INSTITUTO DE MATEMATICA PURA E APLICADA

Tese de doutorado

Panoramas omnidirecionais expandidos

Aldo René Zang

Luiz Carlos Pacheco Rodrigues Velho
ORIENTADOR

Março 2016

Abstract

Every day we face mixed reality environments, where synthetic and real captured scenes are intertangled to our amusement. We see them in the cinema industry, in the television, ads pieces and even scientific simulations. In this thesis we introduce a fresh framework to produce computer generated photo-realistic rendering combined with real sets. The synthetic content is undistinguished from the original elements, while they both share the same in loco captured lighting.

Omnidirectional panoramas are old friends from computer graphics and the entertainment industry. Since the 80ies they are used for reflection mapping, lighting fields and non-traditional medium (old clunky panorama viewers come to mind here). Much has changed since. In the virtual and augmented reality areas, the current course of hardware evolution lead the industry ripe to embrace this new era.

Marginal to the computer graphics advances, there is visible commotion in the photo and video hardware industries. Light-field cameras, panorama and omnidirectional stereo (i.e. spherical stereo) devices, structured-light sensors to name a few. Gadgets coming from the most imaginative piece of sci-fi literature seems to be consumer-ready faster than an optimistic futurist could hope for. And here again, panoramas are everywhere.

It comes then as no surprise that a considerable subset of this work addresses the peculiarities of captured *HDR* omnidirectional panoramas. Thus the resulting pipeline is author-friendly to any panorama-viewer device such as smartphones, fulldomes, caves and head mounted displays (i.e., VR glasses).

There is much that hasn't been accounted for in previous works though. This is partially due to the limitations of a panorama compared to a full representation of a scene. A panorama is a representation format, nonetheless, and as such it can be expanded. Therefore, we introduce in this thesis an *Expanded Panorama* format, where along the lighting information we encode the space geometry as a depth channel in the camera space. With our expanded panoramas we not only can re-lit synthetic objects. The rendering can be done as a single-pass run-through. This excuses the need of multiple-pass calibrations and post-production to fine tune the blending between the original and the new scenarios. The expanded panorama is explored here as a complete solution, including a proper hidden surface determination solution and lighting algorithms.

Resumo

Cenas de realidade mista, onde objetos sintéticos são introduzidos em cenas reais capturadas, são utilizadas diariamente em todo tipo de aplicações que vão desde cinema, televisão, publicidade até simulações científicas. Nesta tese vamos apresentar soluções para o problema de renderização foto-realista de objetos sintéticos em cenários reais com iluminação real capturada. Também propomos uma solução para o problema de renderização foto-realista de objetos sintéticos em um panorama omnidirecional *HDR* capturado. Este pipeline permite desenvolver produções destinadas a serem exibidas em displays que suportam conteúdo panorâmico, como óculos de realidade virtual, smartphones, caves e domos esféricos.

Panoramas omnidirecionais são utilizados em computação gráfica desde inicios dos anos 80 em diversas aplicações, tais como mapeamento de reflexão ou como mapas de iluminação. O curso do desenvolvimento tecnológico e industrial atuais abre espaço a novas aplicações para panoramas, especialmente na área das novas mídias imersivas que vem ganhando popularidade rapidamente. De olho nestas novas tendências da indústria de cinema, apresentamos soluções para resolver problemas relacionados a este tipo de produção de conteúdo panorâmico, propondo uma nova representação do espaço, armazenando conjuntamente com o panorama omnidirecional a informação geométrica codificada como um canal de profundidade. Outra novidade em nossas soluções de renderização para cenas de realidade mista é o fato do pipeline ser resolvido em uma única passada, sem necessidade de múltiplos passos de calibração nem pós-produção para compor os objetos sintéticos com o cenário real.

Recentemente também experimentamos avanços significativos no setor de dispositivos de captura fotográfica, vídeo e estrutura geométrica. Dispositivos como câmeras panorâmicas e de captura de profundidade são cada vez mais acessíveis. Os avanços deste setor abrem novos caminhos para o uso de panoramas. Por outra parte percebemos a necessidade de reformular as representações atuais e introduzir novas representações que permitam explorar melhor os atributos e capacidades das novas tecnologias de captura e mídias de exibição. Para contribuir com o desenvolvimento desta área propomos uma nova formulação geral de panorama expandida, o panorama omnidirecional com múltiplas camadas, que permite explorar novas representações e aplicações e atender melhor às necessidades das novas tecnologias. Entre as aplicações desenvolvidas, estudamos os problemas visualização de superfícies visíveis e visualizações com iluminação *view-dependent* a partir de uma imagem panorâmica expandida.

Agradecimentos

Os agradecimentos serão incluidos na versão final.

Sumário

Capítulo 1

Introdução

Na última década experimentamos grandes avanços no setor de dispositivos móveis. Hoje eles estão à nossa volta no cotidiano. Temos visto também cada vez mais aplicações interativas e de realidade virtual para este tipo de dispositivos.

O constante crescimento no setor de equipamentos de captura e exibição traz consigo uma demanda por aplicações e produção de mídia adequada a estas tecnologias. Um ponto muito explorado nos últimos anos é a captura e exibição de imagens panorâmicas omnidirecionais. A navegação de um panorama omnidirecional num dispositivo móvel é bastante explorada uma vez que este tipo de dispositivo conta com um grande número de sensores de orientação e geolocalização. Por outro lado também apareceram no mercado várias soluções de captura de imagens panorâmicas, entre elas, câmeras de alto desempenho como a *LadyBug*, e outras mais portáveis, como a *Theta 360* da Ricoh.

A demanda por conteúdo panorâmico omnidirecional continua em pleno crescimento devido ao alto número de dispositivos com capacidade de exibição de conteúdo panorâmico, tais como o *Oculus Rift*, *Samsung Gear VR* e até mesmo gadgets para os dispositivos móveis (smartphones e tablets) exibirem conteúdo panorâmico aproveitando seus sensores de localização e giroscópios. Esta onda de produção e consumo de conteúdo multimídia panorâmico vem sendo auxiliada e retroalimentada também pelo surgimento de diversos dispositivos de captura de imagem panorâmica e de profundidade de cena, como o *Structure Sensor*, *Project Tango* do Google e até mesmo o *Google Jump*.

