradeço ao meu orientador Leandro Fernandes, por ter acreditado neste ideia e ter sabido me motivar durante todo esse processo. Tenho bastante orgulho de ter trabalhado com um profissional como ele, que foi um exemplo de professor, orientador e pessoa, tanto nos momentos calmos quanto nos mais tempestuosos dessa jornada. Agradeço ao professor Esteban Clua por ter me motivado a seguir por este caminho desde que eu decidi optar pelo mestrado no final da minha graduação. Em tempo, agradeço também aos professores Rodrigo de Toledo e Adriano Cruz, por terem me proporcionado as cartas de recomendação necessárias para tal, e não só terem acreditado no meu potencial desde o ínicio como também terem expressado a vontade de acompanhar os meus próximos passos na academia.

Em tempo, agradeço à NVIDIA pelo apoio às pesquisas conduzidas no medialab, tal como ao colega Thales Sabino responsável pelo desenvolvimento do arcabouço que inspirou as ideias apresentadas neste trabalho. Agradeço também à Universidade Federal Fluminense e ao Cnpq pelo suporte e incentivo durante o desenvolvimento deste trabalho.

Enfim, a todos os envolvidos direta e indiretamente a essa conquista, obrigado!

Resumo

Na área de computação gráfica, a demanda por sistemas que sintetizam imagens fieis ao mundo real é sempre crescente, em especial com o desenvolvimento da indústria de jogos e a aplicação das técnicas na indústria de filmes. Até então, no que se refere à síntese de imagem em aplicações em tempo real, as técnicas de rasterização são as mais utilizadas em função do seu custo computacional. Porém, com o desenvolvimento de técnicas de traçado de raios cada vez melhores, é de comum opinião que eventualmente se atinja um ponto onde elas sejam aplicáveis de forma otimizada e, assim, substituam as anteriores, em um cenário onde consigam unir a sua alta qualidade visual com um desempenho adequado. Atualmente, mesmo os traçadores de raio em estado da arte encontram dificuldades em se consolidar completamente em cenários de tempo real. Isso acontece pois geralmente a quantidade de informação a ser processada para atingir os níveis de qualidade de imagem esperados é consideravelmente maior do que se consegue processar dentro das restrições de tempo. Este é um problema constante, já que conforme novas técnicas de aceleração são propostas, cresce também a demanda por maior qualidade de imagem.

Naturalmente, respeitando as exigências de tempo de cada aplicação, não é toda a cena que pode ser processada, sendo elas limitadas por uma série de características: número de objetos, tipos de materiais, fontes de luz, etc. Cada uma destas características tem um custo computacional associado, tal como um impacto visual distinto na imagem final. Sendo assim, caso seja possível distribuir proporcionalmente o esforço para sintetizar cada um desses elementos, priorizando os mais representativos, uma margem maior de cenas se tornaria viável dentro das mesmas restrições de processamento. Infelizmente, a distribuição não uniforme de processamento em arquiteturas paralelas não é trivial, e endereçar cada *thread* do *hardware* gráfico (GPU) a uma região específica da cena pode trazer uma carga computacional extra capaz de inviabilizar a sua execução em tempo real.

Este trabalho apresenta um *pipeline* que viabiliza a distribuição não uniforme de raios primários em aplicações de traçado de raios em tempo real, capaz de organizar os recursos disponíveis para a síntese de um quadro da aplicação, considerando as particularidades da cena a ser processada. Para tal, uma série de estruturas deve obter informações relacionadas à importância de cada região da cena em questão e traduzi-las em termos de recursos de processamento do algoritmo de maneira eficiente e compatível com a arquitetura paralela da GPU, de forma a explorar o caráter paralelo do algoritmo e adequá-lo às restrições de tempo do ambiente interativo. Além disso, este trabalho também propõe duas heurísticas originais compatíveis com a técnica proposta para ilustrar à sua adaptabilidade às diferentes aplicações.

Palavras-chave: traçado de raio, importância, sistemas em tempo real, amostragem, distribuição não uniforme, heurísticas.

Abstract

Since the very first computer graphics studies, the need for systems that could accurately simulate scenes from the real world have been increasing, particularly with the gaming and movies industry development. So far, the real time rendering area have been dominated by rasterization approaches, but with the development and optimization of the ray tracing techniques, we may reach a point when they will be fully consolidated in the area, reaching suitable performance for real time application and thus, replace the latter, maintaining the superior visual quality result. However, nowadays, even the state-of-the-art ray tracers find it difficult to operate fully in real time systems. Usually, the amount of information that needs to be processed to reach the expected quality level is substantially greater than the render capability within the time constraints, and as newer and faster techniques are being proposed, more and more image quality are expected by the growing industry.

Naturally, based on the time constraints for each application, the ray tracer can only render a fixed number of scenes, limited by features like number of objects, kinds of materials, number of light sources etc. Although some features are much more distinctive than others in a scene, they also represent a much bigger processing load the application, and thus, it may be a good ideia to schedule the application resources accordingly, prioritizing the most representative sections of the scene, instead of distributing the limited resources evenly throughout the enviroment. Unfortunately, the non-uniform processing distribution in parallel architectures is not a simple task, and assigning each processing thread of the graphic hardware (GPU) to a specific region of the scene can add and extra workload, making its use in real time applications unfeasable.

This work presents a pipeline that allows a non-uniform primary ray distribution in real time ray tracing applications that is capable of organize the available resources to render a scene based on its particularities. For this purpose, a series of structures were created in order to obtain importance related information for each region of the scene and translate it in a way that the GPU understands as processing resources. These structures are designed to run smoothly in the GPU and take full advantage of its parallel processing capabilities, so that they can be better fit to real time systems. This work also provides two original heuristic functions consistent with the proposed technique to illustrate its adapatability in different applications.

Keywords: ray tracing, importance, real-time systems, sampling, non-uniform distribution, heuristics.

Sumário

Lista de Figuras **viii**

1 Introdução **1**

1.1 Ideia Central 4

1.2 Desafios 5

1.3 Visão Geral da Técnica Proposta 7

1.4 Demonstração e Análises 9

1.5 Contribuições 10

2 Trabalhos Relacionados **11**

2.1 Refinamento Através de Estratégias de Amostragem 12

2.2 Priorização de Raios 16

2.3 Ray Tracing Híbrido 18

3 Método para Distribuição Não Uniforme de Raios e Composição da Imagem Final **21**

3.1 Construção do Mapa de Importâncias 24

3.2 Construção do Mapa de Alocação de Raios 26

3.3 Cálculo da Imagem Integral 27

3.4 Definição do Mapa de Endereçamento de Raios 31

3.5 Traçado de Raios com Distribuição Não Uniforme 33

3.6 Composição da Imagem Final 34

4 Heurísticas para a Construção de Mapas de Importância **36**

4.1 Rasterização de Propriedades Ópticas 37

4.2 Computação de Mapa de Saliências 41

5 Experimentos e Análises **46**

5.1 Definição das Métricas Utilizadas 47

5.1.1 Relação Sinal-Ruído de Pico 47

5.1.2 Índice de Similaridade Estrutural 48

5.2 Visão Geral e Discussão dos Experimentos 52

5.2.1 Análise de Qualidade Visual 53

5.2.2 Análise de Desempenho 57

6 Conclusões e Trabalhos Futuros **61**

6.1 Trabalhos Futuros 62

Referências **65**

Lista de Figuras

1.1 Comparação ilustrativa entre uma distribuição não-uniforme de raios (A_i) e a maneira convencional uniforme (B_i). As seções em destaque mostram o salto de qualidade obtido em determinadas regiões da cena. Nota: a versão digital do texto oferece melhor apreciação dos resultados apresentados nesta imagem. 3

1.2 À esquerda, um exemplo de simulador de corrida onde o foco está sempre centralizado na pista e existe borramento constante nas adjacências, e à direita, um jogo em terceira pessoa com visão fixa superior onde o foco de atenção do observador varia constantemente com a posição do personagem em tela. Imagens retiradas de *Project Gotham Racing*^{TM}*4 - XBOX 360* e *Dead Nation*^{TM}*- PlayStation 3*. 5

2.1 Efeito de *aliasing* e sua redução em função da superamostragem de raios em determinadas regiões da cena clássica de Whitted. Nota: a versão digital do texto oferece melhor apreciação dos resultados apresentados nesta imagem. 14

2.2 Processo de priorização de raios usando fila de prioridades proposto por Steinberger. Adaptado de [Steinberger et al. 2012]. 17

2.3 *Framebuffer* do *pipeline* híbrido proposto por [Sabino 2012]. O *framebuffer* é composto pelas camadas de: (a) posição, (b) normal, (c) profundidade, (d) cor difusa, (e) cor especular e (f) propriedades opticas. (g) é a imagem final composta após a fase de iluminação. 19

2.4 Discrepância entre duas abordagens diferentes de síntese de imagem em um ambiente do *framework* de [Martinez et al. 2012] onde o tamanho da janela de traçado de raios não é suficiente para aplicar os efeitos de iluminação por todo o modelo. 20

3.1 *Pipeline* para a síntese de uma imagem a partir do método proposto. . . . 23

3.2 Zonas de importância sobre uma imagem gerada por traçado de raios: à esquerda, a imagem original e, à direita, uma possível configuração de como os valores de importância podem ser atribuídos a cada região. 25

3.3 Amostragem da integral de uma região retangular a partir de suas coordenadas. 28

3.4 Uso da técnica de *recursive doubling* para o cálculo de um conjunto de somas parciais. Extraído de [Hensley et al. 2005]. 30

3.5 Cálculo do número acumulado de raios escalonados para disparo até o pixel (i, j). Linha 3 do Algoritmo 2. 32

3.6 Traçado de raios sobre o plano de imagem partindo da câmera em direção ao centro do pixel (i, j). Alterando ligeiramente a direção do raio a partir de um deslocamento de $(\Delta i, \Delta j)$ no plano de imagem, garante-se que o conjunto de raios direcionados a um mesmo pixel atravessem a cena por caminhos diferentes, e assim, agreguem informação adicional à imagem. . . 33

3.7 Reconstrução da imagem final a partir de um Mapa de Traçado de Raios . 35

4.1 Uma cena antes de aplicação da técnica de análise de saliências (cima), seu mapa de saliências correspondente não ajustado (meio) e, seu mapa de saliências ajustado empiricamente com m na metade da média de I_{map} (baixo). Cena extraída de [Su and Takahashi 2010]. 44

5.1 Comparação ilustrativa entre uma imagem original (a) e diversos tipos de distorções aplicados a ela, todas com o valor do MSE = 210. O valor do SSIM entretanto, varia conforme cada distorção perde similaridade estrutural com a original, sendo para (b) SSIM = 0,916; (c) SSIM = 0,990; (d) SSIM = 0,695; (e) SSIM = 0,705 e (f) SSIM = 0,775. Extraído de [Wang et al. 2004]. 49

5.2 Cena clássica de Whitted e seu Mapa de Importâncias correspondente. . . 52

5.3 Cena Spheres, gerada aumentando o grau de dificuldade da cena Whitted, e seu Mapa de Importâncias correspondente. 53

5.4 Gráficos de erro das diferentes configurações na cena Whitted em relação às metricas estudadas. 56

5.5 Gráficos de erro das diferentes configurações na cena Spheres em relação às metricas estudadas. 57

5.6 Tempo de execução das estruturas do pipeline em relação ao tempo total gasto para síntese de imagem conforme o aumento do número de raios primários, para cada uma das cenas estudadas. 58

Capítulo 1

Introdução

No campo da computação gráfica, algumas aplicações demandam uma representação fiel da realidade. Dentre elas, encontram-se simuladores de diversos tipos, jogos cada vez mais realistas e filmes, que não só movimentam uma quantidade extraordinária de dinheiro todos os anos, como também exercem uma influência direta a um público extenso, distribuído pelo mundo todo. Para suprir essa demanda, técnicas de síntese de imagem vêm sido estudadas exaustivamente pela academia durante as últimas décadas e, dentre as suas aplicações, aquelas baseadas em traçado de raios em específico têm ganhado cada vez mais atenção.

A técnica de traçado de raios (do inglês, *ray tracing*), desde que foi proposta por [Whitted 1980], tem sido um grande alvo de estudos, já que, ao simular a interação da luz de uma cena com o ambiente, é capaz de produzir imagens com um grau bastante elevado de realismo. A ideia do traçado de raios é simples e pode ser resumida como *traçar raios do observador em direção à cena e calcular o efeito das suas interações com o ambiente.* O algoritmo é, por construção, altamente paralelizável e a maior dificuldade encontrada na sua aplicação é a quantidade de processamento que cada raio demanda enquanto interage com a cena: caso um raio colida com uma superfície reflexiva, um raio refletido é gerado no ponto de interseção e disparado a partir do ângulo refletido em direção à cena; caso um raio colida com uma superficie translúcida, um raio é gerado e transmitido através do material em uma nova direção; e finalmente, um raio de sombra é criado para testar se o ponto de interseção do raio primário (i.e., raios disparados à partir da câmera em direção à cena) se encontra ocludido de alguma fonte de luz presente no ambiente. Vale ressaltar que cada superfície pode ter diferentes graus de reflectividade, refratividade e estar sobre a sombra de um ou mais objetos, das mais diversas opacidades. Sendo assim, esses casos não são mutualmente exclusivos e todo o processamento deve ser composto

em um resultado final para a cor daquele ponto. No fim deste processo, uma quantidade suficiente de interreflexão, desvios por refrações e geração de sombra para cada fonte de luz pode tornar a sua computação impraticável.

A técnica de traçado de raios, mesmo aplicada da sua forma mais simples, é capaz de produzir resultados suficientemente fieis a realidade para a maioria das aplicações. Porém, como pode-se perceber facilmente, aplicar computacionalmente um sistema que simule o comportamento de uma quantidade suficientemente grande de raios pode não ser viável, devido à quantidade de processamento exigido para o cálculo da intensidade final de cada pixel. Naturalmente, o caráter paralelo do algoritmo vem sido muito explorado para a obtenção de um traçador de raios que seja aplicável em ambientes de tempo real, onde é preciso que a imagem final seja gerada em frações de segundo. Por mais que a ideia do algoritmo seja facilmente traduzida para um contexto paralelizável, é difícil manter a coerência no acesso a todos os dados, já que o comportamento de cada raio de luz é independente dos demais, o que afeta diretamente a performance das implementações da técnica. Além disso, o custo computacional de traçar raios sobre uma cena qualquer cresce vertiginosamente conforme a complexidade da cena em questão. Parâmetros como o número de objetos em cena e o tipo de material que compõe cada um têm uma influência direta sobre a quantidade de computação necessária para processar a cena. Isso é decorrente do fato de que cada um do raios que saem da câmera, ao se chocar com um objeto reflexivo, por exemplo, projeta aquele ponto em um novo ponto da cena, que também exige processamento. Este fato se repete para cada colisão de raio com uma superfície não trivial de ser calculada, isto é, aquelas que exigem processamento de reflexão ou refração. Dessa maneira, para cada raio, a aplicação precisa disparar diversos raios secundários para auxiliar o cálculo da intensidade do pixel que ele atravessa.

Aplicar uma técnica computacionalmente custosa em ambientes regrados por restrições de tempo pode se tornar uma tarefa difícil em certos casos. No caso do traçado de raio, não é diferente. Dependendo das restrições de tempo apresentadas, o tipo de cenário que a técnica é capaz de processar se torna bastante limitado. Dentro desse contexto, é interessante que a utilização dos recursos disponíveis seja otimizada e que não haja desperdício de processamento. Logicamente, dada uma cena qualquer a ser renderizada, nela estão inclusas regiões cujo processamento é mais simples e direto, e outras, cujos efeitos de iluminação esperados demandam um processamento muito mais rebuscado, dependendo do traçado de raios secundários e da amostragem de informação de diversas outras partes da cena. Ingenuamente, espera-se que ao distribuir o processamento que se dispõe baseado nas particularidades de cada cena, seja possível dar um salto de qua-

lidade nas cenas que podem ser processadas dentro do limite estipulado. A Figura 1.1 ilustra o salto de qualidade obtido ao transformar uma distribuição uniforme de 2 raios por pixel do plano de imagem em uma baseada na importância das característas da cena. Infelizmente, distribuir os raios desta maneira não é uma tarefa trivial e exige que cada instância de processamento do *hardware* gráfico (GPU) seja manualmente associada a um raio a ser disparado, o que sem a estrutura necessária, insere um caráter sequencial no processamento de cada *thread* da aplicação paralela do algoritmo. Isto adiciona uma carga extra de computação em um processo já custoso e por si só, é capaz de inviabilizar a utilização da técnica.

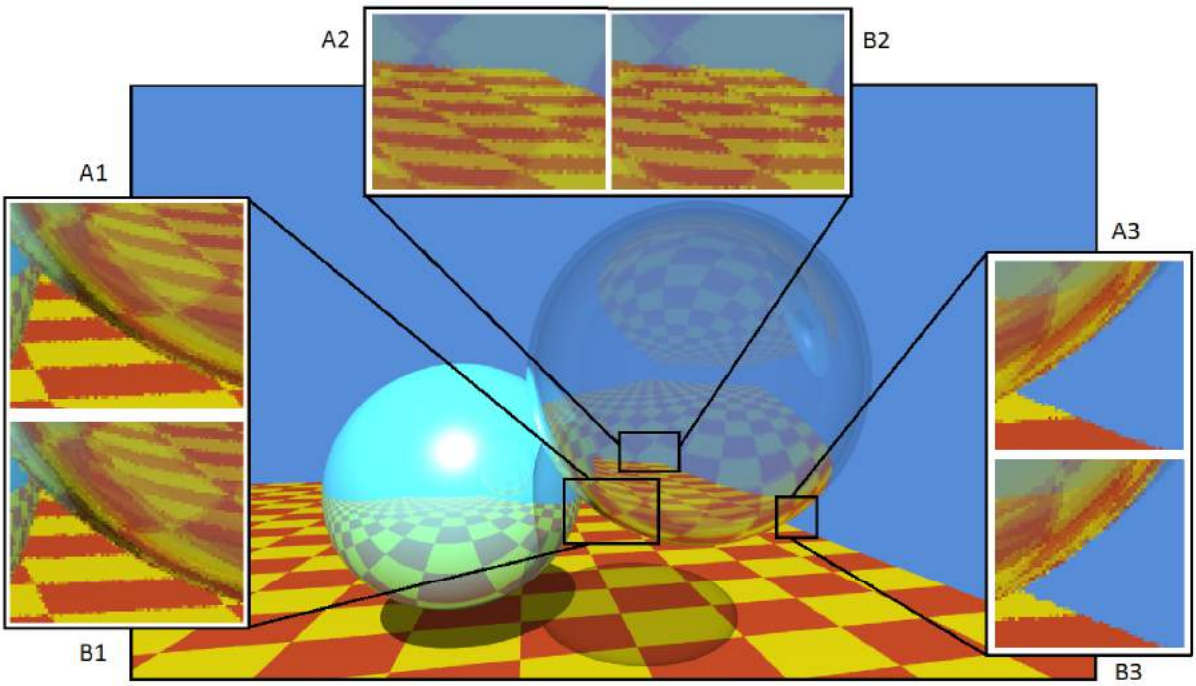

Figura 1.1: Comparação ilustrativa entre uma distribuição não-uniforme de raios (A_i) e a maneira convencional uniforme (B_i). As seções em destaque mostram o salto de qualidade obtido em determinadas regiões da cena. Nota: a versão digital do texto oferece melhor apreciação dos resultados apresentados nesta imagem.

Este trabalho propõe uma alternativa para o escalonamento dos raios primários da técnica clássica de traçado de raios em ambientes de tempo real. Nestes ambientes, a técnica implementada pelos motores de traçado de raios atuais ainda é bastante limitada e se encaixa apenas em cenários capazes de serem processados dentro de um orçamento pré-determinado de tempo que não comprometa a interatividade do sistema. Sabendo a quantidade de processamento que pode ser realizada dentro das limitações estipuladas pela aplicação, pode-se, em vez de distribuir este processamento de forma homogênea, determinar as regiões onde ele será melhor aproveitado. Fazendo isso, deseja-se atingir

uma qualidade de imagem superior à alcançada originalmente, estando sob as mesmas condições. Uma estratégia para obter esse resultado é fazer uso de heurísticas capazes de determinar essas regiões de interesse. Para isso, propomos um *pipeline* de traçado de raios, composto de uma série de estruturas desenvolvidadas com a finalidade de traduzir a informação de importância, mapeada sobre o plano de imagem, para dados que a placa de vídeo consiga manipular. É importante observar que a abordagem proposta introduz uma série de novas etapas no processo original de síntese de imagem. Em linhas gerais, essas estapas costistem em estimar o quanto de processamento se dispõe, avaliar das zonas de importância, distribuir o orçamento de raios primários de acordo, endereçar das *threads* da GPU aos raios disparados de maneira não uniforme, processar cada raio paralelamente e, finalmente, sintetizar as informações obtidas em uma imagem final. Todas essas etapas devem ser executadas de maneira eficiente, sem comprometer o bom uso das características paralelas da arquitetura.

1.1 Ideia Central

Dentre as diversas vertentes de trabalhos relacionados ao uso da técnica de traçado de raios, este trabalho faz parte do grupo de pesquisa que utiliza heurísticas bem definidas para aprimorar a resposta do algoritmo, bem como se encaixa nos que propõem novas maneiras de abordar o problema de forma concorrente. Sua ideia central pode ser descrita como o estudo da seguinte hipótese:

> *"É possivel fazer uma distribuição não uniforme de raios primários em uma cena tridimensional utilizando estruturas de dados adequadas para que a aplicação seja executada de forma concorrente na GPU e apropriada para ambientes de tempo-real. Para guiar tal distribuição, é preciso aplicar heurísticas pertinentes para que a síntese de imagem seja feita aproveitando as particularidades da cena em questão."*

Vale ressaltar que mesmo que este trabalho vise promover melhor qualidade visual nas regiões perceptualmente mais importantes de uma determinada cena, esse conceito varia de aplicação para aplicação. Por exemplo, em um simulador de corrida, faz sentido que seja aplicada uma função heurística que associe maiores valores de importância para regiões próximas ao centro do plano de imagem em detrimento às periféricas, já que ao dirigir, o observador se concentra prioritariamente nos detalhes do meio da pista, sempre a sua frente, ignorando a região da cena nas bordas do plano de imagem, muitas vezes

até borrada em função da velocidade. Essa ideia contrasta, por exemplo, com um jogo simples em terceira pessoa, onde normalmente a atenção do observador é focada ao redor do personagem principal, aonde quer que este esteja no plano de imagem. Por esse motivo, por mais que demonstremos a criação de heurísticas que se adaptem ao *pipeline* proposto (Capítulo 4), criar heurísticas para estudar todos os casos referentes a cada tipo de aplicação, tal como fazer comparações de performance entre elas, não fazem parte do escopo deste trabalho. A Figura 1.2 ilustra a diferença entre os dois cenários citados nessa discussão.