1.1 O problema

Nesta tese procuramos contribuir com o desenvolvimento da computação gráfica aportando soluções e novas ideias para problemas relacionados ao uso de imagens panorâmicas omnidirecionais no contexto de renderização foto-realista. Assim, realizamos diversas pesquisas relacionadas ao uso de imagens panorâmicas omnidirecionais no contexto citado. Nas mesmas são estudados problemas que vão desde amostragem de mapas de iluminação, i.e., panoramas omnidirecionais *HDR*, passando por novos algoritmos de iluminação para renderização de cenas de realidade mista, e culmina com um estudo de novas propostas para representar informações da função plenóptica através de imagens omnidirecionais estendidas.

O **primeiro problema** a ser tratado é o problema de calcular a contribuição de iluminação direta em cenas iluminadas com um mapa de iluminação, i.e., resolver a equação de iluminação para a contribuição direta das fontes de luz. Neste sentido, propomos soluções para o processo de amostragem dos mapas de iluminação.

O **segundo problema** a ser tratado é o de renderizar objetos sintéticos em cenas reais capturadas de maneira foto-realista. usamos como ponto de partida o trabalho de Paul Debevec [17], e desenvolvemos uma solução que resolve o problema sem necessidade de extensos pré-processamentos e pós-processamento de resultados parciais. Esta abordagem mostrou-se flexível especialmente no contexto de renderização de animações.

O **terceiro problema** decorre de situações que não podem ser tratadas adequadamente com o uso de imagens panorâmicas omnidirecionais *HDR* tradicionais. O problema em questão é lograr a inserção de objetos sintéticos em uma imagem panorâmica omnidirecional de maneira foto-realista. A solução deste problema derivou na formulação de um novo tipo de panoramas omnidirecionais, os panoramas omnidirecionais *RGB-D* ou mapas de iluminação *RGB-D*.

Tanto os panoramas omnidirecionais tradicionais quanto os panoramas *RGB-D* têm a limitação de que somente armazenam informação da função plenóptica para um único ponto do espaço. Esta limitação é o motivo de boa parte dos problemas ligados ao uso de panoramas omnidirecionais como mapas de iluminação no contexto de renderização. Esta limitação dos panoramas omnidirecionais atuais motivou o estudo de um **quarto problema**: *"encontrar uma representação adequada para codificar mais informação da função plenóptica, e procurar novos usos para a mesma"*. A nova formulação pode ser usada tanto como mapa de iluminação em rendering, quanto para reconstruir visualizações parciais aproximadas da

função plenóptica para pontos dentro de um domínio local $\Omega \subset \mathbb{R}^3$.

No restante deste trabalho utilizaremos com frequência o termo mapa de iluminação, para nos referir a uma imagem panorâmica omnidirecional *HDR*.

1.2 Contribuições

Entre as contribuições das pesquisas realizadas durante a realização desta tese podemos destacar:

- **Primeiro problema:**
  - apresentação de um método híbrido para classificação e amostragem de mapas de iluminação. eu uso em *IBL*, que impactam nos tempos de renderização;
- **Segundo problema:**
  - Um método de renderização foto-realista, para renderização de cenas de realidade mista, [1, 60, 64];
- **Terceiro problema:**
  - apresentação do conceito de panorama *RGB-D* omnidirecional, e um framework para construí-los a partir de panoramas omnidirecionais tradicionais;
  - Um framework de renderização de cenas de realidade mista, utilizando panoramas *RGB-D* omnidirecionais, [23, 24, 62, 63];
- **Quarto problema:**
  - Uma nova formulação para amostras densas da função plenóptica através de imagens panorâmicas omnidirecionais com múltiplas camadas, *POMC*, [65].
  - Soluções para renderizar vistas não triviais de panoramas estendidas (*POMC*) resolvendo problemas de visibilidade, [65];
  - Soluções para reconstruir aproximações foto-realistas da função plenóptica, i.e., renderizar vistas de panoramas estendidas com posições de câmera fora da origem e produzir efeitos view-dependent foto-realistas.

1.3 Estrutura

A tese consta de 9 capítulos e uma seção de apêndices. Conceitualmente podemos dividir a tese em duas partes divididas da seguinte forma:

Primeira parte: trata dos problemas de renderização foto-realista utilizando iluminação real capturada. Agrupamos aqui os 3 primeiros problemas.

- **Capítulo 3:** tratará do problema do cálculo de iluminação direta em *ray tracing* usando mapas de iluminação. Apresentaremos as técnicas tradicionais e uma técnica híbrida para amostrar mapas de iluminação, [61].
- **Capítulo 4:** será estudado o problema de renderizar objetos virtuais em cenários reais capturados. Apresentaremos um framework de produção de cenas de realidade mista. Como parte central do framework apresentaremos o algoritmo desenvolvido para renderizar de cenas de realidade mista de maneira foto-realista, [1, 60, 64].
- **Capítulo 5:** vamos apresentar um novo tipo de mapa de iluminação, mapas de iluminação *RGB-D*, com propriedades adicionais à dos mapas tradicionais que permitem reconstruir a posição das luzes. A partir desta localização espacial os cálculos de iluminação podem ser efetuados de maneira mais precisa, permitindo efeitos mais realistas para sombras, reflexões e transparência. Apresentaremos um framework para construir mapas de iluminação *RGB-D* a partir de mapas de iluminação tradicionais. Também apresentaremos um algoritmo de renderização que resolve o problema de inserção de objetos sintéticos no cenário real capturado no mapa de iluminação.

Segunda parte: introduz uma nova formulação para imagens panorâmicas omnidirecionais que permite armazenar uma amostragem local da *função plenóptica*. Também são estudados na segunda parte, problemas de visualização e renderização utilizando a nova formulação para imagens omnidirecionais.

- **Capítulo 6:** definiremos o conceito de *panorama omnidirecional com múltiplas camadas* (*POMC*) e seus usos possíveis em Computação Gráfica. Serão discutidas possíveis representações para panoramas *POMC* e estratégias para construí-los, [65].

- **Capítulo 7:** será abordado o problema de determinação de superfícies visíveis no problema de visualizações geradas a partir de um *POMC*. Serão estudadas duas abordagens: renderização baseada em *point splatting* e renderização com malhas, [65].
- **Capítulo 8:** será estudado o problema de renderização de efeitos de iluminação view-dependent em visualizações com *POMC*. O problema será tratado pensando em aplicações de visualização em tempo real. Serão apresentadas duas soluções, utilizando rasterização e um esquema híbrido baseado em *deferred rendering* e *ray tracing*.

Devido ao fato de que a maior parte das pesquisas em computação gráfica é publicada em inglês, e os termos conhecidos também estão nesta língua, optamos por manter alguns termos no idioma original, especialmente aqueles que são de amplo conhecimento na comunidade de computação e computação gráfica. Por exemplo, termos como: *ray tracing, path tracing, rendering, buffer*, entre outros, poderão ser utilizados em inglês segundo o contexto.

Capítulo 2

Contexto histórico

Em 1968, Appel [4] apresentou o algoritmo de *ray tracing*. A ideia era lançar uma grid regular de raios desde a posição da câmera e encontrar o objeto mais próximo que bloqueia esse raio. Assim, esta primeira versão era um algoritmo de determinação de superfícies visíveis. Em 1980, Turner Whitted [57] implementou uma versão mais completa sobre a ideia original de Appel, continuando o traçado do raio após a primeira interseção. Desta maneira, logrou introduzir novos efeitos de iluminação tais como reflexão, refração e sombras. Whitted produziu imagens que impressionaram na época pela qualidade, e desde então a técnica de *ray tracing* desenvolveu-se significativamente.

Um trabalho seminal no campo da simulação fisicamente correta da iluminação, intitulado *"The Rendering Equation"* [34], foi apresentado por Jim Kajiya em 1986, e generalizou as ideias de transporte de luz para qualquer tipo de geometria e qualquer tipo de refletância das superfícies. A equação principal do artigo de Kajiya expressa que a iluminação que emana de uma superfície em cada direção é uma convolução da luz que chega à superfície desde todas as direções com a *BRDF* (Bidirectional Reflectance Distribution Function), função que descreve como a superfície reflete a luz.

Kajiya descreveu um processo para renderizar as imagens de acordo com esta equação utilizando uma técnica numérica estocástica conhecida como *path tracing* (traçado de caminhos). Assim como a técnica de *ray tracing* [57], o *path tracing* gera imagens traçando raios desde uma câmera para as superfícies da cena e, em seguida, traça raios a partir das superfícies para determinar a iluminação incidente sobre as superfícies. No *path tracing*, os raios são traçados não somente na direção das fontes de luz, mas também aleatoriamente em todas as direções a fim de contabilizar a iluminação indireta proveniente do resto da cena.

(a) Mapeamento de Chou. **(b)** Mapeamento de Miller.

Figura 2-1: (a) Robô em CG, com mapeamento de reflexão. (b) Modelo *Dog* em CG, com mapeamento de reflexão obtido a partir da fotografia de um enfeite de natal.

Em 1974, Edwin Catmull [10] desenvolveu o método de *mapeamento de textura*, que permite aplicar uma imagem sobre uma superfície num processo semelhante a um decalque. Após o trabalho de Catmull, diversas aplicações pioneiras de mapeamentos apareceram na literatura e nos efeitos especiais em filmes. Em 1976, Blinn e Newell [6] apresentaram uma extensão dos resultados de Catmull para texturas e reflexão. Blinn introduziu a técnica de mapeamento de reflexão, como uma forma de mapeamento de textura.

O realismo em computação gráfica avançou muito com técnicas que podem capturar iluminação a partir do mundo real e usá-la para criar iluminação em cenas geradas por computador. Se considerarmos um ponto em particular em uma cena, a luz nesse ponto pode ser descrita como o conjunto de todas as cores e intensidades de luz chegando ao ponto desde todas as direções. Uma forma relativamente simples de capturar esta função para um ponto de localização do mundo real é obtendo uma imagem de uma esfera espelhada, chamada de imagem omnidirecional, que reflete luz proveniente de todo o ambiente em direção à câmera.

No inicio da década de 1980, Gene Miller [44] e Mike Chou [58] (figura (2-1)), aplicaram de maneira independente a técnica de Blinn e Newell [6], utilizando fotografias de um cenário real. Em 1983, Lance Williams [58], introduz o *MIP-mapping*, um esquema de pré-filtragem para eliminar aliasing nos algoritmos de mapeamento de textura.

Pouco tempo depois, o mapeamento de reflexão já foi utilizado na indústria do entretenimento para simular as reflexões sobre objetos especulares, como a espaçonave no filme *Flight of the Navigator* (figura (2-2a)), de Randal Kleiser em 1986 e, a mais famosa foi o robô metálico *T1000* do filme *Terminator 2* dirigida por James Cameron em 1991 (figura (2-2b)). Em 1986, Ned Greene [30] publicou um artigo formalizando a técnica de mapeamento de reflexões, mostrando que os mapas de ambientes podem ser pré-filtrados e indexados com tabelas de somas de áreas a fim de obter boas aproximações e antialiasing. Em todos estes

(a) Flight of the Navigator, 1986.

(b) Terminator 2, 1991.

Figura 2-2: (a) Nave espacial com mapeamento de reflexão. (b) Mapeamento de reflexão no robô *T1000* do filme *Terminator 2*.

exemplos a técnica não somente tem produzido reflexões realistas, mas também fez parecer que o objeto realmente forma parte do ambiente onde foi inserido. Este foi um avanço importante para o realismo nos efeitos visuais.

A partir de 1995, com o trabalho de McMillan e Bishop [42] sobre renderização baseada em imagens, aparece uma nova ótica sobre a relação entre iluminação e rendering. McMillan e Bishop fazem uso da função plenóptica, introduzida em 1991 por Adelson e Bergen [2], e mostram como mesmo sem geometria era possível obter novos pontos de vista de uma cena a partir de informações de profundidade recuperadas pela análise de imagens capturadas. O trabalho de McMillan e Bishop impulsiona muitas pesquisas relacionadas ao estudo da função plenóptica nos anos seguintes.

Em 1996, o trabalho de Levoy e Hanrahan [40], intitulado *"Light Field Rendering"* apresenta um método para gerar novos pontos de vista desde posições de câmera arbitrárias, sem necessidade de informação de profundidade ou correspondência de features, basta combinar informação disponível em imagens capturadas. O aspecto chave desta técnica reside em interpretar as imagens de entrada como fatias 2D da função 4D que representa o *Light Field* (campo de luz).

Paralelamente ao trabalho de Levoy e Hanrahan, Gortler et al. [29], apresentaram o *"Lumigraph"*, uma descrição alternativa da função plenóptica. O *Lumigraph* é um subconjunto da função plenóptica completa, que descreve o fluxo de luz em todas as posições e em todas as direções. Com o *Lumigraph*, novas imagens do objeto pode ser geradas rapidamente, independente da complexidade geométrica ou da iluminação da cena ou objeto. O ponto interessante, deste trabalho está na representação explorada para eliminar a alta redundância de informação existente na descrição completa da função plenóptica.