Figura 1.2: À esquerda, um exemplo de simulador de corrida onde o foco está sempre centralizado na pista e existe borramento constante nas adjacências, e à direita, um jogo em terceira pessoa com visão fixa superior onde o foco de atenção do observador varia constantemente com a posição do personagem em tela. Imagens retiradas de *Project Gotham Racing*^{TM}*4 - XBOX 360* e *Dead Nation*^{TM}*- PlayStation 3.*

Naturalmente, existem aplicações que se aproveitam das diferenças entre as regiões de maior percepção do observador em uma determinada aplicação [Guenter et al. 2002]. Porém parte do objetivo deste trabalho é propor uma maneira abrangente de adequar uma aplicação às particularidades de cada um destes cenários de maneira intuitiva, evitando a necessidade de modificar o arcabouço principal do sintetizador de imagem.

1.2 Desafios

Para atingir o objetivo deste trabalho, pode-se identificar alguns desafios principais que precisam ser superados:

1. Criar um *pipeline* de tradução da informação de importância das regiões do plano de imagem e de execução do traçado de raios que se adeque à natureza paralela do *hardware* gráfico atual; e

2. Determinar maneiras de fazer a distribuição dos raios primários através da cena de

forma eficiente.

Dada a arquitetura utilizada para o desenvolvimento deste trabalho (i.e. NVIDIA OptiX), foi identificado um desafio secundário:

- Definir uma estrutura de endereçamento de raios a pontos do plano de imagem que possa ser acessada de forma concorrente pelo algoritmo de traçado de raios.

A proposta deste trabalho requer que uma informação de importância para cada região da cena seja traduzida na forma de parâmetros para o algoritmo de traçado de raios. Tratando-se de um ambiente de tempo-real, essa informação deve ser organizada e codificada de uma maneira eficiente, e em seguida, usada para calcular a proporção de recursos dentre aqueles que a aplicação dispõe a ser destinada a cada região do plano de imagem. Aproveitando o caráter paralelo do algoritmo de traçado de raios, cada passo de pré e pós processamento (tradução de informação de importância e composição da imagem final a partir do resultado do traçado de raios, respectivamente) que a técnica proposta necessite deve ser igualmente capaz de ser processado em GPU, para que assim, nenhum gargalo de processamento sequencial seja adicionado (desafio 1). A solução proposta é criar um *pipeline* onde, em cada etapa, a informação de importância seja organizada de forma que duas ou mais *threads* não a acessem ao mesmo tempo. Além disso, para manter a viabilidade do algoritmo, deve-se evitar que essas informações sejam buscadas na memória de forma exaustiva, sendo o ideal recuperá-las com o menor numero de operações possível. Essa meta pode ser atingida adaptando-se o uso da técnica de Imagem Integral (do inglês, *Summed Area Table* [Crow 1984]) para que o *pipeline* seja capaz de transmitir a informação de quantos raios foram escalonados até determinado ponto, para cada pixel do plano de imagem. Feito isto, é possível fazer o endereçamento de cada *thread* à informação pertinente ao raio que for traçar de maneira direta (Capítulo 3).

Para fazer a distribuição de raios pela cena como proposto, o traçador de raios necessita de uma regra bem estabelecida que dite para cada pixel do plano de imagem a quantidade de raios que serão disparados sobre ele. No Capítulo 4, são propostas duas maneiras diferentes de estimar a importancia das regiões do plano de imagem baseado na cena em questão (desafio 2). A primeira é a rasterização prévia da cena, capaz de identificar regiões com alto teor de propriedades opticas que é um indicativo de que devem ser marcadas como importantes e assim concentrar recursos de processamento; e a segunda, a computação de um mapa de saliências, que utiliza a distinção de saliências nas superfícies da cena como métrica para modelar as areas de interesse do sistema visual humano.

Sabendo organizar a informação de importância da cena a ser renderizada, e em seguida transportar e traduzir essa informação ao módulo traçador de raios de uma maneira eficiente, resta-nos criar uma estrutura de entrada para que os raios da aplicação tradicional do algoritmo de traçado de raios distribuídos uniformemente pelo OptiX sejam redirecionados como proposto (desafio secundário). Este problema é atacado através da criação de um uma estrutura que, dadas as informações passadas pelo *pipeline* até então, é capaz de armazenar uma coordenada específica do plano de imagem para cada índice de raio a ser traçado (Seção 3.4). Dentro do módulo traçador de raios tradicional programado como um *kernel* OptiX, cada instância de raio gerada é associada automaticamente a um índice na memória e direcionada em sequência para que atravesse de maneira uniforme o plano de imagem. Para aplicar a distribuíção segundo as particularidades da cena, deve-se substituir os índices de acordo com a distribuição pretendida e em seguida ajustar a direção dos raios instanciados de forma coerente no momento que forem disparados. A partir daí, o algoritmo traçador de raios faz a amostragem da cena e armazena o resultado retornado por cada raio numa textura de saída, que em seguida é recombinada a partir do mesmo mapa de endereçamentos em uma imagem final.

1.3 Visão Geral da Técnica Proposta

Primeiramente, a partir de uma heurística compatível, é gerado um Mapa de Importâncias, responsável por descrever o impacto visual de cada região do plano de imagem. Esse valor de importância pixel a pixel é a base para calcular o quanto de processamento se concentrará em cada região da cena. Partindo do princípio que uma imagem não atrai a atenção do sistema visual humano de forma homogênea e/ou apresenta diferentes níveis de interação com a luz do meio, naturalmente, devemos também associar diferentes quantidades de raios a serem traçados para cada uma dessas regiões, promovendo, assim, melhor qualidade visual nas regiões perceptualmente mais importantes. Vale lembrar, que esta função heurística compatível exerce o papel de uma função preditiva, pois, mesmo com a cena não renderizada, ela deve ser capaz de associar importância a cada pixel da imagem final.

Na segunda etapa do método, é preciso traduzir o valor de importância absoluto de cada elemento do Mapa de Importâncias para um valor inteiro que represente a quantidade de raios que deve atravessar o pixel correspondente a ele no plano de imagem. Para isso, precisa-se avaliar o Mapa de Importâncias como um todo e normalizá-lo pela quantidade de recursos disponíveis à aplicação. Sem essa etapa, os recursos disponíveis podem ser

gastos de maneira gulosa nas primeiras posições do Mapa de Importâncias e não há como garantir que o algoritmo traçe raios por todo o plano de imagem antes de estourar sua cota de processamento. Assumindo que a função heurística que tenha gerado o mapa de importâncias seja admissível, ou seja, que nunca informe um valor de importância aproximado maior do que o valor de importância do arranjo ótimo, pode-se dizer que, nesta etapa, a informação de importância pixel a pixel do plano de imagem é traduzida em uma distribuição heterogênea de recursos (raios primários) sobre a cena.

Em seguida, utilizando o conceito de Imagem Integral, é possível determinar a quantidade de recursos alocados para distribuição da origem do plano de imagem até cada um de seus pixels. Esta informação é essencial para disparar os raios escalonados de forma paralela dentro do módulo traçador: a Imagem Integral permite que cada raio tenha acesso à soma parcial da quantidade de recursos já alocada, e assim saber quantos raios já foram escalonados para disparo nas posições anteriores do plano de imagem. O endereçamento de um raio para um pixel, então, é feito de forma direta, independente da forma com que estes estão distribuídos. A informação de endereçamento de cada raio é organizada na forma de um Mapa de Endereçamento de Raios, para que no momento do disparo, seja possível realocar cada raio e direcioná-lo à posição do plano de imagem a que esse esteja associado.

Finalmente, no momento em que os raios são disparados, o módulo de traçado de raios é chamado e gera uma estrutura semelhante ao Mapa de Endereçamento de Raios, contendo o valor de cor obtido por cada raio. Essa estrutura de saída do traçador é então transformada em uma imagem da mesma dimensão do plano de imagem para qual foram alocados os recursos. Isso pode ser feito com base no do número de raios que atravessa cada pixel da imagem, obtido em tempo constante, e a partir da integral de imagem, necessária para fazer o cálculo do endereçamento reverso. Associando uma *thread* a cada pixel do plano de imagem, em condições ideias, onde seja possível rodar todas as *threads* em paralelo sem interrupções, pode-se construir a imagem final em complexidade de tempo $O(K)$, onde K é o número máximo de raios que atravessam cada pixel. Naturalmente, essa análise é dependente do valor definido para K, bem como da capacidade de processamento paralelo do hardware gráfico em questão. No caso geral, para uma imagem suficientemente complexa e um número máximo arbitrário de raios atravessando cada pixel, o estudo da complexidade desta etapa envolve *warps*, pausas no processamento, e demais particularidades da arquitetura da placa de vídeo utilizada. Sendo assim, ele não será abordado em detalhes.

1.4 Demonstração e Análises

A validação da técnica proposta e a demonstração da execução do traçado de raios a partir do modelo de *pipeline* criado foram feitas utilizando C++ e CUDA para suporte ao paralelismo de pré e pós processamento. Para o módulo traçador de raios, foi utilizado o NVIDIA OptiX, que é um motor para construir aplicações baseadas em traçado de raios. O OptiX oferece um motor de traçado de raios de alto nível, um *pipeline* programável baseado em uma linguagem de *shader* específica, um compilador próprio para geração de código executado em GPU e uma representação em forma de grafos de cena. Tudo isso facilita o processo de desenvolvimento, permitindo maior foco nas etapas de geração de raios, iluminação, amostragem de dados da cena e administração de recursos, já que pode-se abstrair as etapas de construção e atravessamento das estruturas de divisão espacial de cena e de aceleração.

Quanto aos experimentos, eles são feitos comparando a qualidade das imagens geradas pelo algoritmo de traçado de raios convencional para aplicações em tempo real com os resultados obtidos a partir do *pipeline* proposto. Para cada cena estudada, a partir de Mapa de Importâncias, o *pipeline* proposto é construído e assim é definida uma distribuição não uniforme de raios. O modulo traçador, então, faz a amostragem da cena a partir desse mapa e a reconstrói para que possa ser comparada com a imagem gerada pelo método tradicional. Esse procedimento é repetido aumentando gradativamente o número de recursos disponíveis, e assim, a quantidade de raios a serem disparados por pixel. Analisando diferentes orçamentos de raios, é possível verificar se existe um ganho de qualidade caso sejam dispostos conforme uma determinada heurística, em vez de serem disparados uniformemente sobre a cena. Além disso, no Capítulo 4, são propostas duas heurísticas originais para a geração de Mapas de Importâncias de forma geral.

Para que uma comparação de qualidade possa ser feita, é preciso que se defina algumas métricas. Neste trabalho, foi adaptado o conceito de relação entre sinal e ruido de pico (PSNR, do inglês, *Peak Signal to Noise Ratio* [Welstead 1999]) para que este pudesse ser aplicado dentro do escopo de comparação de imagens. Em seguida, como métrica alternativa, foi utilizado o índice de similaridade estrutural (SSIM, do inglês *Structural Similarity* [Wang et al. 2004]), que abrange essa comparação a conceitos como luminância, contraste e estrutura das imagens. O Capítulo 5 define as métricas utilizadas com um maior rigor e aborda os experimentos executados de forma detalhada.

1.5 Contribuições

Esta dissertação apresenta as seguintes contribuições:

- Um *pipeline* paralelo para a realocação de raios primários em uma aplicação de traçado de raios em tempo real;
- Heurísticas para estimar a importância dos elementos de uma cena tridimensional a ser renderizada.

Além disso, ao longo do desenvolvimento deste trabalho foram feitas as seguintes demonstrações:

- É possível implementar as estruturas de dados propostas em paralelo na GPU;
- É possível contornar as limitações de endereçamento entre as instâncias de raio e as *threads* da arquitetura da GPU para traçar raios de forma heterogênea no plano de imagem.

Finalmente, este trabalho tem como objetivo dar um passo adiante em uma área que é estudada tão exaustivamente, através de métodos que aprimoram a performance do algortimo clássico de traçado de raios dentro das restrições dos ambientes em tempo real. Esperamos obter resultados que encoragem o uso e estudo dos métodos propostos em aplicações futuras, visando um dia atingir uma configuração ótima de distribuição de raios e assim a qualidade máxima de imagem possível dentro das restrições de tempo estipuladas por sistemas em tempo real.

Capítulo 2

Trabalhos Relacionados

Na área da computação gráfica, em geral, os trabalhos buscam novos métodos e formas para otimizar a síntese de uma imagem final: seja visando uma maior qualidade da imagem projetada; seja remodelando a representação das estruturas durante o método para facilitar esse processo; ou seja polindo de diferentes formas algoritmos de processamento da cena a afim de adequá-los a alguma restrição física dentro do escopo da aplicação, como por exemplo aplicações de tempo real, que exigem uma taxa de quadros por segundo suficientemente alta para obterem uma boa funcionalidade. O escopo deste trabalho é justamente dentro de aplicações de tempo real, cuja demanda tem crescido vertiginosamente com o cada vez maior desenvolvimento de simuladores e jogos eletrônicos, que para manter a interatividade, exigem a geração de imagens com alto fotorrealismo de uma forma fluida, transparente para o observador.

Diferentes formas de processar uma imagem trazem diferentes benefícios: vantagens e desvantagens inerentes a cada uma das abordagens escolhidas. Sendo assim, por muito tempo, foi de comum acordo dentro da comunidade científica, que enquanto trabalhando com aplicações de síntese de imagem interativas, tratava-se quase que exclusivamente com técnicas de rasterização. Técnicas estas que por mais que obtenham uma qualidade de imagem inferior ao sintetizar a imagem final quando comparadas às técnicas de traçado de raios, eram as únicas a atingir a taxa de processamento por segundo adequada às aplicações em tempo real [Wald 2004]. Conforme novos trabalhos foram sendo desenvolvidos e novas tecnologias vieram sendo aplicadas na forma de novo *hardware*, as técnicas de traçado de raio, que até então não eram viáveis nesse escopo em função do seu grande custo computacional, foram ganhando um espaço considerável e assim, hoje em dia, há um grande esforço da comunidade em busca de consolidá-las definitivamente no escopo das aplicações de síntese de imagem em tempo real [Keller et al. 2013].

Diferentes abordagens para otimizar a aplicação do algoritmo clássico de traçado de raios vêm sido propostas e o escopo deste trabalho abrange uma vasta bibliografia, onde as mais diversificadas formas de atacar o problema estão presentes:

1. Refinamento através de estratégias de amostragem. (Seção 2.1)
2. Priorização de raios baseados em heurísticas. (Seção 2.2)
3. Motores híbridos de traçado de raios e rasterização. (Seção 2.3)

Toda a bibliografia sobre aplicação de algoritmos de traçado de raio em tempo real tem um objetivo comum: tornar possível a obtenção da qualidade dos resultados do algoritmo de traçado de raios dentro das restrições de tempo e recursos dos ambientes interativos. Cada uma das vertentes enumeradas acima usa de uma estratégia específica para atingir este objetivo. Ora tenta-se limitar o uso do algoritmo de traçado de raios à apenas determinadas etapas do processamento, conjugando-o com algoritmos mais simples; ora opta-se pelo refinamento informado da técnica, utilizando estimativas de orçamento de tempo como métrica. Das diversas abordagens possíveis, várias contribuíram para o desenvolvimento deste trabalho e serão comentadas brevemente no decorrer das próximas seções, de forma a estabelecer uma contextualização do leitor com a estratégia escolhida para atacar o problema.

A literatura disponível sobre o traçado de raios clássico é bastante extensa, mas caso o leitor se interesse, o livro sobre traçado de raios de [Glassner 1989] é recomendado, pois aborda o desenvolvimento do traçado de raios clássico com bastante clareza. Além disso, os seguintes artigos seminais contém informação adicional sobre as diferentes técnicas envolvendo o traçado de raios: [Arvo and Kirk 1989] abordam os fundamentos das técnicas de aceleração, [Cook 1986] aborda o problema do *ailiasing* em amostragem de raios regularmente espaçadas, [Lee et al. 1985] relacionam o uso de diversos raios por pixel com a estimativa da integral da imagem e assim, a sua influência na qualidade da imagem final.

2.1 Refinamento Através de Estratégias de Amostragem

No algoritmo tradicional de traçado de raios, deve-se traçar uma série de raios por toda a cena com o objetivo de reunir informações suficientes para a construção da imagem final. Naturalmente, dada uma cena qualquer, espera-se que determinadas regiões agreguem

mais informações e sejam mais representativas que outras. Sendo assim, um algoritmo de amostragem de dados no qual essas regiões fossem priorizadas seria capaz de extrair uma quantidade maior de informação e levaria a um resultado final mais fiel à cena em questão. Seguindo esse princípio, tanto as abordagens de traçado de raios tradicionais, que utilizam amostragem pontual de um raio por pixel, quanto as mais sofisticadas, como a abordagem de traçado de raios distribuída [Cook et al. 1984] que calcula o valor de cada pixel através da estimativa da integral da sua função de densidade na sua área, adotam diferentes estratégias de amostragem visando aperfeiçoar a qualidade da imagem final. Essas estratégias compreendem modelos adaptativos de amostragem, modelos hierárquicos ou até estocásticos, cada um visando obter melhores resultados em um determinado tipo de cenário.

O problema da amostragem em aplicações de traçado de raios vem sido atacado desde a proposta do ray tracing recursivo por [Whitted 1980]. O algoritmo do Whitted faz a amostragem da imagem na forma de uma grade quadrada regular e subdivide cada célula de forma recursiva se as amostras retiradas dos seus cantos forem suficientemente diferentes, dentro de uma métrica estabelecida. Este é um exemplo de sistema adaptativo que refina a amostragem de maneira informada, entretanto por não ter um caráter estocástico, apresenta *aliasing* estruturado, dada a forma com que a imagem final é construída. O *aliasing*, nada mais é do que o fenômeno de distorção de um sinal qualquer ocasionado por uma sequência de erros provenientes da maneira de como é representado. Na computação gráfica, ele se apresenta principalmente através do serrilhamento das curvas ou linhas diagonais em uma imagem de resolução suficientemente baixa. Observando atentamente as bordas dos objetos das regiões em destaque da Figura 1.1, pode-se verificar a aparição deste fenômeno através do seu efeito de serrilhado característico, tal como a sua redução conforme a distribuição não uniforme do *pipeline* proposto neste trabalho superamostra a mesma região. A Figura 2.1 apresenta um *zoom* de umas das regiões em destaque, a fim de ilustrar ainda mais essas diferenças, perceptíveis especialmente no contorno das esferas e nos ladrilhos da reflexão do piso, conforme estes vão diminuindo em perspectiva.

Com a finalidade de otimizar a computação dos testes de colisão entre os raios disparados e o ambiente, o conceito de refinamento sucessivo de amostragem continuou sendo estudado. [Glassner 1984] propôs o uso de uma *octree*, uma estrutura de divisão espacial da cena, ao computar a interseção dos raios com o ambiente. [Reeth et al. 1996] usaram métodos projetivos em *quadtrees* e em grades regulares. [Foley and Sugerman 2005] levaram essa ideia além, demonstrando que algoritmos para percorrer *kd-trees* eram adequados ao problema ao integrá-los a um traçados de raios. Hoje em dia, a maioria dos

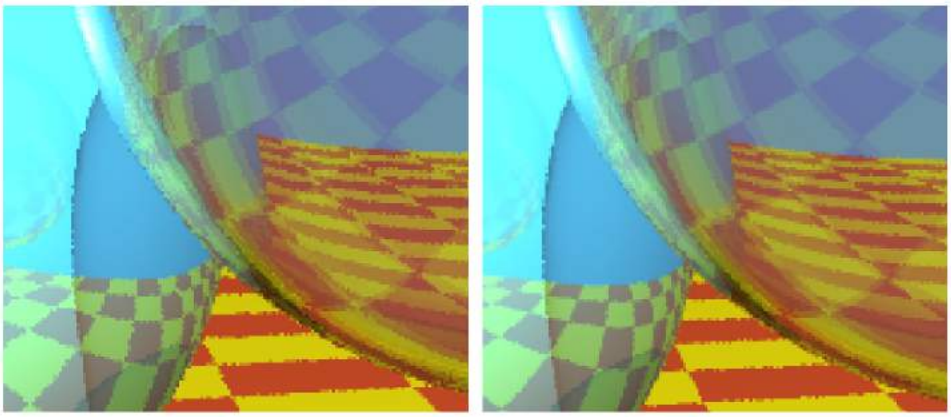

Figura 2.1: Efeito de *aliasing* e sua redução em função da superamostragem de raios em determinadas regiões da cena clássica de Whitted. Nota: a versão digital do texto oferece melhor apreciação dos resultados apresentados nesta imagem.

motores de traçado de raios mais modernos, como o que utilizamos neste trabalho, o NVIDIA OptiX [Parker et al. 2010] usam algoritmos de aceleração mais sofisticados, como o *Split Bounding Volume Hierarchy (SBVL)* [Stich et al. 2009], ou o *Linear Bound Volume Hierarchy (LBVL)* [Lauterbach et al. 2009], capazes de atingir uma performance melhor do que os anteriores.

Além de algoritmos de divisão do espaço tridimensional, outras técnicas visando tirar proveito da coerência de dados vem sendo desenvolvidas para aprimorar o uso do traçado de raios em cenários interativos. [Ruff et al. 2013] desenvolveram um sistema capaz de armazenar informações de frames anteriores em *cache* de forma a aproveitar o acesso coerente na memória e obter uma amostragem eficiente dos seus dados. Como resultado, foram capazes de atingir um aumento na taxa de quadros por segundo da aplicação em cenas estáticas.

Para estratégias estocásticas de distribuição de raios em uma cena espera-se que as amostras sejam geradas em maior densidade nas regiões da imagem que sejam mais representativas. Isso pode ser obtido aumentando o número de amostras nas regiões que apresentam um maior erro dentro de uma determinada métrica, seja ela alto contraste, variância, erro quadrático médio, etc [Mitchell 1987, Mitchell 1991]. Para evitar a geração de *aliasing* é importante que as amostras não sejam geradas em um padrão que se replique uniformemente pelo plano de imagem. Sendo assim, algumas estratégias de amostragem envolvem a aplicação de filtros na amostragem dos pixels para simular uma distribuição não-uniforme, ou até, aproveitam a discrepância entre a imagem e os seus diferentes padrões de amostragem para definir novas maneiras de fazê-la [Shirley 1991]. Além disso, pode-se refinar a amostragem dos raios de forma adaptativa baseado nas informações

já calculadas em quadros anteriores [Bolin and Meyer 1998, Rigau et al. 2003]. Essa estratégia pode reduzir drasticamente o número de amostras necessárias para obter a mesma qualidade de imagem comparado às anteriores, porém também apresenta suas limitações, como por exemplo na presença de efeitos de *motion blur*.