O mapeamento de reflexão tem produzido resultados convincentes para objetos brilhan-

tes, mas foi necessário continuar inovando para conseguir resultados com objetos animados e sem o brilho metálico, tais como criaturas e seres humanos ou objetos diversos. Uma limitação do mapeamento de reflexão é que ele não pode reproduzir efeitos de sombreamento de um objeto nele mesmo ou a inter-reflexão da luz entre superfícies do objeto sintético inserido.

Outra limitação do mapeamento de ambiente tradicional é que uma única fotografia digital de um ambiente dificilmente capta toda a gama de luz visível em uma cena. Assim, normalmente as luzes são "cortadas" pelo ponto de saturação do sensor, impedindo o registro da intensidade real. Isto não é um problema grave para superfícies metálicas brilhantes, pois as reflexões também são cortadas na renderização da imagem final. Entretanto, o problema aparece nas superfícies que espalham a iluminação incidente, refletindo para a câmera somente uma parte da luz incidente, onde a captura incorreta da iluminação do ambiente pode produzir erros graves na visualização final.

Em 1997, Debevec e Malik [19] desenvolveram uma técnica para capturar a totalidade da gama dinâmica da luz em uma cena, conhecida como *HDRI* (High Dynamic Range Imaging). São tomadas fotografias da cena com diferentes configurações de exposição da câmera. Depois, calculando a curva de resposta da câmera, as imagens de gama dinâmica limitada são reunidas numa única imagem de alta gama dinâmica, *HDR* (High Dynamic Range), que representa a radiância real da cena. Em 1998, Debevec apresentou uma abordagem para iluminar objetos sintéticos com iluminação extraída do mundo real, técnica conhecida como *IBL* (Image-Based Lighting) [17]. A ideia básica na técnica de *IBL* é utilizar a iluminação real combinada a um sistema de iluminação global para simular a iluminação incidente sobre os objetos sintéticos. Esta nova abordagem produz todos os efeitos da aparência real do objeto iluminado pela luz do ambiente, incluindo o autossombreamento, e pode ser aplicada a qualquer tipo de material, tal como metal, plástico, vidro, entre outros.

Os métodos de *HDRI* e *IBL*, além de suas técnicas e sistemas derivados, são hoje amplamente utilizados na indústria de efeitos visuais. Estas técnicas forneceram aos artistas de efeitos visuais novas ferramentas de iluminação e composição digital de atores, aviões, carros e criaturas, dando impressão que os mesmos realmente estão presentes durante a filmagem, e não que foram adicionados através de computação gráfica.

As técnicas de *IBL* impulsaram novos trabalhos associados à utilização eficiente dos mapas de iluminação para recriar a iluminação real na cena sintética. Estes trabalhos focalizam

em técnicas de amostragem dos mapas de iluminação para conseguir acelerar o processo de renderização. Entre estes trabalhos está o plugin *LightGen* [12], desenvolvido por Cohen e Debevec em 2001, que converte um mapa de iluminação em luzes direcionais.

Em 2003, Agarwal et al. [3], apresentaram um método de amostragem estruturada por importância de mapas de iluminação. O método otimiza a integração da iluminação distante sobre superfícies lambertianas e semibrilhantes, sendo mais rápido que os métodos baseados em amostragem de Monte Carlo. Também em 2003, Kollig e Keller [37], apresentaram um algoritmo para determinar regras de quadratura para realizar o cálculo da iluminação direta em cenas com predominância de objetos difusos a partir de mapas de iluminação. A técnica produz imagens com sombras suaves, e ganha em velocidade dos métodos de Monte Carlo.

Em 2004, Ostromoukhov et al. [47], introduziram um esquema de amostragem de mapas de iluminação que realiza subdivisões recursivas de maneira hierárquica segundo a função de importância dada pela luminância do mapa. Este esquema depende somente do mapa de iluminação para obter as amostras e é extremamente rápido. Em 2005, Paul Debevec [18], apresentou um esquema de amostragem para mapas de iluminação que aproveita a ideia de corte mediano utilizada no algoritmo de quantização por corte mediano. Trabalhos posteriores tentam realizar a amostragem do produto do mapa de iluminação com a *BRDF* da cena, para minimizar a quantidade final de amostras. Nesta direção, o trabalho de Burke et al. [8], introduziu a noção de amostragem bidirecional por importância, onde as amostras são tomadas da distribuição do produto da iluminação pela refletância da superfície. Outros trabalhos seguindo esta linha são os de Cline et al. [11] e Mei et al. [43], ambos de 2006.

Em 2011, Kevin Karsch et al. [35] apresentaram um método para renderizar objetos sintéticos em fotografias convencionais. A técnica é semiautomática, o usuário introduz alguns dados sobre a estrutura da cena através de uma interface baseada em sketch. O método produz resultados convincentes, mas dada a falta de informação, tem algumas limitações, especialmente para lidar com objetos especulares. Também em 2011, Zang e Velho [64], apresentaram um método para renderizar cenas de realidade mista, utilizando mapas de iluminação capturados juntamente com o fundo da cena, tudo em *HDR*. A técnica consegue resolver a renderização se o uso de composição e lida bem com cenas complexas.

Recentemente, com as novas tecnologias de mídias e dispositivos móveis, o uso de imagens panorâmicas começou a cobrar força. Dispositivos móveis como smartphones e tablets com sensores de posição e orientação espacial, e dispositivos de imersão como o *Google*

CardBoard, *Oculus Rift* e *Gear VR* da *Samsung* estão permitindo aos usuários em todo o mundo interagir com estas novas mídias.

Como parte das pesquisas orientadas a estes novos paradigmas, em 2012, Felinto et al. [23] apresentaram um trabalho que explora o uso de imagens omnidirecionais *HDR* para a produção de conteúdo, visando o interesse crescente em panoramas. Hoje esta demanda continua em pleno crescimento devido ao alto numero de dispositivos com capacidade de exibição de conteúdo panorâmico, tais como o *Oculus Rift*, e até mesmo gadgets para os dispositivos moveis (smartphones e tablets) exibirem conteúdo panorâmico aproveitando seus sensores giroscópios e de localização. Esta onda de produção e consumo de conteúdo multimídia panorâmico vem sendo auxiliada e retroalimentada também pelo surgimento de vários dispositivos de captura de imagem panorâmica e de profundidade de cena, como o *Structure Sensor* e *Tango Project* do *Google*.

Atualmente, como resultado de anos de pesquisas, a indústria está começando a mostrar soluções reais para conseguir capturar informação plenóptica de maneira eficiente. Um exemplo disto é a câmera *Lytro Immerge*, que permite capturar um campo de iluminação panorâmico, e promete resolver inúmeros problemas associados à produção de conteúdo. Entretanto, estas tecnologias ainda não são disponíveis ao público em geral e seu uso eficiente dependerá da solução a vários problemas ligados à interpretação e estruturação do grande volume de dados capturados por este tipo de dispositivos.

Capítulo 3

Renderização com mapas de iluminação

A renderização de cenas complexas, com iluminação indireta, iluminação proveniente de ambientes reais e outros tipos de fontes de luz representou um desafio e despertou grande interesse na comunidade de computação gráfica nas últimas décadas. Atualmente a renderização foto-realista de cenas sintéticas com mapas de iluminação extraídos de ambientes reais é utilizada diariamente e demanda soluções eficientes, uma vez que técnicas de incorporação de objetos sintéticos em filmes e comerciais são muito procuradas.

Um problema central no processo de renderização é calcular a *equação de espalhamento* ou *scattering equation*

$$L_o(p,\omega_o) = \int_{\mathbb{S}^2} f(p,\omega_o,\omega_i) L_i(p,\omega_i) |\cos\theta_i| \mathrm{d}\omega_i. \tag{3.1}$$

No contexto de iluminação direta, o interesse está apenas na radiância incidente diretamente das fontes luminosas, denotadas por $L_d(p,\omega)$. Dado o carácter linear da iluminação, podemos somar as contribuições individuais de todas as luzes, de modo que a integral da equação (3.1) pode ser escrita como

$$\sum_{j=1}^{luzes} \int_{\mathbb{S}^2} f(p,\omega_o,\omega_i) L_{d(j)}(p,\omega_i) |\cos\theta_i| \mathrm{d}\omega_i, \tag{3.2}$$

onde $L_{d(j)}$ denota a radiância incidente em p proveniente da j-ésima fonte de luz e

$$L_d(p,\omega_i) = \sum_j L_{d(j)}(p,\omega_i). \tag{3.3}$$

Quando usamos um mapa de iluminação, deixamos de ter uma luz pontual e para obter

sua contribuição num ponto da cena é preciso calcular uma integral sobre $\mathbb{S}^2$. Mesmo se considerarmos que o mapa tem uma resolução finita, por exemplo 1024×512 pixels, teríamos 1024×512 somas, o que torna a solução computacionalmente inviável. Para reduzir este somatório, entram em cena os métodos de integração probabilísticos com estimadores de Monte Carlo. Então se escolhem N direções ω_i com algum método probabilístico, obtendo a estimativa

$$\frac{1}{N}\sum_{j=1}^{N}\frac{f(p,\omega_o,\omega_i)L_d(p,\omega_j)|\cos\theta_j|}{p(\omega_j)}. \tag{3.4}$$

Neste capítulo faremos um estudo das técnicas utilizadas estimar a iluminação direta para mapas de iluminação. Também apresentaremos uma técnica de pré-processamento multifuncional que permite trabalhar com Mapas de Iluminação de diversas formas: obtendo uma coleção de luzes direcionais mediante métodos de amostragem como Corte Mediano de Debevec [18]; ou Amostragem Hierárquica por Importância com Mosaicos de Penrose [47]; ou selecionando um número grande de amostras para ser passadas às rotinas de Monte Carlo de renderizadores como *PBRT*, *Luxrender* e *POV-Ray* entre outros. Também trataremos os Mapas de Iluminação de maneira intermediária, estratificando-os, selecionando regiões que serão transformadas em luzes direcionais e outras que manteremos para passá-las às rotinas de Monte Carlo. Um estudo teórico sobre a equação de espalhamento (3.1) pode ser encontrado no capítulo 15 do livro *"Physically Based Rendering: from theory to implementation"* de M. Pharr e G. Humphreys [48, 50].

3.1 Métodos tradicionais de renderização com mapas de iluminação

Como escolher as fontes de luz para obter uma iluminação realista quando a informação da qual dispomos é um mapa de iluminação omnidirecional? Basicamente podemos dividir a literatura em duas abordagens: a primeira consiste em aplicar algum algoritmo de seleção de amostras sobre o mapa de iluminação para criar um conjunto de luzes direcionais. A segunda opção é utilizar o mapa de iluminação como uma luz de área a uma distância infinita que envolve a cena e aplicar algoritmos de seleção de amostras durante o cálculo de iluminação.

3.1.1 Aproximação do mapa de iluminação com luzes direcionais

A aproximação de um mapa de iluminação com um conjunto de luzes direcionais (*DL*), pode ser realizada utilizando métodos de amostragem. Dois dos métodos mais populares são:

3.1.1.1 Amostragem por Corte Mediano

O método fornece a posição das amostras no mapa e a iluminação total da célula à qual pertence cada amostra. As posições (u,v) das amostras no mapa são transformadas para suas correspondentes direções (x,y,z) na esfera unitária, obtendo-se um conjunto Ω de luzes direcionais. O método, apresentado por Debevec [18], utiliza o algoritmo de corte mediano.

3.1.1.2 Amostragem Hierárquica por Importância com Mosaicos de Penrose

Esta estratégia tem como base o algoritmo hierárquico apresentado por Ostromoukhov [47]. O resultado da amostragem hierárquica é uma lista com a localização das amostras no mapa de iluminação. A partir da localização, calculam-se as posições (x,y,z) na esfera unitária. A seguir calculam-se as células de Voronoi esféricas [26] para as amostras, e finalmente é calculada a radiância em cada célula para obter a radiância de cada luz direcional.

O conjunto de luzes obtido pelos métodos mencionados é utilizado para renderizar a cena. Para ambos os métodos, a radiância de cada luz é calculada da mesma forma. A radiância total de um mapa é dada por:

$$E = \int_{\mathbb{S}^2} Rad(\omega)\mathrm{d}\omega = \int_0^{2\pi}\int_0^{\pi} Rad(\phi,\theta)\sin(\theta)\mathrm{d}\theta\mathrm{d}\phi.$$

Logo, para um mapa de iluminação $\mathbb{M}$ de resolução $2N \times N$ no formato equiretangular

$$E = \int_0^{2\pi}\int_0^{\pi} Rad_{\mathbb{M}}(\phi,\theta)\sin(\theta)\mathrm{d}\theta\mathrm{d}\phi = \frac{2\pi\pi}{2N^2}\sum_{i=0}^{2N-1}\sum_{j=0}^{N-1} Rad_{\mathbb{M}}(i,j)\cdot\sin\left(\frac{(j+\frac{1}{2})\cdot\pi}{N}\right).$$

Desta forma, a radiância da célula $C(p_k)$ correspondente à amostra p_k é:

$$Rad_{C(p_k)} = \frac{2\pi\pi}{2N^2}\sum_{(i+\frac{1}{2},j+\frac{1}{2})\in C(p_k)} Rad_{\mathbb{M}}(i,j)\cdot\sin\left(\frac{(j+\frac{1}{2})\cdot\pi}{N}\right). \tag{3.5}$$

A renderização com luzes direcionais é simples de ser implementada, a iluminação na cena é uniforme, pois todas as luzes são utilizadas na renderização de cada pixel, portanto

a cena apresenta pouca variância local. Entre os inconvenientes podemos mencionar a alta quantidade de luzes necessárias para obter efeitos realistas na renderização e o problema de penumbras e sombras com bordas definidas por causa da discretização do ambiente de iluminação. Este último problema pode ser evitado, na maioria dos casos, aumentando o número de luzes. Porém, isto pode acarretar em tempo de computação excessivo.