Um outro trabalho notável nesta área é o desenvolvido por [Rousselle et al. 2011], que foi capaz de obter melhoras de qualidade significativas minimizando de maneira gulosa o erro numérico de uma imagem em relação à quadros anteriores através de técnicas de amostragem adaptativas e reconstrução da imagem final usando o método de Monte Carlo [Shirley et al. 1996]. A técnica consiste em usar um algoritmo guloso para minimizar o erro quadrático médio da aplicação de uma série de filtros pré determinados sobre o quadro anterior e em seguida, dado o filtro escolhido, distribuir amostras adicionais de forma a reduzir ainda mais esse erro. O resultado desse algoritmo influencia diretamente na distribuição escolhida. Posteriormente, o uso de técnicas de retirada de ruídos antes da estimativa de erro para escolher os filtros alcançou ainda melhores resultados em iterações seguintes [Rousselle et al. 2012].

Naturalmente, pode-se desenvolver um método para gerar um bom padrão de amostragem sobre uma cena baseando-se na importância de cada parte dela. Estabelecendo uma função de densidade de importância sobre um domínio 2D referente ao plano da imagem final, inúmeras técnicas de refinamento podem ser aplicadas para a obtenção de uma amostragem mais próxima da ótima. Assim como uma heurística de importância, outros parâmetros são viáveis: a BRDF de uma superfície, sigla em inglês para *Bidirectional Reflectance Distribution Function*, ou seja, a função que retorna a radiância refletida de cada ponto da superfície em função de suas características e dos ângulo de entrada e saída dos raios de luz; a distribuição de luminosidade; propriedades físicas da geometria da cena, como agrupamento de saliências ou rugosidades; ruídos da cena e saliências de superfície, dentre outros. Dada uma função que se comporte de forma análoga à importância que cada aspecto da cena apresenta, o problema de amostragem a partir dela pode ser descrito da seguinte forma:

1. Dada a função de densidade de importância I sobre o domínio do plano de imagem D, analiticamente ou apresentada como uma sequência de valores discretos. I pode ser normalizada, sem perda de generalidade para que sua integral sobre o domínio seja sempre 1 e ainda, não apresente valores negativos de importância, respeitando $0 \leq I(x, y), \forall (x, y) \in D$.

2. Encontrar um conjunto de amostras discreto, tal que a densidade local do con-

junto δ ($\delta = numero_de_amostras/unidade_de_area$) seja proporcional ao valor de importância de I.

Este é um problema difícil, porém pode ser relaxado e ainda assim obter resultados interessantes. Uma solução viável suficientemente boa, mesmo que não ótima, pode ser capaz de aperfeiçoar a forma com que a amostragem de dados é feita na cena, e subsequentemente, a forma com que esta é renderizada utilizando um orçamento fixo de recursos. Na Seção 3.1 deste trabalho, o método utilizado para gerar uma distribuição de densidade de importâncias semelhante é explicado em maior riqueza de detalhes, e nas seções seguintes, como a partir desta distribuição, pode-se chegar a um resultado que satisfaz o segundo item da definição do problema acima. Vale ressaltar que, devido a estrutura do *pipeline* proposto separar a cena renderizada do Mapa de Importâncias gerado a partir da função heurística, o método não é necessariamente baseado na imagem gerada como em estratégias anteriores [Rousselle et al. 2011, Rousselle et al. 2012], e assim, apresenta uma limitação a menos quanto ao caráter adaptativo que o sistema é capaz de atingir.

2.2 Priorização de Raios

Uma segunda abordagem para aprimorar o uso do algoritmo clássico de traçado de raios sobre uma cena em tempo real pode ser entendida como um caso particular de um problema de amostragem, porém sob uma ótica diferente. Intuitivamente, ao traçar raios sobre uma cena tendo limitações de tempo, escolhe-se um subconjunto do número de raios a serem disparados e esses raios são instânciados dentro de um orçamento pré estabelecido. Em vez disso, pode-se, analogamente, ordenar todos esses raios dentro de uma fila de prioridades, cuja métrica é baseada na importância do destino de cada um, e dispará-los enquanto o caráter interativo da aplicação se mantiver. Este algoritmo é por construção um algoritmo guloso, mas pode atingir uma qualidade ótima de imagem, dentro das restrições de tempo e processamento apresentadas, dada que a ordenação de prioridade dos raios seja a melhor possível. Novamente, uma solução viável suficientemente boa é capaz de aprimorar o algoritmo clássico em muitas vezes [Kainz et al. 2011], e permitir que se atinja uma qualidade de imagem típica de traçado de raios em aplicações de tempo real.

O trabalho de [Steinberger et al. 2012] produz um *buffer* auxiliar para cada quadro gerado, eficiente o suficiente para ser atualizado a cada passada de geração de raios durante a aplicação, responsável por relacionar cada região a uma determinada densidade de raios.

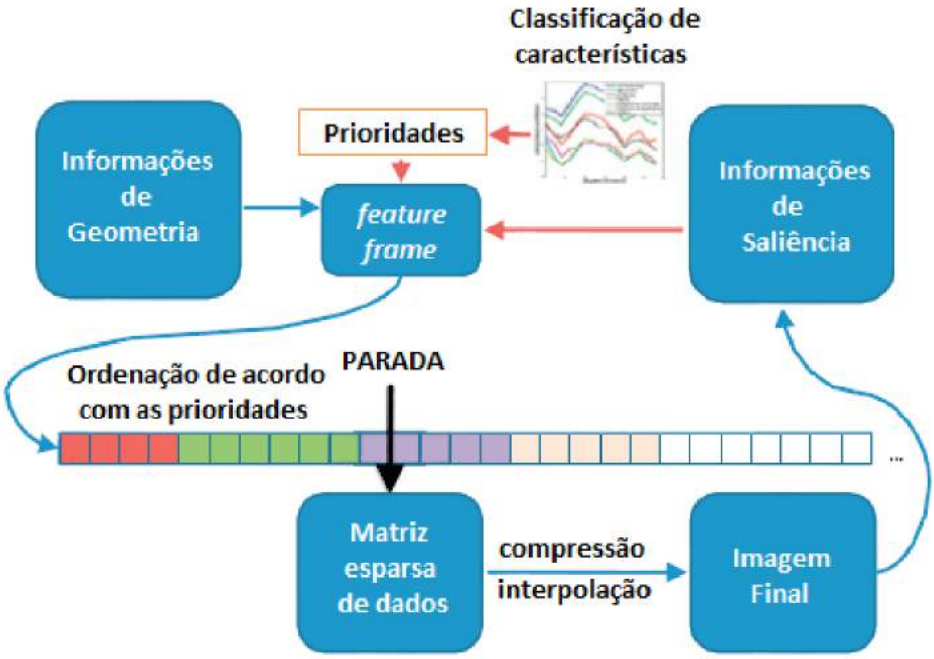

Figura 2.2: Processo de priorização de raios usando fila de prioridades proposto por Steinberger. Adaptado de [Steinberger et al. 2012].

A partir deste *buffer*, é definido um mapa de prioridades em espaço de tela para cada característica encontrada em espaço de objeto, tais como silhuetas, contornos sugestivos, vales e picos, e informação de saliência referentes a quadros anteriores. Este *buffer* é avaliado durante a passada de instanciação dos raios a serem traçados, e assim cada um deles pode assumir um valor de prioridade e ser ordenado conforme esse parâmetro. A fila de prioridade de raios é então processada até que um determinado limite de tempo seja atingido, e dessa maneira, uma taxa de quadros por segundo mínima pode ser garantida pelo algoritmo. O resultado dessa passada é uma matriz esparsa de dados obtidos pelo lançamento de cada raio através do ambiente, e sendo assim é necessária uma passada extra de compressão e construção da imagem final através de técnicas de interpolação. Finalmente, uma imagem final é gerada e sobre ela são aplicadas técnicas de detecção de saliências e características sugestivas para auxiliar a construção da fila de prioridade de raios do quadro seguinte. A Figura 2.2 ilustra as etapas deste processo.

Em contrate com essa técnica, o método proposto neste trabalho não produz um *buffer* esparso de dados, evitando processamento extra. Além disso, o método apresentado é determinístico e garante que, se necessário, pelo menos uma instância de raio seja atribuída à cada pixel. Os resultados dos raios traçados através da imagem são armazenados em endereços de memória adjacentes que são processados pela mesma *thread*, garantindo

coerência. Outra característica notável é que a abordagem proposta não impõe restrições quanto à construção das tabelas bases para a amostragem dos raios (*feature frames*). No *pipeline* apresentado no Capítulo 3, é mostrado que tanto as tabelas nos quais esses métodos baseiam sua amostragem quadro a quadro, como também o mapa de prioridades criado estão implícitos na estrutura e concepção do Mapa de Importâncias utilizado neste trabalho.

2.3 Ray Tracing Híbrido

Como a simulação de cada raio de luz em um espaço tridimensional da cena é impraticável dentro de condições normais, diversas abordagens vieram sendo criadas para diminuir a quantidade massiva de raios que precisam ser instanciados e disparados. A maior carga computacional do algoritmo de traçado de raios está justamente na grande quantidade de raios recursivos que precisam ser gerados após os primeiros raios atingirem a cena. Esses raios secundários podem ser utilizados para modelar diversos efeitos de luz, como sombras, reflexão simples e difusa, refração, transparência, entre outros. Para evitar a criação sucessiva de raios secundários, pode-se definir previamente o tamanho máximo da árvore de recursão do algoritmo e assim, estimar um limite superior de raios a serem traçados durante a síntese da imagem. Mesmo com isso feito, a carga computacional total do algoritmo pode ser bastante significativa dependendo da cena em questão, e nesses casos, é natural que sejam propostas alternativas para contornar a necessidade de processar toda a cena exaustivamente com o traçado de raios. Há diversas interpretações na literatura ao referenciar o uso de arcabouços de traçado de raios híbridos: algumas dividindo e distribuindo a carga total de trabalho a fim de dedicar menos tempo de processamento à região crítica do *pipeline*; e outras, restringindo o uso de algoritmos de alto custo computacional com o auxílio de técnicas secundárias de renderização, diminuindo assim o tamanho da região crítica da aplicação.

Utilizando técnicas de processamento híbrido entre núcleos da CPU e da GPU, pode-se obter ganhos de performance consideráveis dividindo os estágios tradicionais do algoritmo de traçado de raios entre esses núcleos, modelando-os na forma de tarefas independentes. No arcabouço desenvolvido por [Beck et al. 2005], a carga de trabalho foi dividida da seguinte forma: a primeira fase do algoritmo foi atribuída à GPU na forma de uma passada para geração de mapa de sombras, uma passada para identificação de geometria da cena, e finalmente uma passada para aplicação de efeito de borramento. Na passada de geometria, os números relativos aos triângulos dos modelos foram codificados como cores

RGB dentro de um *framebuffer*, assim como o resultado do mapa de sombras foi suavizado e escrito dentro do canal *alpha* do mesmo buffer. Através deste *framebuffer*, a CPU teria acesso aos dados necessários para realizar as passadas responsáveis para a geração dos efeitos de refração e reflexão, usando o algoritmo simples de traçado de raios. A etapa final do *pipeline* consiste numa aplicação da técnica de *Phong Shading* [Phong 1975], responsável por combinar os resultados das passadas anteriores numa imagem final.

Resultados consideráveis foram obtidos a partir da divisão do *pipeline* da geração da imagem final, deixando a síntese dos objetos a cargo de técnicas de rasterização, combinando com os efeitos de iluminação global simulados com o algoritmo de traçado de raios, adicionando-os à imagem base para atingir um maior patamar de qualidade. No arcabouço proposto por [Sabino 2012], a passada de geometria do algoritmo é dividida em seis etapas cujos resultados são armazenados em camadas distintas de um *framebuffer*. São eles: posição, normal, profundidade, cor difusa, cor especular e propriedades ópticas. Enquanto as cinco primeiras camadas consistem em informações relevantes para otimizar a rasterização da primeira etapa da cena, a última armazena propriedades da superfície tratadas pelo algoritmo de traçado de raios, como índice de refração, opacidade e expoente especular. A composição da imagem final permite que a imagem rasterizada seja incrementada de elementos gerados com a qualidade de um módulo traçador de raios para os objetos previamente determinados como de propriedades ópticas relevantes (Figura 2.3).

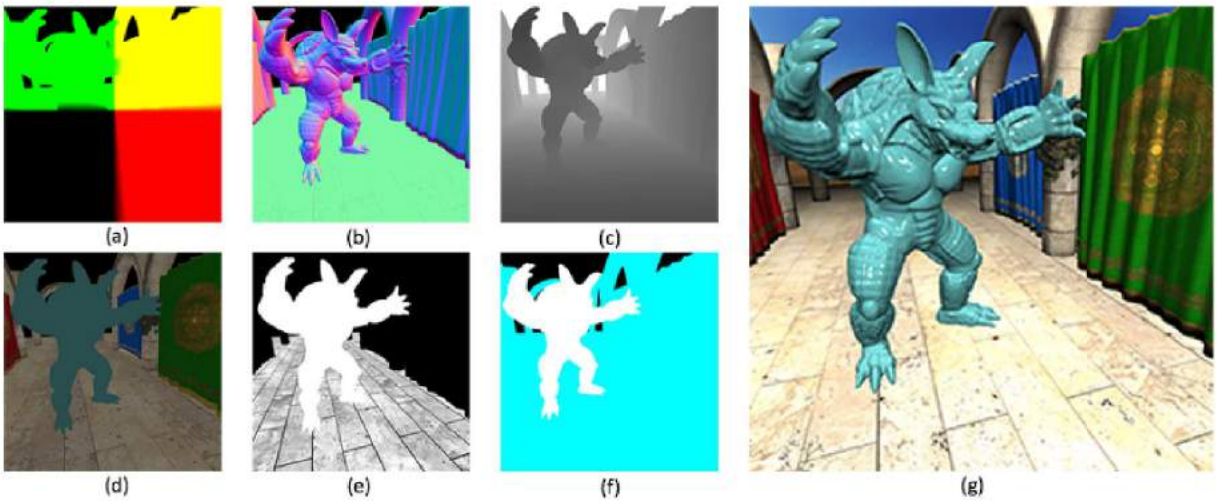

Figura 2.3: *Framebuffer* do *pipeline* híbrido proposto por [Sabino 2012]. O *framebuffer* é composto pelas camadas de: (a) posição, (b) normal, (c) profundidade, (d) cor difusa, (e) cor especular e (f) propriedades opticas. (g) é a imagem final composta após a fase de iluminação.

Seguindo essa linha de trabalho, foram desenvolvidas heurísticas para contornar o problema da discrepância entre objetos adjacentes avaliados de forma diferente para a

aplicação das propriedades ópticas: em certos casos, a adição desses efeitos de iluminação em parte da cena se torna prejudicial ao efeito final, chamando atenção para diferenças de qualidade perceptíveis na imagem. Um abordagem de aplicação do traçado de raio por região foi desenvolvida por [Martinez et al. 2012] a partir deste conceito, que aumentava o tamanho da janela de aplicação dos efeitos de iluminação do traçado de raios até que o orçamento de tempo estipulado estourasse. Ao fixar a taxa de quadros por segundo desejada, uma determinada porção da cena conseguia ser processada e em muitos casos, atingir um salto de qualidade na imagem final que a rasterização pura não conseguia. Entretanto, em determinados cenários onde a discrepância entre as duas abordagens é visível dentro de um mesmo objeto da cena, a estratégia em vez de aumentar a imersão do observador através do fotorrealismo, prejudica-a, chamando atenção para as diferenças entre as regiões com efeitos de iluminação adicionados pelo traçado de raios e as regiões sem ele. A Figura 2.4 ilustra este cenário.

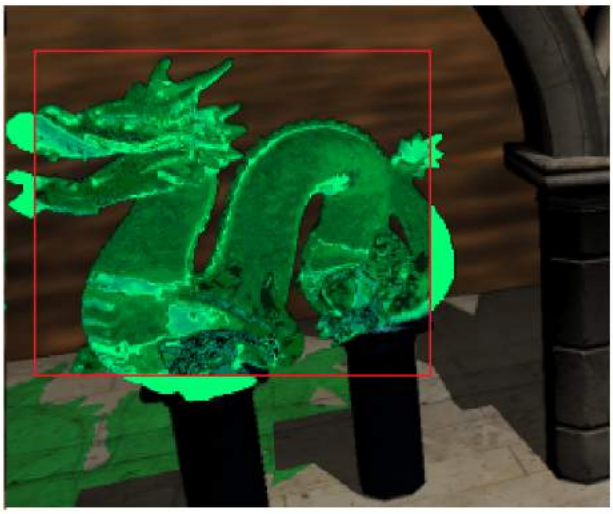

Figura 2.4: Discrepância entre duas abordagens diferentes de síntese de imagem em um ambiente do *framework* de [Martinez et al. 2012] onde o tamanho da janela de traçado de raios não é suficiente para aplicar os efeitos de iluminação por todo o modelo.

Este trabalho teve como inspiração o estudo desse fenômeno e foi desenvolvido, em suas primeiras instâncias, com a motivação de atenuar a discrepância visual obsevada na tentativa de melhorar uma imagem rasterizada, adicionando a ela efeitos de iluminação obtidos através do traçado de raios dentro de um limite insuficiente de tempo. O conceito de Mapa de Importâncias foi proposto como uma alternativa à janela de traçado de raios apresentada na Figura 2.4, visando identificar uma maneira mais adequada de distribuir os recursos disponíveis sem que o caráter de imersão ou o fotorrealismo da aplicação fosse prejudicado.

Capítulo 3

Método para Distribuição Não Uniforme de Raios e Composição da Imagem Final

Em linhas gerais, o algoritmo de traçado de raios é um algoritmo recursivo que projeta a partir do observador um raio através de cada um dos pixels que constituem o plano de imagem, em direção à cena em questão. Cada um destes raios, ao interagir com os objetos da cena, está sujeito a uma série de testes diversos: primeiramente, existe a necessidade de verificar qual dos objetos que o raio atravessa está mais próximo do observador e assim, o ponto de interseção do raio com a cena. Caso o raio não atinja nenhum objeto, o pixel em questão deve receber a cor do plano de fundo do ambiente. Caso contrário, a partir do ponto de interseção, é preciso levar em consideração a iluminação proveniente do resto da cena, já que a cor em questão pode ser uma combinação da iluminação direta ou indireta das diversas fontes de luz presentes no ambiente, além de poder conter componentes de luz refratada ou refletida de algum outro objeto. Para isso, o algoritmo traça três tipos de raios secundários, responsáveis por verificar cada uma dessas possibilidades. São eles: raios refletidos, raios refratados e raios de sombra.

O Algoritmo 1 compreende toda a computação necessária para determinar a cor de um pixel atravessado por um raio. Em uma aplicação de traçado de raios convencional, basta executá-lo para cada pixel do plano de imagem e a imagem final é gerada. No caso de um ambiente onde vários raios são traçados por pixel, é necessário que seja feita uma combinação da cor obtida por cada raio para gerar a cor final do pixel.

Algoritmo 1 Algoritmo clássico para calcular a cor de um pixel do plano de imagem a partir de um raio

Requer: $PixelAtual$, $Observador$
Assegurar: $CorPixel$

```
MenorDistancia ← ∞
ObjetoMaisProximo ← ∅
Cria um raio R a partir do Observador passando pelo PixelAtual
Para cada objeto da cena faça
    Se R intercepta o objeto atual e a distancia é menor que MenorDistancia então
        Atualiza MenorDistancia e ObjetoMaisProximo
    Fim Se
Fim Para
Se ObjetoMaisProximo = ∅ então
    Preenche PixelAtual com a cor do fundo
Se não
    Lança um novo raio para cada fonte de luz para verificar sombras
    Se a superfície é refletora então
        Lança um novo raio refletido baseado no ângulo de reflexão
    Fim Se
    Se a superfície é transparente então
        Lança um novo raio refratado baseado no ângulo de refração
    Fim Se
    Compõe a CorPixel a partir da cor resultante de todos os raios traçados
Fim Se
```

No modelo proposto por este trabalho, a partir de uma heurística bem definida, é gerado um Mapa de Importâncias responsável por definir a distribuição de recursos através do plano de imagem. Em seguida, é gerado um Mapa de Alocação de Raios, que quantifica a importância de cada região na forma de um número fixo de raios que deve atravessar cada pixel do plano de imagem. A próxima etapa consiste em criar um Mapa de Endereçamento de Raios, assim é possível redirecionar cada instância de raio gerada ao seu pixel correspondente. Dessa forma, pode-se gerar uma distribuição não uniforme através do plano de imagem e obter diferentes níveis de detalhamento em diferentes regiões da cena de uma maneira que seja conveniente. Ao disparar os raios primários, o processamento de cada raio individualmente é mantido o mesmo da abordagem convencional, com a diferença de que a imagem final já não pode ser obtida de forma direta. Como o endereçamento de cada instância de raio a um pixel da imagem é feita de maneira não homogênea, cada raio não sabe quantos outros participam da composição da cor do pixel que atravessa. Assim, é necessário armazenar o resultado de toda a amostragem em espaços de memória adjacentes, e em seguida, em uma etapa de pós processamento, identificar e combinar as cores obtidas pelas diferentes quantidades de raios referentes a um mesmo pixel em uma única cor da imagem final. A Figura 3.1 ilustra as etapas deste processo e as seções deste capítulo abordam em detalhes cada uma dessas etapas, incluindo o uso da técnica de imagem integral para que este processo possa ser executado de forma otimizada.