3.1.2 Amostragem direta por importância

Dado um mapa de iluminação M_o, a amostragem direta é utilizada quando o mapa é considerado como uma luz de área infinita envolvendo a cena. O método de amostragem direta escolhido foi o de amostragem por importância (*IS*), descrito por Pharr e Humphreys [49].

A amostragem direta por importância permite obter renderizações fisicamente corretas de boa qualidade utilizando poucas amostras, dependendo da distribuição das fontes de iluminação na cena. Este método também proporciona sombras com bordas suaves e efeitos realistas em penumbras. A principal desvantagem é a granulosidade na imagem final devido à variância associada aos métodos probabilísticos, que pode ser reduzida aumentando o número de amostras. Porém esta solução resulta computacionalmente cara.

3.1.3 Mapas de pré-amostragem

Uma alternativa à utilização de luzes direcionais e à amostragem direta do mapa, o método do mapa de pré-amostragem (*MPA*), consiste em gerar um grande número de amostras a partir do mapa de iluminação, e passá-las ao integrador para serem utilizadas durante a renderização, em lugar de utilizar o mapa ou mesmo uma quantidade fixa de luzes direcionais (ver figura (3-1)).

Desenvolvemos o plugin *PRESAMP* para o *PBRT* [50], que utiliza um mapa de pré-amostragem para realizar a renderização. Na criação do mapa de pré-amostragem é preciso guardar as seguintes informações:

- a posição das amostras no mapa de iluminação;
- uma imagem equiretangular da partição do mapa em regiões de influência das amostras obtidas;
- uma lista da radiância total da região de influência de cada amostra.

O software desenvolvido permite criar mapas de pré-amostragem com dois algoritmos:

3.1.3.1 Amostragem por Corte Mediano

o mapa de pré-amostragem é gerado facilmente, pois o algoritmo devolve uma lista com as posições das amostras e a radiância de cada região de influência. A imagem da partição pode ser calculada durante a execução do algoritmo, utilizando as divisões sucessivas feitas no mapa, e guardadas no final.

3.1.3.2 Amostragem Hierárquica por Importância com Mosaicos de Penrose

o mapa de pré-amostragem será gerado da seguinte maneira:

- obtém-se o conjunto de amostras e suas posições;
- é calculada a partição de Voronoi da esfera unitária para o conjunto de amostras;
- a partição de Voronoi obtida é guardada no formato Equiretangular;
- para cada amostra é calculada a radiância total da sua célula de Voronoi.

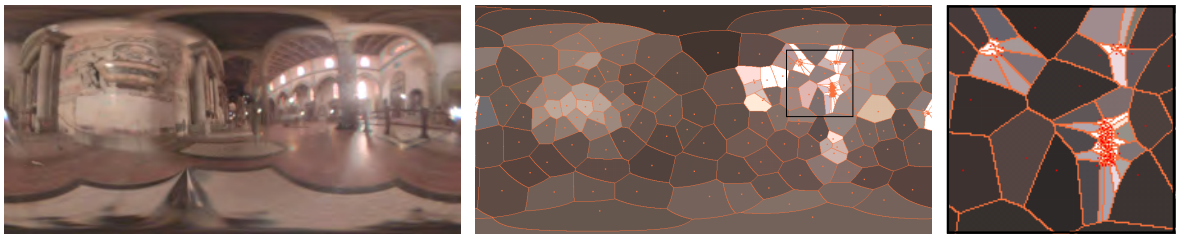


Figura 3-1: Mapa de pré-amostragem com 256 amostras do mapa *Galileo*, gerada com Amostragem Hierárquica por Importância [47].

3.2 Esquema híbrido para renderização

Como vimos na seção anterior, as abordagens tradicionais do problema de iluminação baseada em imagens é feita de duas maneiras: uma gerando um conjunto de luzes direcionais ou pontuais a partir de um mapa de iluminação e renderizando a cena com este conjunto. A segunda alternativa é utilizar o mapa de iluminação para obter uma função de importância e tomar amostras durante a renderização para calcular posições numa luz de área definida pelo mapa de iluminação. A seguir apresentamos o Esquema Híbrido (*HIB*) que permite utilizar

na mesma renderização tanto luzes direcionais quanto amostragem direta do mapa, segundo a necessidade do problema.

3.2.1 Separação do mapa de iluminação

Primeiramente, estratificamos o mapa de iluminação $\mathbb{M}$ utilizando a função de luminância para definir duas regiões A e B, tais que $A \bigcup B = \mathbb{M}$ e $A \bigcap B = \Phi$. A estratificação é feita de maneira que a região A concentre as zonas com de maior luminância, e a região B as zonas de menor luminância.

Consideremos o mapa de iluminação equiretangular $\mathbb{M} : [0, 2\pi] \times [0, \pi] \to \mathbb{R}^3$ e a função de luminância $L : \mathbb{R}^3 \to \mathbb{R}$ definida por $L(r,g,b) := 0.2125 \cdot r + 0.7154 \cdot g + 0.0721 \cdot b$. A luminância total do mapa $\mathbb{M}$ sobre a esfera é:

$$E_{\mathbb{M}} = \int_0^{2\pi} \int_0^{\pi} L_{\mathbb{M}}(\phi, \theta) \sin(\theta) \mathrm{d}\theta \mathrm{d}\phi.$$

Dado um limiar real $p \in [0,1]$, as regiões A e B são determinadas de forma tal a satisfazer as seguintes condições:

$$\begin{aligned} &\int_0^{2\pi} \int_0^{\pi} \xi_A(\phi, \theta) \cdot L_{\mathbb{M}}(\phi, \theta) \sin(\theta) \mathrm{d}\theta \mathrm{d}\phi = p \cdot E_{\mathbb{M}} \\ &\int_0^{2\pi} \int_0^{\pi} \xi_B(\phi, \theta) \cdot L_{\mathbb{M}}(\phi, \theta) \sin(\theta) \mathrm{d}\theta \mathrm{d}\phi = (1-p) \cdot E_{\mathbb{M}} \\ &\forall (\phi_A, \theta_A) \in A, \forall (\phi_B, \theta_B) \in B, L_{\mathbb{M}}(\phi_A, \theta_A) \geq L_{\mathbb{M}}(\phi_B, \theta_B) \end{aligned} \tag{3.6}$$

Assim, o estrato A concentra $p \cdot 100\%$ da luminância total do mapa $\mathbb{M}$ nas regiões de maior radiância, e B contém $(1-p) \cdot 100\%$ da luminância nas regiões de menor radiância.

A área dos estratos A e B considerados sobre a esfera unitária é dada por:

$$\text{Área}_{\mathbb{S}^2}(A) = \int_{\mathbb{S}^2} \xi_A(\omega) \mathrm{d}\omega = \int_0^{2\pi} \int_0^{\pi} \xi_A(\phi, \theta) \cdot \sin(\theta) \mathrm{d}\theta \mathrm{d}\phi, \tag{3.7}$$

$$\text{Área}_{\mathbb{S}^2}(B) = \int_{\mathbb{S}^2} \xi_B(\omega) \mathrm{d}\omega = \int_0^{2\pi} \int_0^{\pi} \xi_B(\phi, \theta) \cdot \sin(\theta) \mathrm{d}\theta \mathrm{d}\phi, \tag{3.8}$$

onde: $\xi_\Omega : \Delta \to \{0,1\}$ é a função característica de $\Omega \subset \Delta$,

$$\xi_\Omega(\omega) = \begin{cases} 1 & \text{se} \quad \omega \in \Omega \\ 0 & \text{se} \quad \omega \notin \Omega \end{cases}$$

3.2.1.1 Implementação computacional da estratificação:

Na prática, como o mapa de iluminação $\mathbb{M}$ é uma imagem de $2N \times N$ pixels, trabalhamos com uma discretização da esfera, obtida pelo mapeamento esférico do mapa de iluminação. A implementação é realizada da seguinte forma:

- Dado o mapa $\mathbb{M}$, com $2N \times N$ pixels, criamos um array $\overline{M}[2N^2, 2]$ definido da seguinte forma:

$$\overline{M}[k,1] := L_{\mathbb{M}}\left(\text{mod}(k,2N), \left\lfloor \frac{k}{2N} \right\rfloor\right),$$
$$\overline{M}[k,2] := k.$$

- O array $\overline{M}$ é ordenado de forma decrescente segundo a primeira coordenada, $\overline{M}[x,1]$. Dado um limiar $p \in [0,1]$, para obter a região A tal que

$$\int_0^{2\pi}\int_0^{\pi} \xi_A(\phi,\theta) \cdot L_{\mathbb{M}}(\phi,\theta)\sin(\theta)\mathrm{d}\theta\mathrm{d}\phi \approx p \cdot E_{\mathbb{M}},$$

é necessário determinar o menor valor inteiro r tal que

$$\sum_{i=0}^{r} \overline{M}[i,1] \cdot \sin\left(\pi \frac{\left\lfloor \frac{\overline{M}[i,2]}{2N} \right\rfloor + \frac{1}{2}}{N}\right) \geq p \cdot E_{\mathbb{M}}.$$

O valor $r < 2N^2$ representa a área do estrato A em pixels na representação equiretangular plana, pois consideramos cada pixel como sendo um quadrado de lado um. O valor da área do estrato A depois do mapeamento esférico é obtido aplicando a deformação de área a cada $\overline{M}[i,2]$ para $0 \leq i \leq r$ e realizando o somatório:

$$\mathcal{A}_{esf}(A) = \sum_{i=0}^{r} \sin\left(\frac{\left\lfloor \frac{\overline{M}[i,2]}{2N} \right\rfloor + \frac{1}{2}}{N} \cdot \pi\right). \tag{3.9}$$

Agora é possível definir dois mapas $\mathbb{M}_A$ e $\mathbb{M}_B$, tais que

$$\mathbb{M}_A(\phi,\theta) = \mathbb{M}(\phi,\theta)\cdot\xi_A(\phi,\theta),$$

$$\mathbb{M}_B(\phi,\theta) = \mathbb{M}(\phi,\theta)\cdot\xi_B(\phi,\theta),$$

Os mapas $\mathbb{M}_A$ e $\mathbb{M}_B$ podem ser descritos em função da imagem digital do mapa de iluminação $\mathbb{M}$ e do valor r

$$\mathbb{M}_A(x,y) = \begin{cases} \mathbb{M}(x,y) & \text{se } \exists\ 0 \le k \le r \text{ tal que } \overline{M}[k,2] = x + 2N\cdot y, \\ 0 & \text{se } \nexists\ 0 \le k \le r \text{ tal que } \overline{M}[k,2] = x + 2N\cdot y, \end{cases}$$

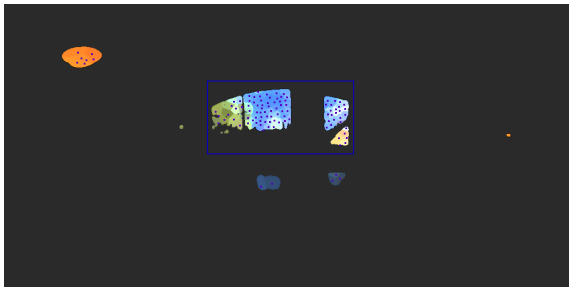
$$\mathbb{M}_B(x,y) = \begin{cases} \mathbb{M}(x,y) & \text{se } \exists\ r < k < 2N^2 \text{ tal que } \overline{M}[k,2] = x + 2N\cdot y, \\ 0 & \text{se } \nexists\ r < k < 2N^2 \text{ tal que } \overline{M}[k,2] = x + 2N\cdot y, \end{cases}$$

obtendo uma formulação de fácil implementação computacional.