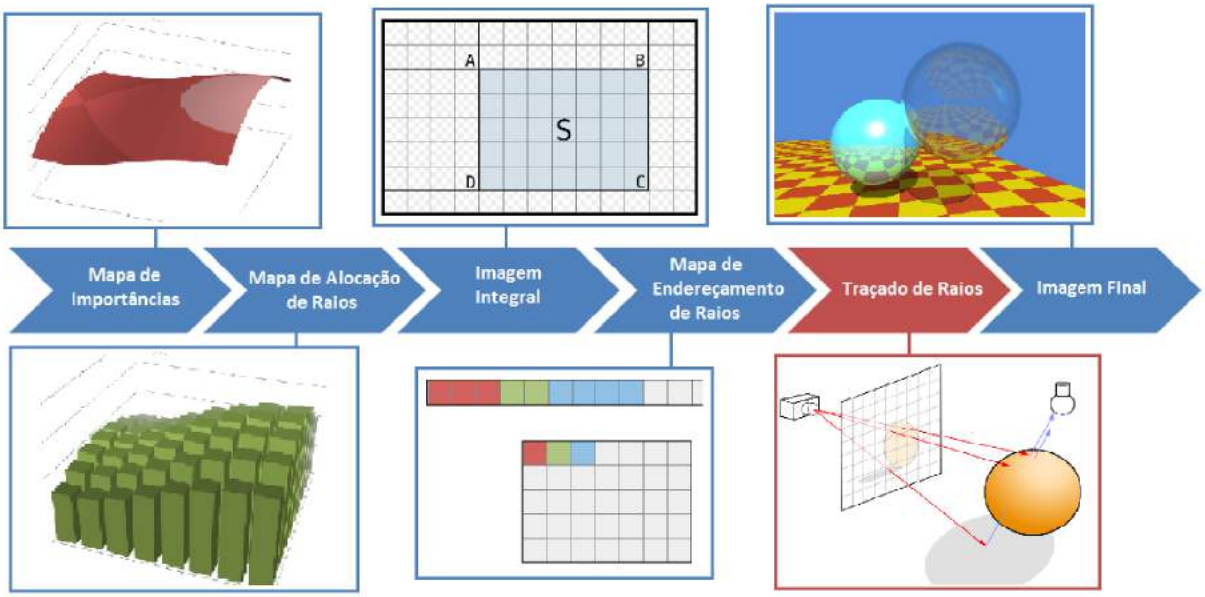

Figura 3.1: *Pipeline* para a síntese de uma imagem a partir do método proposto.

3.1 Construção do Mapa de Importâncias

A finalidade do *pipeline* proposto está em distribuir o processamento pela cena a partir do grau de impacto que cada região pode ter sobre a imagem final. Assim, as regiões ditas mais importantes deverão concentrar um maior número de raios e serem capazes de atingir um maior grau de detalhe, enquanto as regiões menos importantes naturalmente receberão menos destaque. Para que isso seja feito, é preciso de uma estrura que indique o grau de importância de cada região do plano de imagem, e assim informar às próximas etapas do *pipeline* como organizar e distribuir seus recursos. Esta estrutura é chamada de Mapa de Importâncias.

O Mapa de Importâncias é construído baseado numa função heurística h, que tenha seu domínio sobre o plano de imagem e para cada valor dele, retorne um valor de importância. São variações da função h quaisquer funções tal que

$$h : \mathbb{R}^2 \to \mathbb{R}, \tag{3.1}$$

onde $h(x, y) \geq 0$. Sendo x e y coordenadas de um ponto no plano de imagem.

A função pode retornar um valor positivo quantificando o quanto de importância cada pixel compreende, ou até zero, caso seja interessante que aquela célula não tenha raios disparados através dela. Isso faz sentido, por exemplo, num ambiente híbrido, onde queira-se traçar raios sobre uma cena rasterizada apenas para ressaltar áreas com determinados propriedades ópticas. Um cenário como esse poderia ser modelado com uma função heurística que assumisse valores positivos para áreas onde existissem objetos reflexivos e transparentes e que fosse zero para superfícies opacas. O Capítulo 4 aborda em mais detalhes as diferentes funções heurísticas criadas durante o desenvolvimento deste trabalho. São elas funções definidas a partir dos conceitos de rasterização das propriedades ópticas da cena antes do traçado de raios e, identificação de saliências em cenas tridimensionais.

O Mapa de Importâncias é uma estrutura do tipo

$$I_{map} : \mathbb{Z}^2 \to \mathbb{R},$$

cujo domínio consiste dos pixels da imagem e o contra-domínio, do valor de importância médio de h no pixel (i, j). Sendo assim, o Mapa de Importâncias é criado a partir da discretização da função heurística h e é definido como

$$I_{map}(i, j) = \frac{1}{\Delta x \Delta y} \int_{i}^{i+\Delta y} \int_{j}^{j+\Delta x} h(x, y)\, \mathrm{d}x\, \mathrm{d}y, \tag{3.2}$$

onde $0 \leq i < H$ e $0 \leq j < W$ para $(i, j) \in \mathbb{Z}^2$. Sendo H e W a altura e a largura da imagem a ser renderizada. Δx e Δy correspondem, respectivamente, à largura e à altura de um pixel.

Vale ressaltar que a notação utilizada nessa dissertação endereça grandezas inteiras, como elementos das estruturas criadas ou o par coordenado que representa o índice dos pixels do plano de imagem a partir do seu valor de linha i e coluna j. Já para grandezas reais, como pontos no plano de imagem, ou para representar o espaçamento entre o centro dos pixels e sua borda, são utilizados o espaçamento horizontal x e vertical y, sempre em ordem alfabética.

As Figura 3.2 ilustra o conceito de zonas de importância em um plano de imagem. Nela, as zonas mais importantes estão representadas pela cor vermelho, são elas as que apresentam efeitos de refração e inter-reflexão. Regiões menos importantes estão representadas pela cor laranja e em seguida, pela cor amarela, verde, ciano e azul. Essa disposição de cores é um recurso usado para ilustrar o conceito, porém, curiosamente, o comprimento de onda das cores da imagem a direita pode ser utilizado como uma heurística válida para a criação de um mapa de importâncias para a imagem a esquerda da figura, já que as zonas em vermelho (com o maior comprimento de onda) estão dispostas nas regiões que demandam mais processamento, e as em azul (com o menor comprimento), nas que demandam menos.

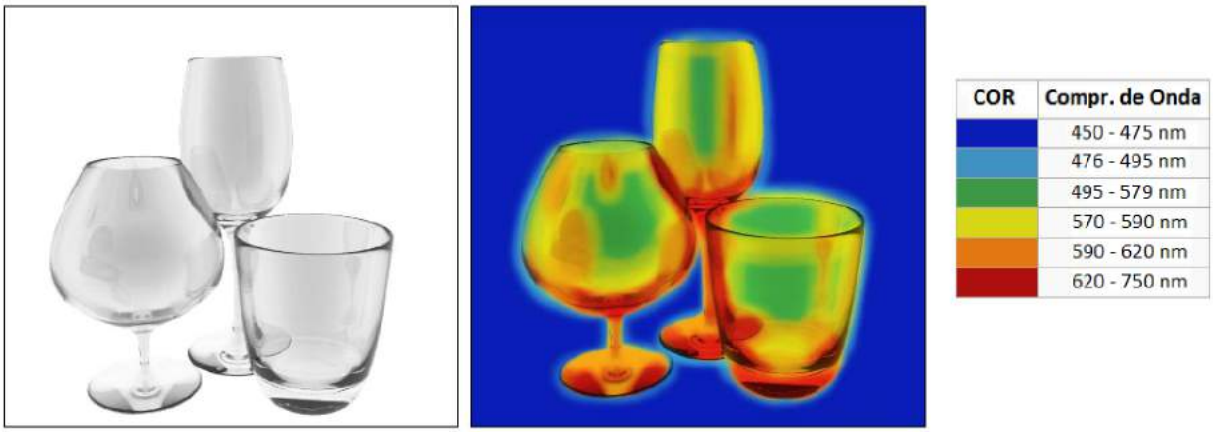

Figura 3.2: Zonas de importância sobre uma imagem gerada por traçado de raios: à esquerda, a imagem original e, à direita, uma possível configuração de como os valores de importância podem ser atribuídos a cada região.

O Mapa de Importâncias é a estrutura inicial da abordagem proposta neste trabalho para renderizar uma cena utilizando o algoritmo de traçado de raios baseado em zonas de importância. Sendo assim, a construção das estruturas subsequentes dependem da viabilidade e da aplicabilidade deste mapa. Um Mapa de Importâncias suficientemente

bom é capaz de ditar conforme os recursos disponíveis devem ser distribuídos com precisão e assim, adequar o processamento de cenas com um alto teor de complexidade a um ambiente com restrições rígidas de quadros por segundo. É importante ressaltar que, respeitando as definições formais, ele pode ser construído de diversas formas: de maneira automática como resultado da aplicação de uma função matemática sobre um domínio 2D; a partir do resultado de um *shader* sobre a cena em questão, o que agrega um caráter adaptativo ao algoritmo; a partir de um algoritmo de predição de movimentos, dentre outros. Como qualquer algoritmo informado baseado no uso de heurísticas, a aplicabilidade do método aqui proposto é diretamente dependente do quão adequado o Mapa de Importância é em relação à cena.

3.2 Construção do Mapa de Alocação de Raios

A segunda etapa do *pipeline* consiste em, a partir da informação de importância para cada região do plano de imagem, determinar a parcela de processamento que vai ser destinada a ela. Em um ambiente de tempo real, para que se possa manter uma taxa aceitável de quadros por segundo durante a execução da aplicação, é preciso que se respeite um orçamento fixo de processamento. Fazendo uma estimativa do número de raios que o algoritmo é capaz de processar sem que as restrições de tempo da aplicação sejam comprometidas, e normalizando a importância acumulada no Mapa de Importâncias da etapa anterior, pode-se traduzir cada valor de importância em informação de quantos raios podem ser traçados através de cada pixel do plano de imagem. Esta informação é calculada e armazenada no Mapa de Alocação de Raios.

O Mapa de Alocação de Raios é uma estrutura semelhante ao Mapa de Importâncias, do tipo

$$A_{map} : \mathbb{Z}^2 \rightarrow \mathbb{N}^0,$$

cujo domínio, assim como o Mapa de Importâncias, compreende os pixels da imagem final sobre o plano de imagem. O seu contradomínio, em vez de valores reais, é formado por valores naturais ou zero, já que representa o número de raios a serem traçados por cada pixel.

O Mapa de Alocação de Raios é definido como

$$A_{map} = \lceil \hat{I}_{map}(i,j)\, K \rceil, \tag{3.3}$$

onde I_{map} é o Mapa de Importâncias definido na Equação 3.2, K é uma constante que re-

presenta o número máximo de raios que podem atravessar um determinado pixel do plano de imagem e $\hat{I}_{map}(i,j)$ é o Mapa de Importâncias normalizado levando em consideração a importância total acumulada no plano de imagem. A ideia por trás da construção do Mapa de Alocação de Raios é distribuir proporcionalmente de uma maneira discreta esta importância e escalá-la em K.

Vale ressaltar que, diferentemente da etapa anterior, cada valor do Mapa de Alocação de Raios depende não somente do valor pontual de uma função de entrada, mas de toda a distribuição desta função sobre o seu o domínio. Além disso, o uso da função teto ($\lceil\,\rceil$) na definição do mapa se dá para que excetuando-se casos onde a importância do pixel seja absolutamente nula, seja destinado pelo menos um raio para cada pixel, de forma a evitar regiões sem rendering que seriam naturalmente geradas a partir de Mapas de Importância com muita discrepância na sua distribuição. Como $\hat{I}$ encontra-se sempre dentro do intervalo $[0,1]$, garante-se, por definição, que cada célula terá no máximo K raios incidentes. O valor ótimo de K pode ser alcançado de forma empírica ou adaptativa, conforme a cena em questão for processada.

3.3 Cálculo da Imagem Integral

Até então, foram introduzidas duas novas estruturas de dimensões idênticas à imagem a ser renderizada, e neste ponto do *pipeline*, a partir delas, pode-se obter em tempo constante informações sobre a importância absoluta de cada pixel do plano de imagem e quantidade de raios que devem atravessá-lo. Naturalmente, dentro do escopo de aplicações em tempo real, se faz necessário assegurar que as operações tanto sobre essas estruturas como também a quaisquer outras subsequentes sejam otimizadas a uma complexidade compatível com as restrições de tempo. Além disso, para tirar vantagem do caráter paralelo do algoritmo de traçado de raios, toda a operação tanto de pré como de pós-processamento deve ser também paralelizável em GPU.

Durante o algoritmo de traçado de raios, é preciso estabelecer um endereçamento de cada instância de raio gerada a um pixel do plano de imagem. Em cenários onde a quantidade de raios atravessando cada um desses pixels é uniforme, esse endereçamento pode ser feito de forma trivial dentro da GPU, já que toda *thread* de processamento está indexada na GPU a partir de um desvio fixo do seu pixel correspondente, e então é possível calcular em tempo constante a partir do seu índice, através de qual posição do plano de imagem o seu raio correspondente deve ser disparado. Em um ambiente onde a

quantidade de raios a serem traçados pode variar de pixel a pixel da imagem, para que esse endereçamento possa ser feito de maneira eficiente, é preciso que se saiba quantos raios estão destinados a cada um dos pixels anteriores. Ingenuamente, pode-se varrer todo o Mapa de Alocação de Raios para obter tal informação, até que se chegue no índice do pixel atual. Esta estratégia, porém, é impraticável devido a sua alta complexidade. Para resolver esta questão, nesta etapa do *pipeline* proposto é introduzido o cálculo da Imagem Integral sobre o Mapa de Alocação de Raios.

A técnica da Imagem Integral (também chamada de *Summed Area Table*, em inglês, ou simplesmente SAT) foi introduzida por [Crow 1984] para tratar computações de mapa de textura. Durante a computação da técnica de *Mip Mapping* [Williams 1983] em um nível de textura, se fazia necessário o cálculo da média da intensidade de todos os pixels de interesse contidos em um determinado retângulo no nível superior. A média dessas intensidades é facilmente obtida dividindo a soma delas pela área do retângulo, e para otimizar esse processo, Crow mostrou que o cálculo dessa soma poderia ser feito em tempo constante em um retângulo arbitrário com apenas uma operação de adição e duas subtrações.

A Figura 3.3 ilustra como pode-se amostrar a integral de uma região retangular arbitrária de uma imagem. Seguindo esse princípio, para calcular o retângulo de interseção da área compreendida entre x_l e x_r com a área entre y_b e y_t, basta fazer

$$T(x_r, y_t) - T(x_r, y_b) - T(x_l, y_t) + T(x_l, y_b),$$

onde $T(x, y)$ é o valor da tabela endereçado pelo par de coordenadas (x, y). Simplificadamente, a partir do canto superior direito do retângulo, subtrai-se toda a área sobressalente abaixo dele e, em seguida, toda a área a sua esquerda. Como a interseção entre essas duas áreas foi subtraída duas vezes, para chegar à área esperada, é preciso adicioná-la novamente, uma única vez.

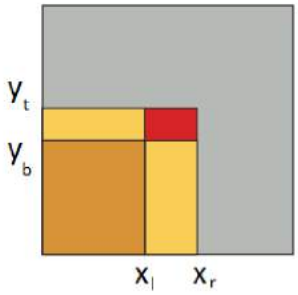

Figura 3.3: Amostragem da integral de uma região retangular a partir de suas coordenadas.

Como a SAT deve acumular o valor da soma de todos os elementos abaixo e à esquerda de um elemento qualquer, a SAT sobre um elemento (i, j) calculada sobre uma matriz m qualquer é definida como

$$SAT(i,j) = \sum_{k=0}^{i} \sum_{l=0}^{j} m(k,l). \tag{3.4}$$

O cálculo da SAT pode ser feito eficientemente de forma recursiva, já que para calcular o somatório até um determinado elemento (i, j) da matriz é preciso que se calcule todas as somas abaixo e à esquerda dele. Na prática, esta propriedade permite que seja construída uma tabela de somas parciais sequencialmente acessando apenas uma vez cada elemento de uma matriz qualquer. Em aplicações paralelas, este encadeamento de dependências pode ser contornado através de algoritmos de redução.

Quando aplicada ao Mapa de Alocação de Raios definido na Seção 3.2, essa técnica permite que seja construída uma tabela da dimensão da imagem em questão, cujos valores representam a quantidade total de raios escalonados para serem traçados até cada pixel, considerando que a origem do sistema se mantenha fixa. A partir das informações de quantos raios devam ser traçados por cada pixel e do total de raios escalonados desde a origem até o pixel em questão, utilizando as propriedades da SAT, pode-se fazer o endereçamento de raios de maneira não uniforme em tempo constante dentro da GPU. Este procedimento será abordado com mais detalhes na Seção 3.4.

O cálculo da SAT nesta etapa do *pipeline* é feito da seguinte forma:

$$SAT(i,j) = \begin{cases} 0 & \text{se } i \leq 0 \text{ ou } j \leq 0 \\ A_{map}(i,j) + SAT(i-1,j) & \\ \quad +SAT(i,j-1) - SAT(i-1,j-1) & \text{caso contrário.} \end{cases} \tag{3.5}$$

Para construir a SAT de maneira eficiente na GPU, assim como em [Hensley et al. 2005], é utilizada a técnica de *Recursive Doubling* introduzida por [Dubois and Rodrigue 1977]. A estratégia para a construção é calcular a soma parcial da tabela linha a linha, e em seguida, coluna a coluna, obtendo finalmente o valor de todo o somatório dos elementos acima e a esquerda para cada uma das células. Para calcular a soma parcial das linhas (e analogamente das colunas), o algoritmo de *recursive doubling* inicialmente soma a cada elemento, o valor do elemento à sua esquerda. Na segunda passada, cada elemento das linhas é incrementado do valor do elemento que está a duas posições à sua esquerda. A cada passada do algoritmo esse desvio é dobrado. Em $O(\log_2 n)$ passadas, sendo n o número de elementos da linha, cada um dos elementos armazenará a soma de todos os outros à

sua esquerda. A Figura 3.4 ilustra a execução desse algoritmo sobre uma linha. Construir a SAT através desse método demanda exatamente $\lceil \log_2(W) \rceil + \lceil \log_2(H) \rceil$ passadas na GPU.

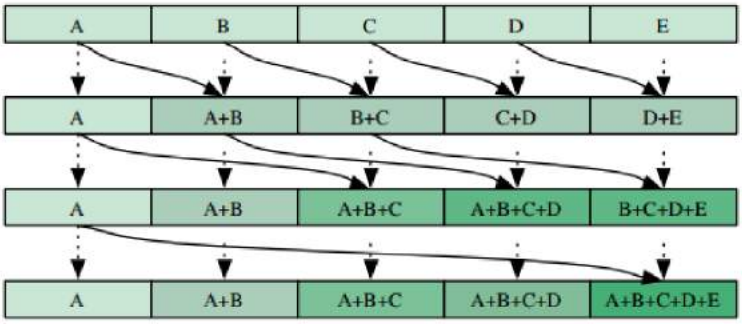

Figura 3.4: Uso da técnica de *recursive doubling* para o cálculo de um conjunto de somas parciais. Extraído de [Hensley et al. 2005].

A motivação para a construção da SAT é otimizar a computação do *pipeline* de síntese da imagem do começo ao fim, mantendo-o numa complexidade apropriada. Embora algoritmos paralelos de busca na GPU, como o *scan*, possam ser usados em etapas posteriores do processo para obter a mesma informação que a SAT é responsável por armazenar, cada acesso de tempo constante da informação na SAT, tomaria tempo logarítmico para ser executado caso essa etapa não existisse. Sendo assim, o *pipeline* em questão é projetado com a adição da SAT para ser processado em $O(\log_2 W + \log_2 H)$ operações neste estágio, mantendo a tabela em memória, para que essa computação seja economizada nas etapas subsequentes. Vale ressaltar que a partir desta etapa, toda a computação acontece em espaço de raios, em vez de em espaço de imagem. Dessa forma, assumindo uma distribuição K-uniforme de raios, ou seja, com K raios disparados por cada pixel do plano de imagem, a criação da SAT não somente otimiza o processamento fazendo o acesso constante de um cálculo que seria feito em tempo logarítimico em duas etapas posteriores (Seções 3.4 e 3.6), como também proporciona um *speed-up* no sistema por um fator de K, já que cada uma dessas passadas exigiria $K\,H\,W$ buscas.

Em tempo, a descrição da Imagem Integral feita nesta seção visa ilustrar o conceito desta etapa do *pipeline* e sendo assim, não foram abordados detalhes de implementação. Esta técnica pode ser aprimorada de uma forma muito mais rebuscada visando atingir máxima performance, porém esta é uma discussão mais extensa que foge do objetivo deste capítulo. Atualmente, a implementação mais rápida da técnica de Imagem Integral é a proposta por [Nehab et al. 2011]. Novamente, o foco desta seção é descrever a importância do cálculo do acumulo dos raios a cada ponto do plano de imagem, e sendo

assim, de implementação para implementação, pode-se escolher tanto utilizar a maneira mais sofisticada de fazê-lo, visando maior performance e usabilidade, ou maneiras mais simples, que agregam praticidade e clareza de implementação. Quanto à essa questão, o uso da Imagem Integral nesta etapa pode ser adaptado em uma estrutura unidimensional de somas parciais e uma série de buscas logarítimicas consecutivas, sem aumento imediato de complexidade. A comparação das diversas maneiras de executar esta etapa do *pipeline* foge do escopo deste trabalho e não será aprofundada.

3.4 Definição do Mapa de Endereçamento de Raios

Depois de calculada a SAT sobre o Mapa de Alocação de Raios, é possível fazer o endereçamento de cada raio instanciado a uma posição do plano do imagem de maneira não uniforme, isto é, associar cada índice de raio a um pixel a partir do número de raios que devem atravessar cada um deles. Para armazenar essa informação é introduzida uma estrutura chamada de Mapa de Endereçamento de Raios, do tipo

$$T_{map} : \mathbb{N}^0 \rightarrow \mathbb{Z}^2,$$

cujo domínio consiste no índice de cada raio a ser lançado, e com o auxílio do número de raios escalonados para cada pixel da imagem, retorna para cada valor desse domínio um par de inteiros representando o pixel da imagem que receberá cada raio. O Mapa de Endereçamento de Raios pode ser encarado como um mapa de alvos (*targets*, em inglês) para cada raio disparado e pode ser construído a partir do Algoritmo 2.

Algoritmo 2 Construir o Mapa de Endereçamento de Raios

Requer: SAT, A_{map}, W, H
Assegurar: T_{map}

1: **Para** $i = 0$ até $H - 1$ **faça**
2: **Para** $j = 0$ até $W - 1$ **faça**
3: $NumAcumRaios \leftarrow SAT(i-1, W-1) + SAT(i, j-1) - SAT(i-1, j-1)$
4: $NumAtualRaios \leftarrow A_{map}(i, j)$
5: **Para** $l = 0$ até $NumAtualRaios$ **faça**
6: $T_{map}(NumAcumRaios + l) \leftarrow (i, j)$
7: **Fim Para**
8: **Fim Para**
9: **Fim Para**

Diferentemente das outras estruturas definidas até então, o Mapa de Endereçamento de Raios não tem seu domínio sobre o plano de imagem. Por ser responsável por armazenar o pixel alvo para cada raio a ser disparado, ele pode ser definido como uma estrutura

unidimensional desde que seja capaz de associar pares de coordenadas a todos os raios a serem criados.

O Algoritmo 2 faz uma varredura em todo o plano de imagem e para cada pixel, verifica o número de raios escalonados para serem disparados através dele. Isso é feito de forma paralela durante o desenrolar dos laços nas Linhas 1 e 2. O número de raios escalonados para o disparo através de cada pixel é representado por $NumAtualRaios$ na Linha 4. Em seguida, o algoritmo calcula o número de raios acumulado, $NumAcumRaios$, que estão escalonados para serem disparados até o pixel anterior. Nesse caso, como os laços de disparo de *threads* estão dispostos primeiramente por colunas e em seguida por linhas, o pixel chamado de anterior de um determinado pixel (i, j) é o pixel $(i, j-1)$. Para calcular $NumAcumRaios$, é preciso somar o número de raios acumulados até o final da linha de baixo $SAT(i-1, W-1)$, com o número de raios da linha atual até a coluna anterior $SAT(i, j-1)$, e finalmente subtrair a região entre a origem e a posição abaixo e a esquerda do pixel atual, que foi somada duas vezes. A Figura 3.5 ilustra esse cálculo.

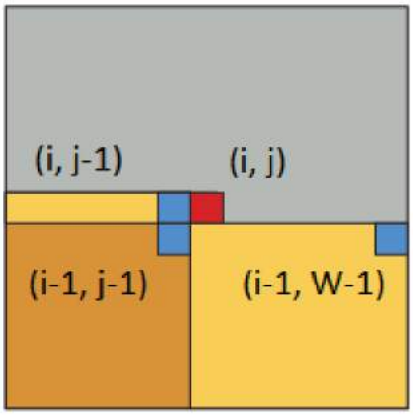

Figura 3.5: Cálculo do número acumulado de raios escalonados para disparo até o pixel (i, j). Linha 3 do Algoritmo 2.

Durante a execução do algoritmo, o número acumulado de raios é necessário, pois sem esta informação, todo laço de repetição do algoritmo depende do anterior tornando-o inviável sua execução paralela na GPU. Sabendo o número de raios acumulados a cada etapa, pode-se usar esse valor como um *offset* para o índice de T_{map} e assim calculá-lo em tempo $O(K)$, utilizando HW *threads*. Isto é feito na Linha 6, que será executada no máximo K vezes, sendo K o número máximo de raios a atravessarem um único pixel.

3.5 Traçado de Raios com Distribuição Não Uniforme

Após a construção do Mapa de Endereçamento de Raios, cada raio a ser criado já pode ser distribuído de maneira conveniente através do plano de imagem. Sendo assim, a etapa seguinte consiste na execução do algoritmo de traçado de raios em paralelo, corrigindo a posição de disparo de cada raio r instanciado para que atravesse seu pixel alvo, armazenado em $T_{map}(r)$. Como existem vários raios disparados no centro do mesmo pixel, para cada um deles, faz-se também necessária a adição de uma componente estocástica $(\Delta i, \Delta j)$ a ele para aprimorar a amostragem na região referente ao pixel, e adicionalmente evitar *aliasing* na imagem gerada. Esta perturbação na direção dos raios deve ser pequena o suficiente para que estes não saiam da região do seu pixel alvo, e gerada aleatoriamente a partir de uma distribuição uniforme. A Figura 3.6 ilustra este processo.

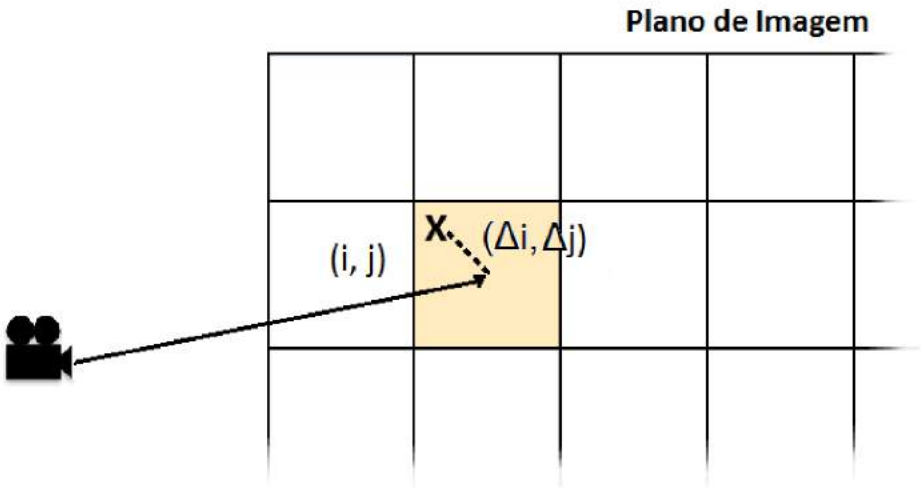

Figura 3.6: Traçado de raios sobre o plano de imagem partindo da câmera em direção ao centro do pixel (i, j). Alterando ligeiramente a direção do raio a partir de um deslocamento de $(\Delta i, \Delta j)$ no plano de imagem, garante-se que o conjunto de raios direcionados a um mesmo pixel atravessem a cena por caminhos diferentes, e assim, agreguem informação adicional à imagem.

Após a execução do algoritmo, cada instância de raio retorna com uma informação de cor, armazenada em um *buffer* simples que pode ser endereçado célula a célula a partir do Mapa de Endereçamento de Raios. Este *buffer* de saída é do tipo

$$O : \mathbb{N} \rightarrow \mathbb{R}^4,$$

armazenando para cada índice de raio do seu domínio, quatro números reais representando os valores dos canais R, G, B e A obtidos a partir do seu traçado sobre a cena. Na etapa

final do processo, este *buffer* será utilizado na composição da imagem final.

3.6 Composição da Imagem Final

Até então, todas as estruturas criadas e algoritmos executados levaram à criação de um *buffer* contendo para cada índice de raio criado e traçado através do plano de imagem um valor de cor amostrado na cena. Sendo assim, este *buffer* segue o mesmo formato do Mapa de Endereçamento de Raios, e é preciso que a informação contida nele seja traduzida para o formato de uma imagem com largura W e altura H. Para que isso seja feito, é necessário que se obtenha a informação de quais índices do *buffer* correspondem a um mesmo pixel e realizar uma operação de composição das cores associadas a eles.

Na Seção 3.4, durante a construção do Mapa de Endereçamento de Raios, a informação sobre o escalonamento de raios contida em espaço de imagem foi traduzida em informação de direção para cada disparo, e armazenada em espaço de raios. Nesta etapa, para a composição da imagem final, deseja-se realizar a operação inversa: a partir de uma estrutura em espaço de raios, o *buffer* de saída O, instanciar os elementos de outra, um quadro da aplicação F em espaço de imagem. O Algoritmo 3 utiliza as estruturas definidas nas seções anteriores para compor a imagem final, objetivo deste *pipeline*.

Algoritmo 3 Compor a imagem final

Requer: O, SAT, A_{map}, W, H
Assegurar: F
1: **Para** $i = 0$ até $H - 1$ **faça**
2: **Para** $j = 0$ até $W - 1$ **faça**
3: $NumRaios \leftarrow A_{map}(i, j)$
4: $offset \leftarrow SAT(i-1, W-1) + SAT(i, j-1) - SAT(i-1, j-1)$
5: **Para** $l = 0$ até $NumRaios$ **faça**
6: $CorRaio \leftarrow CorRaio + O(offset + l)$
7: **Fim Para**
8: $F(i, j) \leftarrow CorRaio/NumRaios$
9: **Fim Para**
10: **Fim Para**

A estratégia do algoritmo é: para cada pixel a ser composto na imagem final, verificar quantos raios estão associados a ele e a partir desta informação, combinar as cores de cada posição de O relativa a esses raios em uma cor resultante. Assim como no Algoritmo 2, utilizando a quantidade acumulada de raios traçados até cada pixel como um *offset* (Linha 4), pode-se tornar o processamento de cada laço independente dos demais. Processando o algoritmo de forma paralela na GPU, cada *thread* é capaz de acessar o índice

do primeiro raio disparado para um determinado pixel em O e utilizando a informação de quantas posições de memória subsequentes são pertinentes à computação daquela cor, cada *thread* garante que seu processamento acontece em uma região exclusiva do *buffer* de saída, sem interferir no processamento das outras. Dessa forma, dispondo de uma *thread* para processar cada um dos HW pixels, a imagem final pode ser composta em tempo $O(K)$, onde K é o número máximo de raios incidentes a um único pixel. Com o auxílio do Mapa de Alocação de Raios e da SAT, o endereçamento entre o plano de imagem e o *buffer* de saída é feito em tempo constante, e esta complexidade é obtida em função da operação de combinar até K cores em um único pixel da imagem final. Esta operação é feita entre as Linhas 5 e 8. A Figura 3.7 ilustra todo o processo. Vale ressaltar, que caso K seja definido empiricamente e não dependa da entrada do algoritmo, toda a computação feita neste passo do algoritmo acontece em tempo constante.

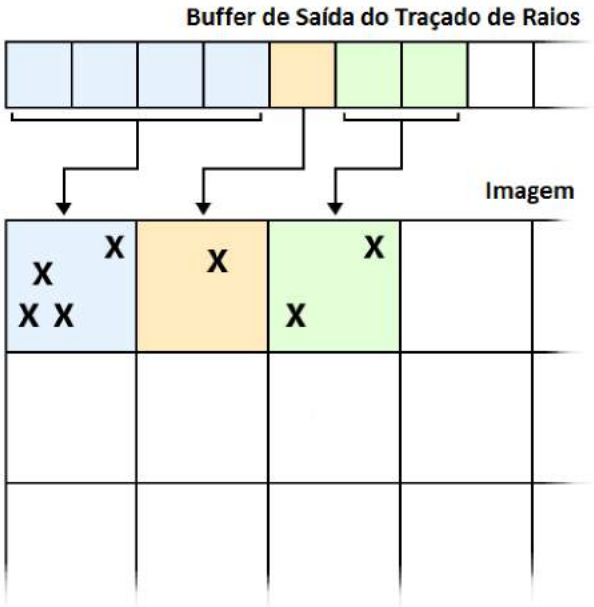

Figura 3.7: Reconstrução da imagem final a partir de um Mapa de Traçado de Raios

Capítulo 4

Heurísticas para a Construção de Mapas de Importância

Sempre que um algoritmo é aplicado para resolver um problema, espera-se que pelo menos duas condições sejam atendidas: que esse algoritmo tenha um tempo aceitável de execução; e que a solução encontrada, se não ótima, seja boa o suficiente para resolver o problema em todos os casos. Naturalmente, a primeira propriedade, por mais que seja sempre encorajada, pode ser relaxada em sistemas que não demandam uma resposta em tempo real. Enquanto isso, a segunda precisa ser atendida para que de fato o algoritmo, ou a técnica em questão, sejam realmente efetivos.

De frente a um problema difícil, ou que apresente restrições de tempo e recursos que limitem as suas soluções viáveis, como no caso do problema de síntese de imagens por traçado de raios em tempo real, alguns caminhos podem ser explorados para que se chegue a uma solução adequada. Uma das estratégias de busca de melhores soluções consiste em métodos informados, onde se há uma previsão de qualidade da solução final que indique qual seria a melhor forma de refinar a solução atual para que se chegue no objetivo. Podemos utilizar uma *função heurística* para servir de métrica para essa previsão. Uma heurística é uma estratégia para resolver ou aprender sobre um problema baseada em informações acessíveis porém, não necessariamente ótimas ou perfeitas [Pearl 1983]. Na Seção 3.1, foi definida uma função heurística h adequada ao escopo deste trabalho.

Partindo do princípio de que uma imagem ou cena tridimensional chama a atenção do observador de formas diferentes, é possível concluir que, tendo uma quantidade finita de processamento, a imagem ótima que pode ser gerada é aquela na qual os aspectos com maior impacto visual estejam renderizados com qualidade, sem que os demais sejam negligenciados a ponto de atrair a atenção do observador negativamente. Este é um

problema difícil e a solução ótima varia não apenas de ambiente para ambiente, mas também de quadro para quadro, dependendo da movimentação da câmera ou dos objetos da cena. Sendo assim, é virtualmente impossível o desenvolvimento de um algoritmo que não utilize nenhuma informação da cena e seja capaz de encontrar previamente uma solução ótima para todos os possíveis cenários. Um sistema como o desenvolvido neste trabalho, que gerencia a forma como os raios são traçados através do plano de imagem, se beneficia muito do uso de heurísticas, e pode se adaptar conforme necessário para que atinja uma solução mais próxima da ótima, apesar das restrições do ambiente.

No decorrer deste capítulo, serão abordadas duas heurísticas que podem ser usadas para gerar um Mapa de Importâncias suficientemente bom para a aplicação da técnica proposta. São elas:

- Utilizando uma renderização simples e computacionalmente barata da cena, estimar, através das propriedades ópticas de cada objeto a compõe, as regiões da imagem final onde se concentram o maior número de reflexões, refrações, espalhamento de luz, etc. que provavelmente serão valorizadas durante a aplicação do algoritmo de traçado de raios (Seção 4.1).
- Tirando vantagem do fato de que o foco de atenção do observador é naturalmente seletivo, aplicar sobre a imagem gerada no quadro anterior, técnicas de processamento de imagem como detecção de saliências, para que sejam priorizadas, no plano de imagem atual, as regiões com maior tendência de atrair a atenção do observador. (Seção 4.2).

Neste trabalho, consta a demonstração de que as heurísticas propostas são admissíveis e podem ser integradas ao *pipeline* descrito no Capítulo 3 sob a forma de Mapas de Importância.

4.1 Rasterização de Propriedades Ópticas

Para desvendar as regiões visualmente impactantes na imagem final, uma boa estratégia é verificar primeiramente as regiões que concentram uma maior riqueza de detalhes de iluminação. Pode-se falar, com alguma propriedade, que objetos altamente reflexivos, ou regiões onde a luz tem um comportamento não usual como o de refração, chamam a atenção do observador para aquela porção da imagem. Além dessas regiões naturalmente costumarem a requerer mais raios para a correta simulação das propriedades que

apresentam, espera-se sempre que tenham um comportamento fisicamente plausível numa imagem renderizada por computador, e a falta de tal comportamento ou a baixa qualidade na sua simulação são responsáveis por atrair atenção negativamente, comprometendo a qualidade visual de todo o sistema. Em certas aplicações, como jogos eletrônicos, aonde o caráter imersivo do sistema é de grande importância, é fundamental que a qualidade visual se mantenha nessas regiões. Sendo assim, a ideia de um *buffer* no formato do plano de imagem armazenando as propriedades ópticas da cena, apresenta-se naturalmente como uma heurística válida para a obtenção de um Mapa de Importâncias.

Recentemente, [Sabino 2012] propôs um *framework* baseado na ideia de um *pipeline* híbrido de traçado de raios, composto de uma etapa de pré processamento a partir da técnica de *Deferred Shading* [Deering et al. 1988] e uma etapa de traçado de raios para adicionar efeitos de iluminação sobre a imagem rasterizada. A técnica de *Deferred Shading* consiste em, através de um *buffer* de geometria (*Geometric Buffer* ou *G-Buffer*) de várias camadas, executar todos os testes de visibilidade da cena antes de qualquer processo de iluminação. Para que isso possa ser feito, este *buffer* deve conter, camada a camada, as informações necessárias para cada pixel: posição, normal, profundidade, cor de albedo e componente especular do material. Todas essas informações são computadas utilizando rasterização. Em um ambiente híbrido, entretanto, o *G-Buffer* pode ser estendido em uma camada, com o objetivo de armazenar propriedades extras que seriam pertinentes apenas no estágio de iluminação do seu *pipeline*, cujo processamento é feito utilizando o traçado de raios. Um exemplo de *G-Buffer* pode ser visto na Figura 2.3.

Neste trabalho, a utilização de um *pipeline* híbrido de rasterização e traçado de raios foi empregada no desenvolvimento de uma heurística para a construção de um Mapa de Importâncias. A estratégia desta heurística envolve identificar de maneira rápida os objetos cujas propriedades ópticas são relevantes através de uma passada de rasterização e em seguida traduzir cada pixel do render obtido em um valor de importância, compatível com a técnica de distribuição não uniforme de raios introduzida no Capítulo 3.

Para constuir a função heurística partindo deste princípio, primeiramente, duas definições adicionais precisam ser feitas:

- B é definido como um *buffer* responsável por armazenar as propriedades ópticas da superfície rasterizada de cada objeto; e
- h' é definida como uma função que mapeia um dado solicitado em B em uma posição (i, j) na grade de pixels do plano de imagem, para $0 \leq i < W$ e $0 \leq j < H$, onde

W e H representam, respectivamente, a largura e a altura da imagem.

Em seguida, para que se possa obter B, executam-se as transformações necessárias para verificar a projeção de cada elemento da cena 3D no plano de imagem, cálculo de *clipping*, e testes de visibilidade em geral, como verificação de oclusão e auto oclusão dos objetos. Isto pode ser feito através de uma passada de rasterização envolvendo um *shader* simples. Esse *shader* deve ser capaz de armazenar para cada fragmento, em vez de sua cor final, as propriedades ópticas desejadas. Nos quatro canais de cor possíveis, armazenam-se as informações para refletividade, índice de refração, opacidade e expoente especular.

O termo de refletividade no primeiro canal de B define a quantidade de luz refletida que é absorvida pelo material antes de chegar ao olho do observador. Indiretamente, este termo também define a necessidade de traçar raios secundários de reflexão a partir de um determinado ponto de uma superfície. Caso isso se faça necessário, sabendo a direção do raio que atinge a superfície de um objeto da cena, o vetor reflexão $\vec{v}_{reflect}$ pode ser calculado por

$$\vec{v}_{reflect} = \vec{v} - 2 \times \vec{n}(\vec{n} \cdot \vec{v}), \tag{4.1}$$

onde $\vec{v}$ é o vetor incidente e $\vec{n}$ é o vetor normal à superfície no ponto em questão, ambos normalizados para a norma 1. O operador $\cdot$ representa o produto interno interno entre $\vec{v}$ e $\vec{n}$.

Já tendo considerado o termo relativo à reflexão para a análise de importância, consideramos a contribuição da refração armazenando no segundo canal de B o termo de refração da superfície, representado pelo índice de refração do material da mesma. Esse índice é responsável por ajustar adequadamente o vetor de direção da luz conforme esta transita entre os diferentes materiais da cena. O vetor $\vec{v}_{refract}$ pode ser calculado através da Equação 4.2, obtida a partir da Lei de Snell [Fassarella 2007]:

$$\vec{v}_{refract} = \frac{\eta_1}{\eta_2}\vec{v} + \left(\frac{\eta_1}{\eta_2}\cos\theta - \sqrt{1 + \left(\frac{\eta_1}{\eta_2}\right)^2 (\cos^2\theta - 1)}\right)\vec{n}, \tag{4.2}$$

onde η_1 e η_2 são os índices de refração dos meios exterior e interior, respectivamente, $\vec{v}$ é o vetor incidente, $\vec{n}$ é o vetor normal à superfície e θ é o ângulo entre $\vec{v}$ e $\vec{n}$.

Os outros canais do *buffer* armazenam a opacidade e o termo de especularidade, que são informações relacionadas à refração e à reflexão respectivamente. Durante a renderização da cena, o termo de opacidade é determinante para verificar a necessidade de

traçar raios secundários para tratar a refração, que só ocorre caso as superfícies sejam translúcidas. Já o expoente especular do material é usado para amostrar o lóbulo de reflexão de uma superfície em diversos modelos de iluminação, como por exemplo o modelo de [Phong 1975]. O valor desses dois parâmetros adicionais pode ser utilizado para intensificar ou atenuar a contribuição dos componentes de reflexão e refração no fator de importância da região.

Para manter a consistência com as definições feitas na Seção 3.1 do Capítulo 3, a função heurística h (Equação 3.1) adequada para a execução do *pipeline* proposto precisa retornar um valor de importância $h(i, j) \geq 0$ para toda coordenada (i, j) do plano de imagem compreendida entre a origem e as dimensões W e H da imagem final. Como h' já faz o mapeamento dos dados no plano de imagem, basta cumprir esta condição para ser uma candidata válida a função h. Se isso não acontecer, é necessária uma função auxiliar h'' que se componha a h' e leve cada elemento de seu contradomínio a um elemento do domínio de h.

Uma discussão interessante sobre o conceito de heurística apresentado nesta seção é que ele pode ser aplicado de duas maneiras diferentes, modificando completamente o sistema. Primeiramente, pode-se entender esta etapa de rasterização e aplicação de *shader* apenas como um pré processamento do *pipeline* proposto, já que o objetivo principal é a criação do Mapa de Importâncias. Sendo assim, sobrepondo o mapa contendo as propriedades ópticas da cena com um que garanta um valor diferente de zero para todo pixel do plano de imagem, temos, como esperado, uma imagem final completamente gerada por traçado de raios distribuídos conforme os atributos da cena. Um alternativa a esse método é transformar o sistema em um renderizador híbrido ao aproveitar a rasterização da primeira etapa. Isto pode ser feito adaptando o mapa de propriedades ópticas do G-Buffer para ser utilizado como Mapa de Importâncias (a partir da média dos canais, por exemplo), e fazendo uso do relaxamento do conceito de Mapa de Importâncias que permite que existam pixels que não sejam atravessados por nenhum raio na distribuição. Caso essa abordagem seja a escolhida, a imagem gerada pelo traçador de raios deve ser composta à imagem rasterizada, atenuada sob a forma de transparência. Vale ressaltar que essa composição também pode ser feita de maneira não homogênea, pesando a participação de cada pixel da imagem raio traçada em relação à rasterizada, baseado tanto na importância quanto nas propriedades ópticas de cada um.

4.2 Computação de Mapa de Saliências

A ideia de computar mapas de saliências em cenas tridimensionais não é nova. Baseando-se no fato de que o sistema visual humano não é capaz de processar toda a informação de uma cena [Tsotsos 1990], muitos trabalhos têm usado a distinção de saliências nas superfícies de um determinado ambiente como métrica para modelos de atenção visual. Alguns exemplos são os trabalhos de [Itti et al. 1998] sobre análise rápida de cenários e [Olshausen et al. 1993] sobre foco de atenção. Outros estudos utilizam o conceito de análise de saliências através de outros temos: [Tsotsos et al. 1995, Ma and Zhang 2003], usam o termo atenção visual, enquanto [Kadir et al. 2004, Itti and et al. 2009] chamam de imprevisibilidade, fator de raridade ou de surpresa. Inicialmente, baseados em modelos biológicos [Koch and Ullman 1985], os modelos de mapa de saliência ganharam popularidade nas aplicações visuais, e hoje, encontram-se na literatura modelos puramente computacionais que usam diversas estratégias de processamento de imagem para estimar o valor de saliência de cada pixel. Caso o leitor se interesse, é recomendada a leitura de [Filipe and Alexandre 2013], um *survey* que contém uma série de diferentes estratégias e modelos de detecção de saliências computacionais e baseados no sistema visual humano.

Naturalmente, a proposta de identificar saliências perceptíveis em uma cena qualquer se aproxima bastante do conceito de estabelecer diferentes regiões de importância em uma cena. Sendo assim, estratégias para calcular esses mapas de saliências, se forem computacionalmente viáveis, podem ser convertidas em heurísticas promissoras para a criação de Mapas de Importâncias.

Em [Su and Takahashi 2010], é proposto um modelo de computação de mapa de saliências que pode ser ajustado para o escopo deste trabalho. Nele, um mapa de importâncias é calculado pela combinação de características de baixo nível da cena como intensidade e cor, que são extraídas utilizando filtros de imagem em diferentes frequências, junto com um conjunto de características temporais para cenas dinâmicas. Em seguida, é feita uma passada para aplicação de uma técnica de *Semantic Depth of Field*, que consiste em uma análise semântica de profundidade de imagem, aplicando o efeito de borramento (do inglês, *blur*) de maneira seletiva e assim desviar a atenção do observador para outros elementos da cena. Esta segunda passada tem como finalidade gerar uma imagem final, e não se encaixa na proposta de definir um Mapa de Importâncias para o *pipeline* proposto neste trabalho. Podemos, entretanto, nos inspirar nas primeiras considerações feitas sobre as características da cena e formular uma heurística que seja computacionalmente viável para aplicações em tempo real. As Equações 4.3, 4.4 e 4.5 modelam as características

chamadas de baixo-nível, intrínsecas à cena.

$$C = \frac{1}{3}(r + g + b) \tag{4.3}$$

$$RG = r - g \tag{4.4}$$

$$BY = b - \frac{1}{2}((r + g) - |r - g|) \tag{4.5}$$

onde r, g e b são as componentes de cor dos canais RGB de cada pixel da imagem; C é o valor que codifica a intensidade de contraste; RG (*red-green*) é um canal duplo de cores opostas, que considera o verde e vermelho; e BY (*blue-yellow*) é um canal análogo que considera a componente azul se opondo com as regiões em amarelo.

Para extrair a informação relativa ao posicionamento das componentes, as chamadas informações de centro e adjacências (*center-surround features*), aplica-se um filtro de diferença de Gaussianas (*DoG filter*), capaz de modelar a resposta dos neurônios do primeiro estágio do sistema visual humano [Young and Lesperance 2001]. A aplicação deste filtro pode ser feita através do *kernel*:

$$DoG(x, y; \sigma) = \frac{\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)}{\sigma^2} - \frac{\exp\left(-\frac{x^2+y^2}{2(2\sigma)^2}\right)}{(2\sigma)^2}, \tag{4.6}$$

onde x e y são as coordenadas do plano de imagem e σ é um parâmetro ajustável que controla a intensidade do filtro. Para extrair características em diferentes frequências, basta ajustar o tamanho da janela σ do filtro DoG, onde $\sigma \in \{2^1, 2^2, 2^3 ... 2^l ... 2^k\}$ (pixels). Em geral, este valor é decidido empiricamente, baseado na resposta visual obtida na aplicação da técnica.

Seja $Input(c)$ um dos canais definidos nas Equações 4.3, 4.4 e 4.5 tal que $c \in \{C, RG, BY\}$ e k seja um valor de escala do filtro DoG definido na Equação 4.6. Definimos um mapa de características espaciais da cena (*spatial feature maps*) F_s como a aplicação do filtro (com uma escala escolhida) à uma das entradas possíveis. Isto é feito através do operador de convolução $*$. Tendo escolhido o valor de k, precisa-se considerar cada um dos três canais definidos anteriormente em todas as suas variações. Para isso, é preciso calcular $3k$ mapas de características espaciais $F_s(c, l)$, iterando o valor de σ:

$$F_s(c, l) = Input(c) * DoG(\sigma_l), \tag{4.7}$$

onde l é um valor inteiro escolhido entre 1 e k.

Finalmente, para consolidar toda esta informação na forma de um Mapa de Im-

portâncias I_{map}, cada um dos mapas de características F_s é normalizado através da sua média e combinado em um único mapa:

$$I_{map} = \sum_{\forall c, \forall l} \frac{F_s(c,l)}{avg(F_s(c,l))}, \tag{4.8}$$

onde $avg(F_s(c,l))$ é uma função que retorna a média de um mapa de características especiais de um determinado canal para todos os possíveis valores de l, responsável por iterar o tamanho da janela do filtro DoG de 2 até 2^k. A notação escolhida é consistente com a definição da Equação 3.2, sendo tanto o formato quanto o conceito do mapa gerado adequados. Sem perder generalidade, podemos assumir a Equação 4.8 como uma heurística h admissível ao escopo deste trabalho.

De aplicação para aplicação, pode-se haver a necessidade de priorizar algum dos mapas de características de baixo nível computados. Nesse caso, o Mapa de Importâncias construído a partir dos mapas de características pode ser definido como a combinação linear de cada um deles, associando diferentes pesos à cada uma das componentes a fim de determinar a sua proporção de importância dentro do cálculo final.

Neste estágio, temos um Mapa de Importâncias calculado compatível com o *pipeline* proposto neste trabalho. Entretanto, [Su and Takahashi 2010] propõem um ajuste adicional, capaz de aumentar a amplitude dos valores de I_{map}, possibilitando-o transmitir uma maior quantidade de informação de importância. Esse ajuste é feito sobre o histograma de I_{map}, através de uma interpolação polinomial de uma sigmoide de quarta ordem para remapear todo pixel p em I_{map} para um novo valor $S(p)$ tal que:

$$S(p) = \frac{1}{d}(ap^2 + bp^3 + cp^4), \tag{4.9}$$

onde:

$$\begin{aligned} a &= -1 + 4m^2 - 3m^3 \\ b &= 2 - 4m^2 + 2m^3 \\ c &= -1 + 3m - 2m^2 \\ d &= -1 + 2m^2 - m^3 \end{aligned}$$

e m é um limiar que satisfaz $S(m) = m$. Essa transformação adicional permite que os valores do Mapa de Importância acima de m sejam reforçados, enquanto os abaixo de m sejam reprimidos. A Figura 4.1 ilustra um mapa de saliências gerado a partir de uma imagem base, e em seguida ajustado conforme indicado.

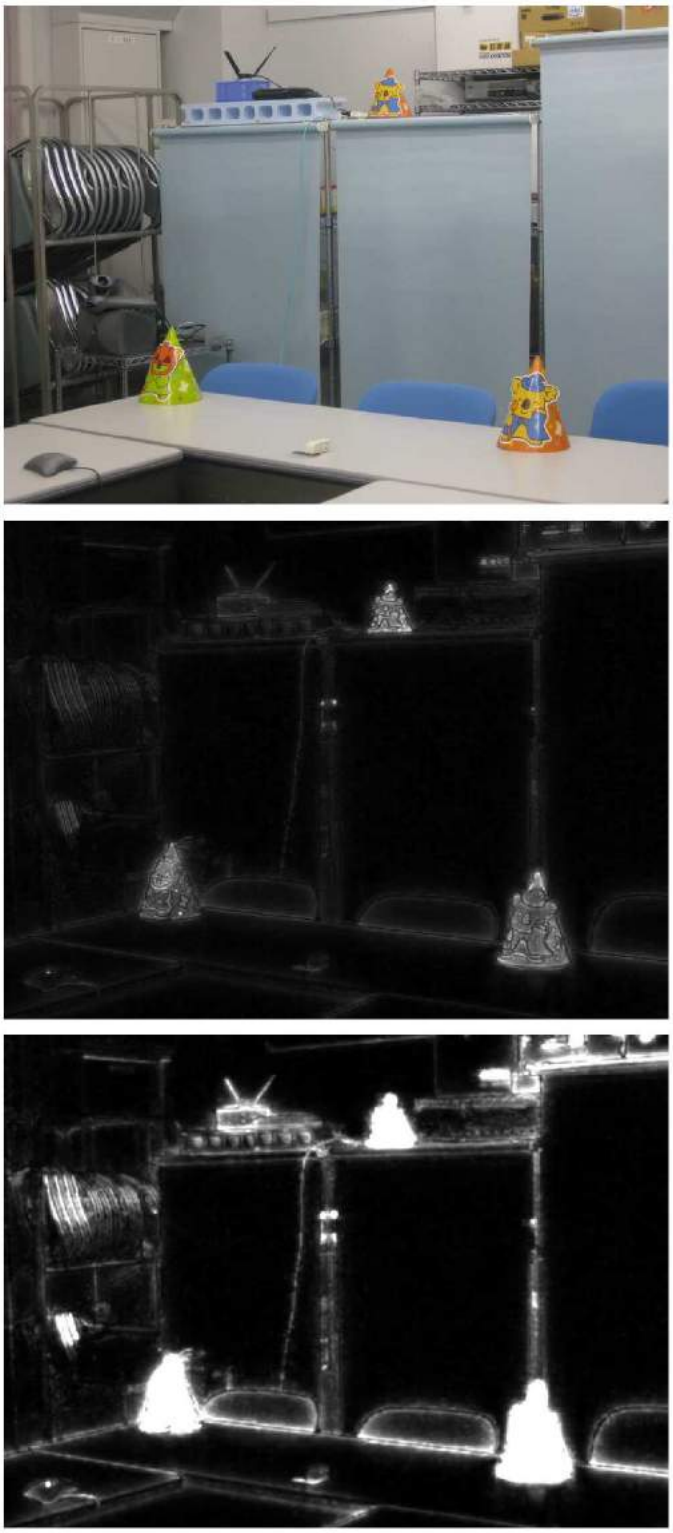

Figura 4.1: Uma cena antes de aplicação da técnica de análise de saliências (cima), seu mapa de saliências correspondente não ajustado (meio) e, seu mapa de saliências ajustado empiricamente com m na metade da média de I_{map} (baixo). Cena extraída de [Su and Takahashi 2010].

Novamente, o mapa gerado é consistente com a definição de Mapa de Importâncias da Seção 3.1 e pode ser usada no estágio inicial do *pipeline* proposto para a geração

de uma distribuição não uniforme de raios sobre a cena. Mesmo assim, propomos um outro ajuste: como nos casos estudados é comum a geração de zonas em preto sólido, que são traduzidas como zonas de importância zero no Mapa de Importâncias, pode-se somar à cada um dos pixels em questão um valor ligeiramente maior que zero. Fazendo isso, garantimos o traçado de pelo menos um raio para cada um dos pixels da imagem final. Caso o sistema não possua uma forma auxiliar de renderizar estes pixels, como em sistemas híbridos, se este ajuste não for feito, a imagem final apresentará zonas não sintetizadas por traçado de raios.

Capítulo 5

Experimentos e Análises

Neste trabalho, foi traçada uma estratégia para, dentro de um ambiente controlado onde existam restrições claras de tempo e processamento, obter um novo patamar de qualidade de imagens renderizadas. Durante todo o *pipeline* proposto, os recursos disponíveis são utilizados a partir de uma heurística bem definida para que, dentro dos parâmetros dela, possa-se maximizar o impacto da imagem final sobre o observador, o que no escopo deste trabalho pode ser entendido como atingir a melhor qualidade de imagem e reduzir o *ailiasing* contando com um orçamento fixo de recursos. Entretanto, medir a qualidade de uma imagem é um processo complicado, já que a opinião humana é subjetiva, dependente de fatores tanto físicos quanto psicológicos.

No campo de processamento de imagem, onde a análise qualitativa das imagens sintetizadas é um problema recorrente, podemos medir a qualidade de uma imagem através de diferentes abordagens: *i*) levando em consideração a diferença pixel a pixel; *ii*) utilizando a correlação estatística entre duas imagens; *iii*) aplicando algoritmos de detecção de arestas; *iv*) usando redes neurais; e finalmente, *v*) recorrendo a modelos que tentam representar o sistema visual humano (HVS, do inglês *Human Visual System*) [Winkler 2001].

Uma análise subjetiva pode ser feita através de um teste de qualidade chamado de MOS (*Mean Opinion Score*, ou Pontuação Média por Opinião, em tradução livre), onde uma imagem é avaliada por uma média de notas compreendidas entre 1 a 5, com a primeira nota representando uma qualidade ruim e a ultima, excelente. Este teste é imprático e demanda tempo excessivo em sua administração, sendo assim, não será abordado. Os experimentos e análises descritos nesta seção vão se concentrar apenas em testes objetivos.

Os experimentos são feitos comparando o *pipeline* tradicional do algoritmo de traçado de raios com o *pipeline* proposto neste trabalho, associado a um Mapa de Importâncias

referente a cada uma das cenas estudadas.

A partir do resultado de cada execução, é preciso estabelecer uma métrica que permita a comparação quantitativa dos resultados, para que uma análise matemática possa ser feita de maneira não tendenciosa. Para estabelecer um parâmetro de comparação, uma imagem raio traçada é renderizada da melhor forma possível, ignorando quaisquer restrições de tempo inerentes ao problema. Esta imagem é utilizada como base de avaliação para todas as outras, tanto as geradas pela técnica proposta quanto as geradas pelo método tradicional, essas sim, sintetizadas sob condições normais da aplicação.

5.1 Definição das Métricas Utilizadas

Neste trabalho foram escolhidas duas métricas para fazer essa avaliação:

- A Relação Sinal-Ruído de Pico, também conhecida por *PSNR* (do inglês *Peak Signal-to-Noise Ratio*), que quantifica a diferença pixel a pixel das imagens comparadas: o sinal de cada pixel gerado a partir da imagem base, e o ruido calculado a partir das renderizadas pelos traçadores em tempo real (Seção 5.1.1).
- O Índice de Similaridade Estrutural, ou *SSIM* (do inglês, *Structural SIMilarity Index*) que abrange a comparação a conceitos como luminância, contraste e estrutura das imagens de entrada (Seção 5.1.2).

As tebelas com os valores para as métricas utilizadas relativos à cada execução serão discriminadas no decorrer deste capítulo, e poderemos analisar a performance da abordagem proposta, nos diferentes cenários para teste.

5.1.1 Relação Sinal-Ruído de Pico

O erro quadrático médio, também chamado de MSE (do inglês, *Mean Square Error*) é umas das métricas mais populares para calcular diferença em processamento de imagens e de videos [Salomon 2007]. Como o nome sugere, ele é definido como a média do quadrado da diferença entre dois valores de um domínio qualquer, no caso, a diferença entre os valores de tons de cinza pixel a pixel de duas imagens A e B:

$$MSE = \frac{1}{W \times H} \sum_{i=0}^{H-1} \sum_{j=0}^{W-1} (A(i,j) - B(i,j))^2 \tag{5.1}$$

A Equação 5.1 representa a definição clássica do erro quadrático médio dentro da área de processamento de imagens, e pode ser utilizada dentro do escopo deste trabalho, que compreende a análise de imagens coloridas, sobre cada canal de cor separadamente. De forma alternativa, pode-se estender a definição para que compreenda as diferenças canal a canal em uma só equação, resultando em:

$$MSE = \frac{1}{3 \times W \times H} \sum_{i=0}^{H-1} \sum_{j=0}^{W-1} (A_R(i,j) - B_R(i,j))^2 + (A_G(i,j) - B_G(i,j))^2 + (A_B(i,j) - B_B(i,j))^2 \tag{5.2}$$

onde A_R, A_G e A_B são os canais vermelho, verde e azul da imagem base, respectivamente. O mesmo vale para a imagem de comparação.

Tecnicamente, enquanto o erro quadrático médio mede a diferença entre duas imagens, a relação entre sinal e ruído de pico, ou PSNR, mede a fidelidade de uma imagem a outra, isto é, o quanto uma imagem se assemelha a uma imagem de referência. Por isso, escolhemos a PSNR como métrica para análise dos experimentos. Ela é definida em função do erro quadrático médio, como:

$$\begin{aligned} PSNR &= 10 \log_{10} \left(\frac{L^2}{MSE} \right) \\ &= 20 \log_{10} \left(\frac{L}{\sqrt{MSE}} \right) \\ &= 10 \log_{10}(L) - 10 \log_{10}(MSE) \end{aligned} \tag{5.3}$$

onde L é o valor máximo que um pixel da imagem atinge. Em geral, $L = 2^{num_bits} - 1$. Vale ressaltar, que para o caso de duas imagens idênticas, onde o MSE é 0, ou seja, não existe ruído, a PSNR é indefinida.

Pelo fato de ser baseada numa comparação de imagens pixel a pixel, a PSNR é simples e fácil de se calcular. Entretanto, pelo mesmo motivo, ela se restringe a uma aproximação bem limitada da distinção de qualidade captada pelo sistema visual humano.

5.1.2 Índice de Similaridade Estrutural

Métodos objetivos para avaliar a qualidade perceptível de uma imagem tradicionalmente vêm tentando quantificar a visibilidade dos erros de uma imagem em relação a uma imagem base. Esses métodos, mesmo que puramente matemáticos, vêm sido incrementados com o passar do tempo para que sejam capazes de modelar propriedades conhecidas do sistema visual humano e assim se tornem uma aproximação cada vez melhor da percepção

do observador.

Os métodos estudados na Subseção 5.1.1 são simples e largamente utilizados como métricas para a comparação de duas imagens. Isso acontece pois são fáceis de calcular e otimizar, e têm um significado matemático bastante claro. Esses métodos, entretanto, não são tão adequados para expressar as nuances da percepção visual do observador [Girod 1993, Eskicioglu and Fisher 1995]. Em função disso e a partir do princípio de que a percepção visual humana é altamente adaptada para extrair informação estrutural de uma cena qualquer, [Wang et al. 2002] propôs o Índice de Similaridade Estrutural (SSIM), que por definição, compreende a análise de características como *luminância, contraste e estrutura.* A Figura 5.1 ilustra como a avaliação do SSIM difere da avaliação clássica baseada no MSE. Nela, a imagem do barco é alterada com diferentes distorções, todas ajustadas para manter o mesmo erro quadrático médio relativo à original. Apesar disso, pode-se observar que as diferenças estruturais incluídas afetam consideravelmente a percepção visual da imagem, resultando numa diferença no SSIM, como esperado.

Figura 5.1: Comparação ilustrativa entre uma imagem original (a) e diversos tipos de distorções aplicados a ela, todas com o valor do MSE = 210. O valor do SSIM entretanto, varia conforme cada distorção perde similaridade estrutural com a original, sendo para (b) SSIM = 0,916; (c) SSIM = 0,990; (d) SSIM = 0,695; (e) SSIM = 0,705 e (f) SSIM = 0,775. Extraído de [Wang et al. 2004].

O SSIM uma métrica válida para a comparação de duas imagens pois, considerando que um dos sinais tenha qualidade perfeita, a valor de similaridade entre eles representa uma medida quantitativa de qualidade do segundo. Neste trabalho, ele é utilizado para comparar quantitativamente imagens geradas por duas distribuições de raios primários distintas em relação a uma imagem base, gerada em altíssima qualidade fora das restrições de tempo de ambientes em tempo real.

O índice de similaridade estrutural é computado da seguinte forma: sejam A e B os dois sinais de imagem a serem comparados, e sejam eles não-negativos e alinhados um com o outro, ou seja, pertencentes ao mesmo domínio. Vale ressaltar que para uma imagem qualquer se torne um sinal compatível, basta que se alinhe seus pixels, transformando o seu domínio originalmente em duas dimensões em um intervalo de 0 a N, onde N é o número total de pixels da imagem.

O primeiro passo é comparar a luminância local dos sinais, que pode ser estimada pelo cálculo da intensidade média:

$$\mu_A = \frac{1}{N} \sum_{i=0}^{N-1} A_i. \tag{5.4}$$

O cálculo da luminância local de B é análogo, obtido pela aplicação da mesma função.

Assim, a comparação entre a luminância de dois sinais é dada por $l(A, B)$ e é uma função de μ_A e μ_B, da forma:

$$l(A, B) = \frac{2\mu_A \mu_B + C_1}{\mu_A^2 + \mu_B^2 + C_1} \tag{5.5}$$

onde C_1 é uma constante incluída para evitar instabilidades, que ocorrem quando $\mu_A^2 + \mu_B^2$ se aproxima de zero. No seu trabalho, Wang et al. aconselha escolher

$$C_1 = (K_1 L)^2,$$

onde L é o valor máximo assumido por um pixel e K_1 uma constante qualquer suficientemente menor que 1.

Na segunda etapa, o método utiliza o desvio padrão σ como estimador para o constrate dos sinais. O desvio padrão relativo ao sinal A é dado por

$$\sigma_A = \left(\frac{1}{N-1} \sum_{i=0}^{N-1} (A_i - \mu_A)^2 \right)^{1/2}, \tag{5.6}$$

e o relativo a B, obtido de forma análoga.

Dessa forma, a função de comparação de contraste se apresenta como uma função $c(A, B)$ do desvio padrão dos sinais, dada pela Equação 5.7:

$$c(A, B) = \frac{2\sigma_A\sigma_B + C_2}{\sigma_A^2 + \sigma_B^2 + C_2}, \tag{5.7}$$

onde $C_2 = (K_2\, L)^2$, definida de maneira análoga a C_1 para evitar instabilidades.

A terceira caraterística levada em consideração por esta métrica é a informação de estrutura de imagem. A comparação de estrutura entre dois sinais é feita através da equação de correlação entre eles, estimada por:

$$\sigma_{AB} = \frac{1}{N-1}\sum_{i=0}^{N-1}(A_i - \mu_A)(B_i - \mu_B). \tag{5.8}$$

E assim, de maneira semelhante às componentes anteriores, a função de comparação de estrutura é dada por $s(A, B)$, definida por:

$$s(A, B) = \frac{\sigma_{AB} + C_3}{\sigma_A\sigma_B + C + 3}. \tag{5.9}$$

Finalmente, combinando as três funções de comparação, o índice de similaridade estrutural entre os sinais A e B é definido como:

$$SSIM(A, B) = l(A, B)^\alpha\, c(A, B)^\beta\, s(A, B)^\gamma, \tag{5.10}$$

onde α, β e γ são três parâmetros positivos utilizados para ajustar a importância relativa de cada um dos componentes. Esses parâmetros podem ser definidos de forma empírica e variam de aplicação para aplicação. Neste trabalho, seguimos a sugestão proposta por Wang et al. para simplificar a expressão final do índice de similaridade, escolhendo $\alpha = \beta = \gamma = 1$ e $C_3 = C_2/2$. Isso reduz a Equação 5.10 a:

$$SSIM(A, B) = \frac{(2\mu_A\mu_B + C_1)(2\sigma_{AB} + C_2)}{(\mu_A^2 + \mu_B^2 + C_1)(\sigma_A^2 + \sigma_B^2 + C_2)}. \tag{5.11}$$

Utilizando a $SSIM$ para medir a performance dos traçadores de raios propostos, espera-se obter uma nova gama de interpretações possíveis, além daquelas obtidas comparando a intensidade pixel a pixel das imagens, como acontece com as métricas mais simples.

5.2 Visão Geral e Discussão dos Experimentos

O arcabouço para aplicar a técnica proposta neste trabalho foi implementado utilizando a linguagem C++. O módulo traçador de raios do *pipeline* de programação foi criado utilizando o NIVIDA OptiX 3.0, e as demais etapas, englobando o pré e pós-processamento requeridos por ele, foram criadas utilizando NVIDIA CUDA 5.0 para a sua execução em paralelo na GPU. O uso do OpenGL se restringe à troca de informações entre os núcleos OptiX e CUDA através do uso de texturas e à exibir a imagem final gerada na tela, não participando da síntese de imagem de modo geral.

Os experimentos foram feitos em uma máquina *desktop* equipada com um processador Intel Core 2 Duo E6750 de 2.67GHz com 4GB de memória RAM e uma GPU NVIDIA GeForce GTX 550 Ti contendo 1024MB de memória. Como sistema operacional foi utilizado o Microsoft Windows 7 versão de 64 bits.

O *pipeline* proposto foi projetado para que pudesse ser integrado a qualquer solução de traçado de raios dado que as estruturas necessárias sejam computadas antes do módulo traçador em todo o quadro da aplicação, e logo em seguida, seja executada a passada de construção da imagem final. Nos experimentos realizados, tanto as etapas do pipeline quanto o núcleo OptiX que age como o módulo traçador de raios são totalmente executados na GPU. Diferentes configurações de cena com os seus Mapas de Importância correspondentes foram usados como *datasets*, e para cada um deles, a cena foi renderizada a partir de diferentes orçamentos de raios e taxas máximas de raios por pixel, responsáveis por gerar diferentes distribuições de raios primários sobre o plano de imagem. A heurística utilizada para gerar os Mapas de Importância enfatiza os materiais reflexivos e refrativos do ambiente em questão, levanto em consideração, também, regiões em sombra. Materiais difusos e o plano de fundo do ambiente recebem menos atenção. A Figura 5.2 mostra a cena clássica de Whitted com o Mapa de Importâncias correspondente, utilizada como arcabouço para o teste do *pipeline*.

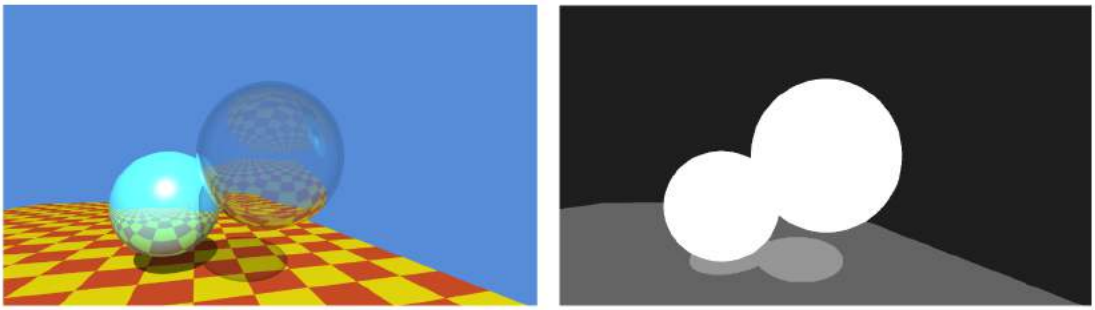

Figura 5.2: Cena clássica de Whitted e seu Mapa de Importâncias correspondente.

Ao decorrer dos testes, foi-se incrementando o grau de complexidade da cena para verificar se o comportamento do sistema se mantinha o mesmo ou se havia alguma modificação significativa quanto à performance. A Figura 5.3 mostra um exemplo de cena onde a complexidade foi elevada através da adição de novos objetos reflexivos e refratáveis, adicionando uma série de novos efeitos de projeção e inter-reflexão na cena anterior.

Figura 5.3: Cena Spheres, gerada aumentando o grau de dificuldade da cena Whitted, e seu Mapa de Importâncias correspondente.

Os experimentos foram realizados da seguinte maneira: para cada cena, foram geradas imagens usando diferentes configurações de distribuições uniformes e não uniformes de raios, os resultados obtidos a partir delas foram comparados com uma imagem base gerada de maneira *offline* a partir de uma distribuição uniforme com uma densidade de raios por pixel muito superior às estudadas.

5.2.1 Análise de Qualidade Visual

Neste trabalho, foram discutidos dois *datasets*, o primeiro é baseado na cena clássica do traçador de raios de Whitted, e o seguinte desenvolvido com o objetivo de aumentar gradativamente a complexidade dessa cena e verificar o comportamento do *pipeline* proposto sob essas condições. Para cada um dos *datasets*, a cena é renderizada utilizando uma variedade de valores para o parâmetro K durante a geração do Mapa de Alocação de Raios (Equação 3.3), e a imagem gerada é comparada com a imagem base a partir de uma das métricas escolhidas. Como paralelo, são geradas distribuições uniformes de raios para a mesmas cenas.

Para os experimentos a seguir, a imagem base de resolução 1280 x 720 pixels foi gerada a partir de uma distribuição uniforme de 25 raios por pixel, por toda a extensão de seu plano de imagem. Os quadros utilizados para comparação foram sintetizados definindo $K \in \{4, 9, 16, 25\}$. Naturalmente, como nenhum valor do Mapa de Importâncias para este *dataset* é zero, no caso onde $K = 1$, as distribuições uniforme e não uniforme de

raios são idênticas, e sendo assim, este caso não foi considerado. O leitor pode observar que K assume valores quadrados perfeitos. Isto acontece pois o tamanho do contexto do núcleo OptiX para traçar os raios é definido em tempo de compilação em função das dimensões da imagem a ser renderizada. Este contexto define o número de raios que vão ser traçados o tamanho do *buffer* de saída do módulo traçador. Nos experimentos efetuados, os diferentes contextos foram criados multiplicando ambas as dimensões da imagem a ser gerada pelos sucessivos números naturais, o que naturalmente geraria uma distribuição uniforme com o quadrado desses números. Como a intenção é não ultrapassar este limite, normalizando o máximo da superfície gerada pelo mapa de importâncias para esses valores (K), garantimos que isso nunca aconteça.

Com a finalidade de verificar se foi alcançado algum salto de qualidade através da técnica proposta, para cada distribuição não uniforme de raios estudada, precisaríamos traçar a mesma quantidade de raios de maneira uniforme, comparar a imagem obtida com a imagem base e verificar os resultados.

Definimos U como o número de raios traçados sobre cada pixel do plano de imagem em distribuições uniformes de raios. Para decidir qual valor de U é o equivalente a cada uma das distribuições escolhidas, utilizamos a seguinte equação:

$$U = \left\lceil \frac{T_K}{1280 \times 720} \right\rceil, \tag{5.12}$$

onde T_K é a quantidade total de raios primários traçados em uma determinada distribuição não uniforme de parâmetro K.

Vale ressaltar que como as quantidades de raios que dispomos em cada configuração de teste nem sempre podem ser distribuídas uniformemente sobre o plano de imagem, e como U é uma grandeza inteira, é preciso arredondar o valor calculado a partir de T_K para cima. Dessa maneira, em determinadas configurações de teste, comparamos os resultados da técnica proposta com distribuições uniformes que utilizam substancialmente mais raios primários. Na Tabela 5.1 estão discriminadas as quantidades de raios disparados tanto nas distribuições de raios geradas pela técnica proposta quanto nas distribuições uniformes equivalentes usadas como paralelo de comparação.

Na tabela, pode-se observar que a diferença de raios entre as distribuições de uma determinada configuração de testes é significativa, variando de por volta de 72.000 a 790.000 raios primários.

As Figuras abaixo apresentam, para cada cena estudada, o resultado da comparação

Tabela 5.1: Configurações de teste para os *datasets* estudados.

	Config.	**Distribuição Não Uniforme**		**Distribuição Uniforme**	
		K	# Raios	U	# Raios
Whitted	1	4	1.611.088	2	1.843.200
	2	9	3.380.414	4	3.686.400
	3	16	5.085.898	6	5.529.600
	4	25	7.702.957	9	8.294.400
Spheres	1	4	1.836.093	2	1.843.200
	2	9	3.909.164	5	4.608.000
	3	16	6.121.724	7	6.451.200
	4	25	9.346.483	11	10.137.600

dos dois tipos de distribuição com a imagem base. Isso é feito para cada uma das quatro possíveis configurações, sob a óptica das duas métricas utilizadas. Para deixar os gráficos mais intuitivos, foi utilizado o conceito de *erro* de uma distribuição em relação à imagem base. No caso da PSNR, definimos o erro E como

$$E = 45 - PSNR, \tag{5.13}$$

já que valores de PSNR acima dos 45dB, em geral, correspondem a compressões que introduzem perdas imperceptíveis ao olho humano.

No caso do SSIM, como o domínio da métrica é o intervalo $[0, 1]$, basta fazer

$$E = 1 - SSIM. \tag{5.14}$$

Sob a óptica do SSIM, para o *dataset* do Whitted, o método proposto conseguiu atingir imagens com 22%, 25%, 24% e 30% a menos de erro para as Configurações de 1 a 4, respectivamente. Para a PSNR a diferença do erro para as diferentes distribuições em cada configuração foi praticamente imperceptível. A Figura 5.4 apresenta um gráfico comparativo para cada um dos casos.

Numa cena como essa, pelo fato de uma grande parte da área do plano de imagem consistir do plano de fundo, é esperado que os valores absolutos de erro obtidos pelas métricas sejam baixos, principalmente em função de boa parte da diferença entre a qualidade das imagens obtidas em cada configuração consistir na redução do *aliasing*, que proporcionalmente ocupa uma porção pequena da imagem final sintetizada. Essa mesma afirmação pode justificar o fato da PSNR ter apresentado uma diferença bem pequena de erro entre as distribuições de cada configuração. Mesmo proporcionalmente baixas,

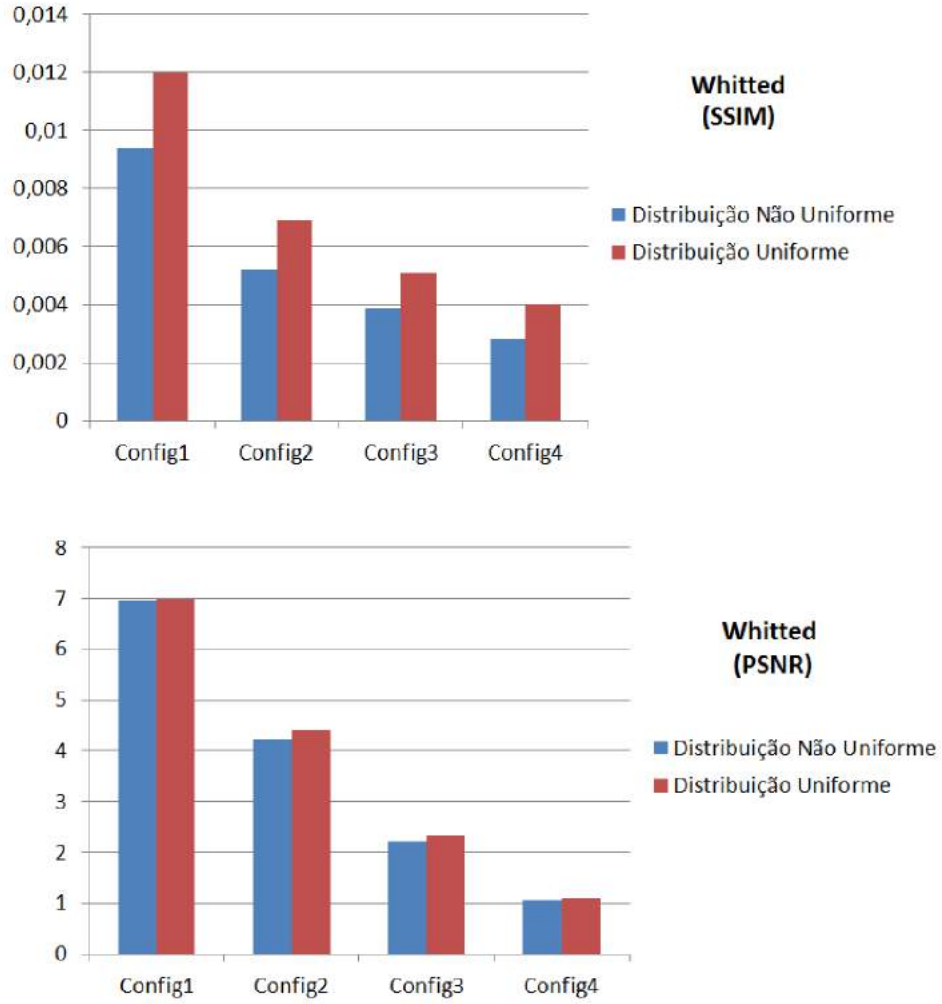

Figura 5.4: Gráficos de erro das diferentes configurações na cena Whitted em relação às metricas estudadas.

as diferenças apontadas entre as distribuições podem ser significantes para o observador, principalmente se concentradas no limite das superfícies, causando o efeito de serrilhado e em zonas de interesse da imagem.

Para a segunda cena, a SSIM apresentou 3%, 12%, 6% e 19% a menos de erro para as Configurações de 1 a 4, respectivamente. Os valores parecem flutuar, aumentando e diminuindo conforme as configurações vão mudando, mas diversos fatores podem ser responsáveis por este comportamento. O pequeno aumento na primeira configuração, por exemplo, pode estar associado ao fato de que para uma cena mais complexa, reordenar os raios primários conforme é feito, tendo um limite pequeno de número máximo de raios por pixel, pode gerar erros retirando recursos de áreas mais simples da cena, e levando esses recursos para areas onde eles ainda não são suficientes para uma síntese adequada do ambiente. Outro fator que não pode ser ignorado, principalmente neste *dataset*, é a

grande variação na quantidade absoluta de raios em determinadas configurações.

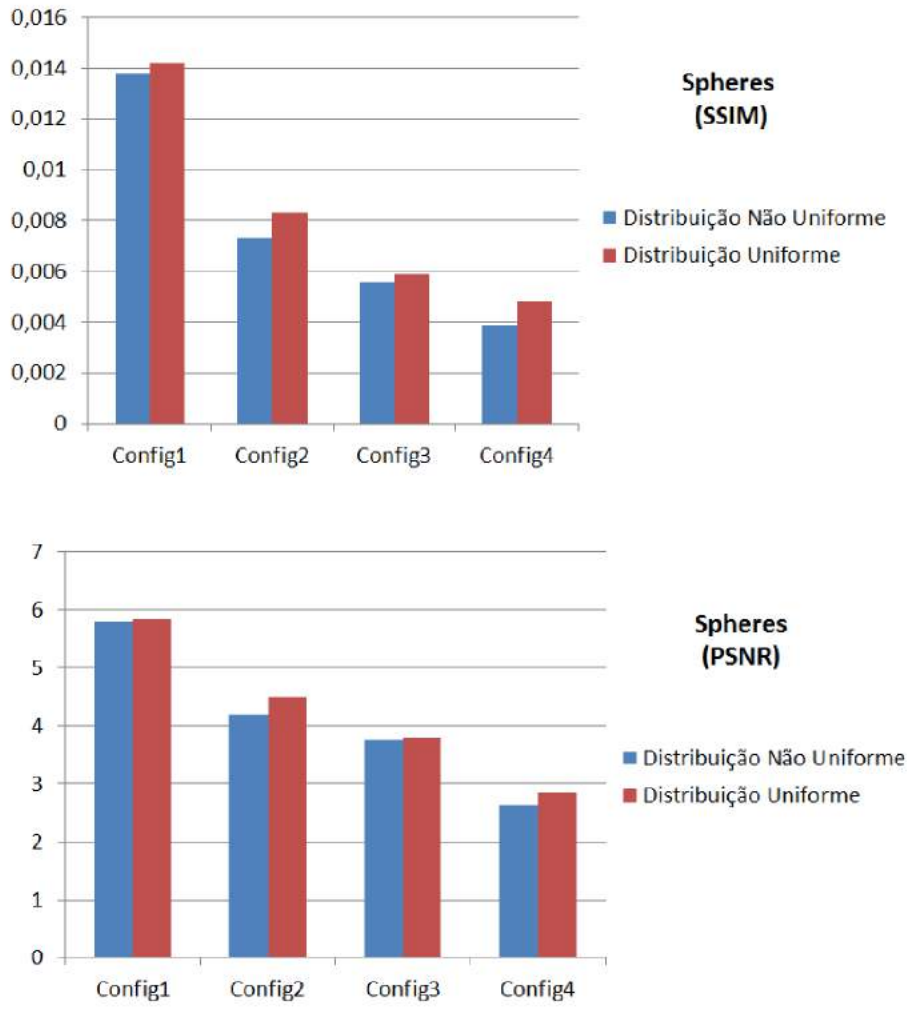

Figura 5.5: Gráficos de erro das diferentes configurações na cena Spheres em relação às metricas estudadas.

Novamente, a PSNR apresenta valores menos expressivos. Porém, ao comparar este *dataset* com o anterior, é possível enxergar um aumento nas variações da métrica, talvez em função do fato de uma cena mais complexa ser capaz de gerar uma quantidade maior de *aliasing*, em termos de proporção da área do plano de imagem que ocupa.

5.2.2 Análise de Desempenho

É possível verificar a aplicabilidade do *pipeline* proposto em sistemas de tempo real, a partir do quanto o processo é escalável, isto é, se o tempo de execução necessário para realizá-lo é suficientemente pequeno e se mantém assim mesmo enquanto o tamanho da entrada cresce. Para que isso possa ser feito, é preciso realizar uma tomada de tempo

nas diferentes etapas da síntese de imagem. Na Figura 5.6 consta a execução dos testes das cenas estudadas, para as distribuições não uniformes de raios discriminadas na seção anterior. Enquanto a linha vermelha marca o tempo de inicio a fim da síntese de imagem, a linha em azul representa o total de tempo de execução gasto tanto para as etapas de pré quanto para de pós-processamento da técnica. A linha tracejada em verde, por sua vez, apresenta um paralelo com o tempo de execução total gasto nas distribuições uniformes equivalentes para cada caso.

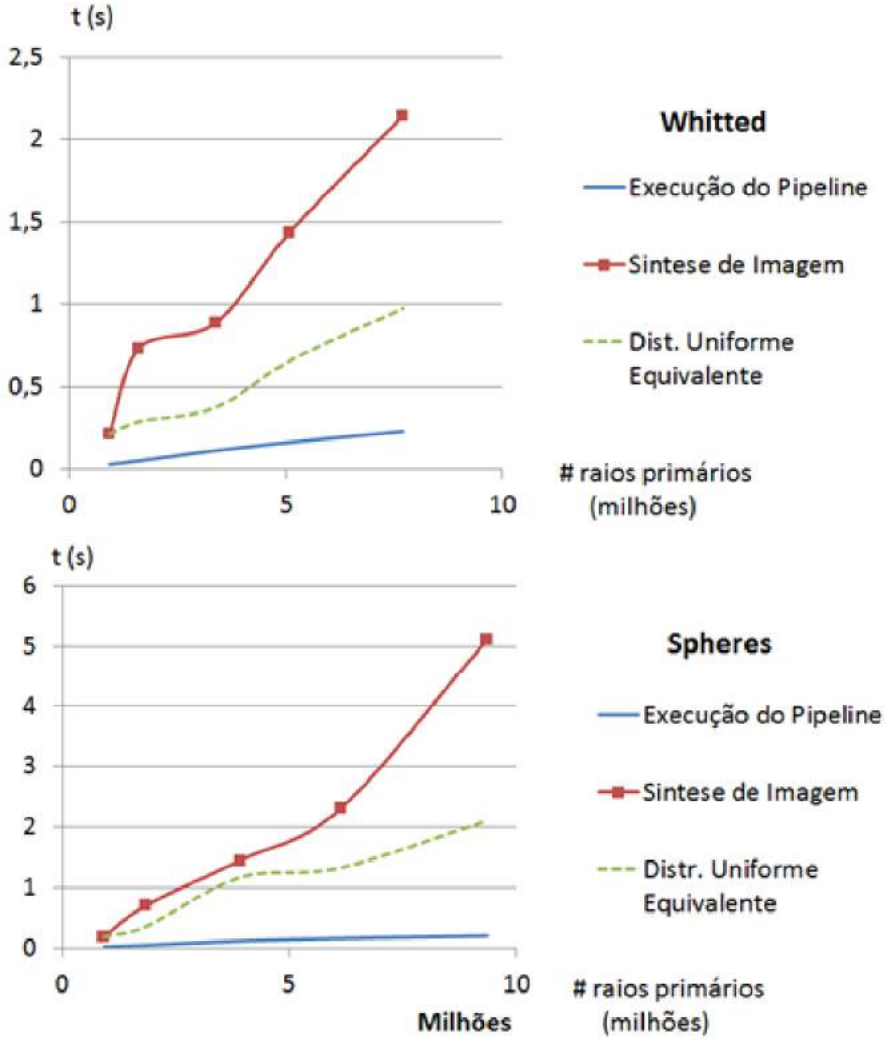

Figura 5.6: Tempo de execução das estruturas do pipeline em relação ao tempo total gasto para síntese de imagem conforme o aumento do número de raios primários, para cada uma das cenas estudadas.

Vale ressaltar que o *pipeline* se comporta como esperado. Nas suas etapas de pré-processamento, executando em paralelo, toda a computação pode ser feita em tempo $O(1)$ para a criação do Mapa de Importâncias a partir da função heurística, $O(\log W\,H)$ para a construção do Mapa de Alocação de Raios a partir de toda a distribuição das

Tabela 5.2: Tempo de execução das etapas dos *datasets* estudados.

	Config.	***Pipeline* (s)**	
		Etapa 1	Etapa 2
Whitted	1	0,014	0,036
	2	0,04	0,072
	3	0,016	0,146
	4	0,014	0,214
Spheres	1	0,014	0,037
	2	0,04	0,120
	3	0,015	0,161
	4	0,014	0,189

Etapa 1: Mapa de Importâncias, Mapa de Alocação de Raios, SAT.
Etapa 2: Mapa de Endereçamento de Raios, Construção da Imagem Final.

importâncias sobre o mapa anterior, $O(\log W\,H)$ para a criação da SAT nas primeiras três etapas. Como W e H são constantes durante toda a execução, todo esse processamento deve também levar tempo constante na execução. As próximas etapas acontecem em espaço de raios, e sendo assim, seu processamento depende dos parâmetros de entrada. Utilizando a SAT, conseguimos computar tanto o Mapa de Endereçamento de Raios como construir a imagem final a partir do *buffer* de saída do traçador conforme esperado. A complexidade de tempo $O(\log K\,W\,H)$ não é possível de ser atingida durante os testes tendo em vista as capacidades da placa gráfica utilizada, sendo assim, o tempo observado pode ser entendido como uma função da complexidade em casos ideais com o adicional do tempo de *warp* que o hardware leva para organizar todas as *threads* instanciadas. É importante observar, entretanto, que essa parcela de tempo também deve variar conforme a entrada. Na Tabela 5.2 podemos observar que de fato o *pipeline* escala como projetado.

Naturalmente, a execução da aplicação depende também da função heurística de entrada do *pipeline*, e dependendo de como ela organizar o Mapa de Importâncias no começo de todo o processamento, o *pipeline* pode ou não se manter adequado às restrições do ambiente. As heurísticas utilizadas na Seção 5.2.1, por exemplo, associam informação de importância aos pixels do plano de imagem baseadas apenas no tipo de material que o compõe. Sendo assim, ao concentrar a maior parte dos raios primários disponíveis para as regiões de alto impacto visual, ela também é responsável pela criação de subsequentes raios secundários parada cada um deles, e dessa forma, não é uma heurística pertinente quando se tem como prioridade o desempenho do sistema como um todo. Nenhuma das heurísticas estudadas leva em consideração os raios secundários e os demais níveis de

recursão do traçador de raios, focando principalmente nas regiões de interesse do observador, o que justifica o resultado obtido. Como paralelo, é importante observar o outro lado do espectro também: nada impede que uma função heurística seja criada apenas levando em consideração o número de raios secundários e/ou sua estimativa de processamento dentro da GPU, deixando de lado as características visuais da cena em questão. Pode-se argumentar com alguma facilidade, que por mais que seja um problema difícil, o ideal é balancear estes dois conceitos em uma única heurística para equilibrar a vantagem das duas abordagens. Tal função heurística pode ser obtida, por exemplo, através de um sistema adaptativo que dada uma heurística inicial, execute o *pipeline* e a corrija conforme for interessante.

Capítulo 6

Conclusões e Trabalhos Futuros

A possibilidade de executar o algoritmo de traçado de raios em aplicações de tempo real tem sido buscada desde o momento em que foi proposto. Tanto a estrutura naturalmente paralela do algoritmo, quanto os resultados que este é capaz de obter são os grandes motivadores de novos trabalhos na área. O traçado de raios é uma técnica clássica, que demanda bastante processamento para sintetizar uma imagem e, sendo assim, por mais que já esteja consolidado em renderizadores *offline*, ainda é preciso que seja otimizado para que se adeque totalmente às restrições de ambientes interativos. Para que esse objetivo seja alcançado, o problema tem sido atacado sob diversas perspectivas: seja através do aprimoramento do hardware corrente; da proposta de novas arquiteturas; da otimização do algoritmo através de estruturas de aceleração; do uso de algoritmos híbridos, combinando vantagens de outras técnicas; da exploração do seu caráter paralelo; ou até através de estratégias auxiliares para um aumento de performance, como o uso de heurísticas.

Esse trabalho se encaixa na vertente que busca explorar o caráter paralelo do algoritmo, e a melhor distribuição de raios primários para aumentar o *speed-up* que pode ser atingido. Para alcançar esse objetivo, foi feito uso de heurísticas auxiliares, capazes de estimar e informar ao algoritmo as regiões de processamento mais críticas, de forma que o processamento paralelo pudesse ser feito com maior eficiência. Para isso, foi criado um *pipeline* de processamento composto de etapas anteriores e posteriores ao traçador convencional, sempre mantendo-as compatíveis com a arquitetura paralela da placa de vídeo e em uma complexidade condizente com o caráter interativo da aplicação.

Dentro do contexto do traçador de raios, a arquitetura da placa de vídeo permite que uma determinada quantidade de raios seja alocada para ser disparada através de cada pixel da cena, permitindo que se aumente o grau de informação obtida e aproximar a cor de cada pixel cada vez mais da realidade. Entretanto, por restrições impostas por

frameworks atuais, essa distribuição de dados deve ser feita de uma forma homogênea, o que se mostra um problema quando temos uma quantidade de recursos limitada. Uma consequência natural desta limitação é que regiões da cena que demandam mais recursos são prejudicadas em função de regiões mais simples, que acabam concentrando uma quantidade maior de raios do que necessitam para chegar a sua cor final. O trabalho proposto nessa dissertação utilizou de métodos heurísticos para estimar a melhor forma de organizar esses recursos e assim aumentar a efetividade do algoritmo.

Foi apresentada uma técnica que consiste em: *i*) a partir de uma heurística bem definida, criar um Mapa de Importâncias para descrever o plano de imagem; *ii*) avaliar a quantidade de recursos disponíveis e gerar um Mapa de Alocação de Raios baseado na importância associada a cada pixel; *iii*) calcular a imagem integral sobre o mapa de alocação de raios para que cada *thread* da GPU associada a um índice de raio consiga fazer acesso direto à sua área de interesse no plano de imagem; *iv*) construir um Mapa de Endereçamento de Raios baseado nas estruturas anteriores, com a posição do plano de imagem para onde cada raio será alocado; *v*) executar o algoritmo de traçado de raios em paralelo, de acordo com o Mapa de Endereçamento gerado; e, finalmente, *vi*) reconstruir a imagem final, combinando as diferentes quantidades de raio pertinentes a cada posição do mapa em um pixel do plano de imagem.

6.1 Trabalhos Futuros

O Capítulo 3 detalha como a realocação de raios é executada dentro do *pipeline* proposto nesta dissertação. Nele, os raios a serem traçados pelo algoritmo de traçado de raios são organizados pelo plano de imagem a partir da informação de importância associada a cada uma das regiões desse plano. O disparo desses raios, tal como seus testes de colisão e computações de cor, são executados de forma paralela em GPU, onde no momento do traçado de raios, cada *thread* fica responsável pela computação de um deles. Por mais que o *pipeline* proposto realize bem a distribuição dos raios primários fora do traçador baseado nas particularidades da cena, ele não leva em consideração nenhuma informação relativa ao *hardware* gráfico para associar as suas *threads* a cada raio especificamente, sendo assim, dentro da arquitetura da GPU ainda há possibilidades para aperfeiçoamento. Como o comportamento de cada raio é independente do outro na GPU, e a memória dela é organizada na forma de blocos, não há garantias que não sejam criadas bolhas durante o processamento concorrente. Se um raio demanda um processamento muito mais exaustivo do que o seu adjacente em memória, casos as suas *threads* associadas estejam organizadas

no mesmo bloco, haverá naturalmente uma espera inerente ao desbalanceamento de carga de processamento entre elas. O balanceamento de carga no processamento concorrente do algoritmo certamente é um objeto promissor de estudos futuros. Uma possível reassociação de raios a *threads* baseada na estimativa de traçado e carga de processamento de cada um pode gerar um *speed-up* ainda maior no algoritmo, e quem sabe, atingir resultados ainda mais promissores. Além disso, caso seja possível estipular uma forma de predizer a criação dessas bolhas de processamento na GPU, pode-se incorporar essa informação na heurística utilizada, por exemplo, remodelando a função geradora do Mapa de Importâncias para que esse cenário fosse minimizado.

Uma limitação do *pipeline* proposto neste trabalho é o fato da distribuição de recursos que ele proporciona ser baseada no número máximo de raios que atravessam cada pixel. O *pipeline* foi projetado desta maneira visando executar suas etapas em baixa complexidade. Utilizando este número como parâmetro, basta uma operação de normalização do Mapa de Importâncias e eventuais ajustes utilizando a função teto nas importâncias pontuais para alcançar uma distribuição global de raios primários condizente com a proposta da técnica. Por mais que esse conceito se mostre suficiente, conseguindo saltos na qualidade de imagem com ainda menos raios que a distribuição uniforme convencional, é difícil fazer um paralelo entre um orçamento fixo de raios e o número máximo de raios a atravessar o plano de imagem nos seus valores de maior importância, afinal, não há garantias de que o valor exato do orçamento de raios que se dispõe possa ser distribuído perfeitamente pela superfície do Mapa de Importâncias gerado pela função heurística. Atualmente, associando um valor definido ao ponto máximo da superfície do Mapa de Importâncias, seja empiricamente ou de forma adaptativa, solucionamos o problema da distribuição tomando esse orçamento fixo como um limite superior de instâncias de raios, sem modificar o mapa. Porém, um estudo adicional pode ser realizado a fim de obter diretamente o parâmetro ideal para a construção de uma distribuição suficientemente boa para a execução da técnica proposta, alterando ligeiramente o mapa de importâncias inicial para que se trabalhe com a estimativa exata de raios disponíveis. Dessa forma, os conceitos de orçamento de tempo para a execução e de regiões de importância de uma cena podem ser relacionados de maneira mais intuitiva com a distribuição de raios gerada.

Outro assunto que pode ser abordado em possíveis trabalhos futuros é o estudo e desenvolvimento de outras funções heurísticas. Cada função heurística deve prover uma estimativa admissível ao problema, sendo consistente com a imagem final. Basicamente, uma boa heurística deve associar regiões de alto impacto visual e que demandam mais processamento a valores de alta importância e regiões mais simples, a valores de importância

proporcionalmente menores. Obviamente, existe uma maneira ótima de organizar os recursos, encontrando o balanceamento perfeito entre a predileção das regiões mais complexas e o detrimento das regiões mais simples de uma forma suave. Tanto encontrar o arranjo ótimo, como provar que este é ótimo, são problemas interessantes de serem abordados, e podem ser atacados de diversas formas. Algumas estratégias para fazer isso são: modelar o problema como um problema de programação inteira; aperfeiçoar a heurística através de sistemas adaptativos; buscar heurísticas ótimas através de sistemas nebulosos ou redes neurais, etc.

Particularmente, acredito que este trabalho foi um passo importante na área em questão, e espero que ele motive o desenvolvimento de novas frentes interessantes de pesquisa. Certamente, este trabalho foi inspirado por muitos antes dele, e me sentirei honrado caso ele seja fonte de inspiração para avanços futuros da técnica de traçado de raios.

Referências

[Arvo and Kirk 1989] Arvo, J. and Kirk, D. (1989). A survey of ray tracing acceleration techniques. *An introduction to ray tracing*, pages 201–262.

[Beck et al. 2005] Beck, S., c. Bernstein, A., Danch, D., and Frohlich, B. (2005). CPU-GPU hybrid real time ray tracing framework.

[Bolin and Meyer 1998] Bolin, M. R. and Meyer, G. W. (1998). A perceptually based adaptive sampling algorithm. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '98, pages 299–309, New York, NY, USA. ACM.

[Cook 1986] Cook, R. L. (1986). Stochastic sampling in computer graphics. *ACM Transactions on Graphics*, 5(1):51–72.

[Cook et al. 1984] Cook, R. L., Porter, T., and Carpenter, L. (1984). Distributed ray tracing. In *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '84, pages 137–145, New York, NY, USA. ACM.

[Crow 1984] Crow, F. C. (1984). Summed-area tables for texture mapping. *ACM SIGGRAPH Computer Graphics*, 18(3):207–212.

[Deering et al. 1988] Deering, M., Winner, S., Schediwy, B., Duffy, C., and Hunt, N. (1988). The triangle processor and normal vector shader: A vlsi system for high performance graphics. *ACM SIGGRAPH Computer Graphics*, 22(4):21–30.

[Dubois and Rodrigue 1977] Dubois, P. and Rodrigue, G. (1977). An analysis of the recursive doubling algorithm. *High Speed Computer and Algorithm Organization*, pages 299–305.

[Eskicioglu and Fisher 1995] Eskicioglu, A. M. and Fisher, P. S. (1995). Image quality measures and their performance. *IEEE Transactions on Communications*, 43(12):2959–2965.

[Fassarella 2007] Fassarella, L. (2007). Lei de snell generalizada. *Revista Brasileira de Ensino de Fisica*, 29:215 – 224.

[Filipe and Alexandre 2013] Filipe, S. and Alexandre, L. (2013). From the human visual system to the computational models of visual attention: A survey. *Artificial Intelligence Review*, pages 1–47.

[Foley and Sugerman 2005] Foley, T. and Sugerman, J. (2005). KD-tree acceleration structures for a GPU raytracer. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*, pages 15–22.

[Girod 1993] Girod, B. (1993). Digital images and human vision. chapter What's Wrong with Mean-squared Error?, pages 207–220. MIT Press, Cambridge, MA, USA.

[Glassner 1989] Glassner, A. (1989). *An introduction to ray tracing.* Morgan Kaufmann.

[Glassner 1984] Glassner, A. S. (1984). Space subdivision for fast ray tracing. *Computer Graphics and Applications, IEEE*, 4(10):15–24.

[Guenter et al. 2002] Guenter, B., Finch, M., Drucker, S., Tan, D., and Snyder, J. (2002). Foveated 3d graphics. In *ACM SIGGRAPH Asia.*

[Hensley et al. 2005] Hensley, J., Scheuermann, T., Coombe, G., Singh, M., and Lastra, A. (2005). Fast summed-area table generation and its applications. *Computer Graphics Forum*, 24:547–555.

[Itti and et al. 2009] Itti, L. and et al. (2009). Bayesian surprise attracts human attention.

[Itti et al. 1998] Itti, L., Koch, C., and Niebur, E. (1998). A model of saliency-based visual attention for rapid scene analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(11):1254–1259.

[Kadir et al. 2004] Kadir, T., Zisserman, A., and Brady, M. (2004). An affine invariant salient region detector. In Pajdla, T. and Matas, J., editors, *ECCV (1)*, volume 3021 of *Lecture Notes in Computer Science*, pages 228–241. Springer.

[Kainz et al. 2011] Kainz, B., Steinberger, M., Hauswiesner, S., Khlebnikov, R., and Schmalstieg, D. (2011). Stylization-based ray prioritization for guaranteed frame rates. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Non-Photorealistic Animation and Rendering*, NPAR '11, pages 43–54, New York, NY, USA. ACM.

[Keller et al. 2013] Keller, A., Karras, T., Wald, I., Aila, T., Laine, S., Bikker, J., Gribble, C., Lee, W.-J., and McCombe, J. (2013). Ray tracing is the future and ever will be... In *ACM SIGGRAPH 2013 Courses*, SIGGRAPH '13, pages 9:1–9:7. ACM.

[Koch and Ullman 1985] Koch, C. and Ullman, S. (1985). Shifts in selective visual attention: towards the underlying neural circuitry. *Human Neurobiology*, 4:219–227.

[Lauterbach et al. 2009] Lauterbach, C., Garland, M., Sengupta, S., Luebke, D., and Manocha, D. (2009). Fast BVH construction on GPUs. In *Proceedings of EUROGRAPHICS*, pages 375–384.

[Lee et al. 1985] Lee, M. E., Redner, R. A., and Uselton, S. P. (1985). Statistically optimized sampling for distributed ray tracing. In *ACM SIGGRAPH Computer Graphics*, volume 19, pages 61–68. ACM.

[Ma and Zhang 2003] Ma, Y.-F. and Zhang, H.-J. (2003). Contrast-based image attention analysis by using fuzzy growing. In *Proceedings of the Eleventh ACM International Conference on Multimedia*, MULTIMEDIA '03, pages 374–381, New York, NY, USA. ACM.

[Martinez et al. 2012] Martinez, F., Clua, E., Sabino, T., and Pagliosa, P. (2012). An adaptive hybrid raster ray-tracing renderer to real-time applications.

[Mitchell 1987] Mitchell, D. P. (1987). Generating antialiased images at low sampling densities. *ACM SIGGRAPH Computer Graphics*, 21(4):65–72.

[Mitchell 1991] Mitchell, D. P. (1991). Spectrally optimal sampling for distribution ray tracing. In *Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '91, pages 157–164, New York, NY, USA. ACM.

[Nehab et al. 2011] Nehab, D., Maximo, A., Lima, R., and Hoppe, H. (2011). Gpu-efficient recursive filtering and summed-area tables. In *Proceedings of the ACM SIGGRAPH Asia*.

[Olshausen et al. 1993] Olshausen, B. A., Anderson, C. H., and Essen, D. C. V. (1993). A neurobiological model of visual attention and invariant pattern recognition based on dynamic routing of information. *Journal of Neuroscience*, pages 4700–4719.

[Parker et al. 2010] Parker, S. G., Bigler, J., Dietrich, A., Friedrich, H., Hoberock, J., Luebke, D., McAllister, D., McGuire, M., Morley, K., Robison, A., and Stich, M. (2010). OptiX: A general purpose ray tracing engine. In *Proceedings of ACM SIGGRAPH*, pages 66:1–66:13.

[Pearl 1983] Pearl, J. (1983). *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley.

[Phong 1975] Phong, B. T. (1975). Illumination for computer generated pictures. *Communications of the ACM*, 18(6):311–317.

[Reeth et al. 1996] Reeth, F. V., Monsieurs, P., Bekaert, P., and Flerackers, E. (1996). Ray tracing optimization utilizing projective methods. In *Proceedings of the 1996 Conference on Computer Graphics International*, pages 47–.

[Rigau et al. 2003] Rigau, J., Feixas, M., and Sbert, M. (2003). Refinement criteria based on f-divergences. In *Proceedings of the 14th Eurographics Workshop on Rendering*, EGRW '03, pages 260–269, Aire-la-Ville, Switzerland, Switzerland. Eurographics Association.

[Rousselle et al. 2011] Rousselle, F., Knaus, C., and Zwicker, M. (2011). Adaptive sampling and reconstruction using greedy error minimization. In *Proceedings of SIGGRAPH Asia*, SA '11, pages 159:1–159:12. ACM.

[Rousselle et al. 2012] Rousselle, F., Knaus, C., and Zwicker, M. (2012). Adaptive rendering with non-local means filtering. *ACM Transactions on Graphics*, 31(6):195:1–195:11.

[Ruff et al. 2013] Ruff, C. F., Clua, E. W. G., and Fernandes, L. A. F. (2013). Dynamic per object ray caching textures for real-time ray tracing. In *Proceedings of the XXVI Conference on Graphics, Patterns and Images (SIBGRAPI)*, pages 258–265.

[Sabino 2012] Sabino, T. L. (2012). Uma arquitetura de pipeline híbrido para rasterização e traçado de raios em tempo real. Master's thesis, Universidade Federal Fluminense.

[Salomon 2007] Salomon, D. (2007). *Data Compression: The Complete Reference*. With contributions by Giovanni Motta and David Bryant.

[Shirley 1991] Shirley, P. (1991). Discrepancy as a quality measure for sample distributions. In *In Eurographics '91*, pages 183–194. Elsevier Science Publishers.

[Shirley et al. 1996] Shirley, P., Wang, C., and Zimmerman, K. (1996). Monte carlo techniques for direct lighting calculations. *ACM Transactions on Graphics*, 15(1):1–36.

[Steinberger et al. 2012] Steinberger, M., Kainz, B., Hauswiesner, S., Khlebnikov, R., Kalkofen, D., and Schmalstieg, D. (2012). Ray prioritization using stylization and visual saliency. *Computers & Graphics*, 36:673–684.

[Stich et al. 2009] Stich, M., Friedrich, H., and Dietrich, A. (2009). Spatial splits in bounding volume hierarchies. In *Proceedings of High-Performance Graphics*, pages 7–13.

[Su and Takahashi 2010] Su, Z. and Takahashi, S. (2010). Real-time enhancement of image and video saliency using semantic depth of field. In *VISAPP (2)*, pages 370–375.

[Tsotsos 1990] Tsotsos, J. K. (1990). Analyzing vision at the complexity level. *Behavioral and Brain Sciences*, 13:423–445.

[Tsotsos et al. 1995] Tsotsos, J. K., Culhane, S. M., Wai, W. Y. K., Lai, Y., Davis, N., and Nuflo, F. (1995). Modeling visual attention via selective tuning. *Artificial Intelligence Review*, 78(1-2):507–545.

[Wald 2004] Wald, I. (2004). *Real Time Ray-Tracing and Interactive Global Illumina.* PhD thesis, Saarland University.

[Wang et al. 2002] Wang, Z., , Wang, Z., and Bovik, A. C. (2002). Why is image quality assessment so difficult? In *in Proceedings of IEEE International Conference of Acoustics, Speech, and Signal Processing*, pages 3313–3316.

[Wang et al. 2004] Wang, Z., Bovik, A. C., Sheikh, H. R., and Simoncelli, E. P. (2004). Image quality assessment: From error visibility to structural similarity. In *IEEE Transactions on Image Processing*, volume 13.

[Welstead 1999] Welstead, S. T. (1999). Fractal and wavelet image compression techniques. In *SPIE Publication*, pages 155 – 158.

[Whitted 1980] Whitted, T. (1980). An improved illumination model for shaded display. *Communications of the ACM*, 23(6):343–349.

[Williams 1983] Williams, L. (1983). Pyramidal parametrics. In *Proceedings of the 10th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '83, pages 1–11, New York, NY, USA. ACM.

[Winkler 2001] Winkler, S. (2001). *Vision models and quality metrics for image processing applications.* PhD thesis, STI, Lausanne.

[Young and Lesperance 2001] Young, R. A. and Lesperance, R. M. (2001). The gaussian derivative model for spatial-temporal vision. *I. Cortical Model. Spatial Vision*, 2001:3–4.