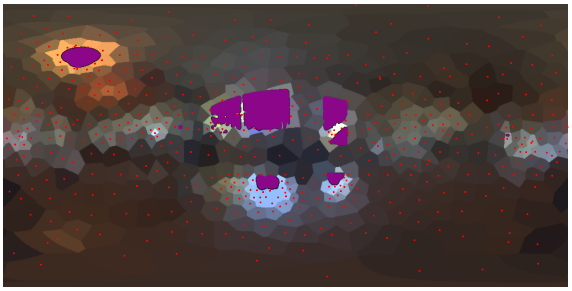
Em geral, para um mapa de iluminação proveniente de um ambiente real, a área da região A para um limiar $p = 0.5$ representa menos que o 5% da área total do mapa. Por exemplo, no mapa *Kitchen* a área da região A para $p = 0.6371$ é aproximadamente 2.8% da área do mapa, figura (3-2). Esta acumulação de radiância numa pequena área pode ser aproveitada selecionando amostras da região A e convertendo-as num conjunto Ω de luzes direcionais, figura (3-2a). Como a área é pequena a região pode ser representada com poucas amostras, i.e. poucas luzes direcionais. A região complementar B, cuja área representa a maior parte do mapa de iluminação, pode ser tratada de duas formas:

- Guardar o estrato B como um mapa HDR $\mathbb{M}_B$, onde os pixels do estrato A têm radiância nula. Neste caso a renderização da cena é feita utilizando amostragem direta por importância (IS) sobre o mapa $\mathbb{M}_B$, enquanto o conjunto de luzes direcionais Ω obtidas da região A é renderizada de maneira tradicional.

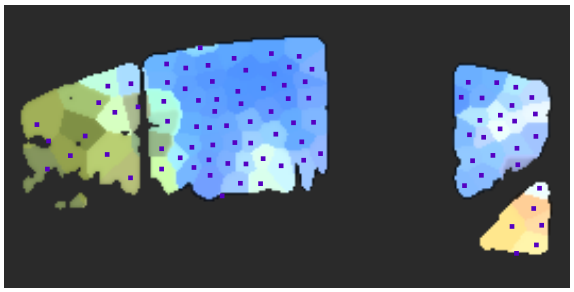
- Amostrar o estrato B com o método de Amostragem Hierárquica por Importância com Mosaicos de Penrose [47] ou de Corte Mediano [18], e guardar um mapa de pré-amostragem $\mathbb{M}_{PA}$, figura (3-2b). A renderização final é feita com o plugin *PRESAMP*, criado para o *PBRT* [50], para renderizar a iluminação dada pelo mapa $\mathbb{M}_{PA}$, juntamente com a renderização do conjunto Ω de luzes direcionais extraídas da região A.



(a) Estrato A, área= 14730 pixels. Geração de 92 luzes com Amostragem Hierárquica por Importância sobre o estrato A.



(b) Estrato $B = Mapa - A$, área= 509558 pixels. Mapa de pré-amostragem do estrato B com 382 amostras.



(c) Detalhe da amostragem realizada em (a).

(d) Mapa de Iluminação *Kitchen*.

Figura 3-2: Estratos A e B do Mapa HDR *Kitchen* segundo o limiar de variação máxima $p = 0.6371$. (a) Foi realizada uma clusterização com frequência $F_i = 1$ para obter as 92 luzes finais.

3.2.2 Determinação dos estratos e limiar de separação p

Um aspecto importante na abordagem híbrida é a escolha certa do limiar p utilizado para realizar a separação do mapa de iluminação, devido a que o valor de p está relacionado diretamente com a área do mapa destinada à criação de luzes direcionais e a área a ser utilizada para amostragem direta ou pré-amostragem. Uma boa escolha de p ajudará a reduzir o número de luzes direcionais utilizadas e a maximizar a iluminação total das mesmas. Estudamos alternativas para realizar estimativas do limiar p, para dar ao usuário uma aproximação prévia junto com a possibilidade de ajustar este valor manualmente. Além da escolha direta do limiar p por parte do usuário, apresentamos a seguir uma estimativa automática do limiar p apropriada para a maioria dos casos.

3.2.2.1 Estimativa pela taxa de variação máxima

O limiar de separação p é determinado a partir do valor mínimo sobre uma função. Primeiramente consideremos a função constante por partes obtida a partir do array $\overline{M}$:

$$f(x) = \begin{cases} \overline{M}[0,1] & \text{se} \quad 0 \leq x \leq \sin\left(\pi \cdot \frac{\left\lfloor \frac{\overline{M}[0,2]}{2N} \right\rfloor + \frac{1}{2}}{N}\right) \\ \overline{M}[i,1] & \text{se} \quad x_{i-1} < x \leq x_i \quad i = 1, \cdots, 2N^2 - 1 \end{cases}$$

onde

$$x_i = \sum_{j=0}^{i} \sin\left(\pi \cdot \frac{\left\lfloor \frac{\overline{M}[j,2]}{2N} \right\rfloor + \frac{1}{2}}{N}\right). \tag{3.10}$$

Seja $\psi = 2N \cdot \sum_{i=0}^{2N^2-1} \sin\left(\pi \cdot \frac{i+0.5}{N}\right)$. A função $f : [0, \psi] \to \mathbb{R}$ está definida para $0 \leq x \leq \psi$. Definimos agora a função integral linear por partes:

$$g : [0, \psi] \to \mathbb{R}, \qquad g(x) = \int_0^x f(x)\mathrm{d}x.$$

Definimos a seguir a função

$$\zeta(x) = \frac{f(0) + f(x)}{2} \cdot x + \frac{f(x) + f(\psi)}{2}(\psi - x).$$

A função $\zeta(x)$ representa uma aproximação da integral da luminância do mapa de iluminação pela soma das áreas de dois trapézios, como na figura (3-3) (a). Neste esquema, a melhor relação entre área e iluminação para o estrato $A(x)$ é obtido no ponto $\bar{x}$, que minimiza a função $\zeta(x)$. Tanto o cálculo do valor $\bar{x}$ quanto a implementação computacional são simples, pois a função ζ é avaliada numa quantidade finita de pontos x_i, para $i = 0, \cdots, 2N^2 - 1$, associados aos pixels do mapa de iluminação $\mathbb{M}$. Calculamos o conjunto de mínimos globais:

$$\Psi = \left\{\bar{x}_j : \zeta(\bar{x}_j) \leq \zeta(x_i); \forall i, j = 0, \cdots, 2N^2 - 1; i \neq j\right\}$$

onde os x_k's são definidos pela equação (3.10). Claramente o conjunto $\Psi \neq \emptyset$, e no caso de termos $\#(\Psi) > 1$ tomamos $\bar{x} = \min_x \Psi$, dado que $\text{Área}_{esf.}(A(x)) = x$ e principalmente pretendemos minimizar a área de $A(x)$.

Uma vez determinado o valor de $\bar{x}$, o valor da área do estrato A será o próprio $\bar{x}$. O cálculo do valor do limiar de separação de luminância p segue de calcular o menor inteiro r tal que

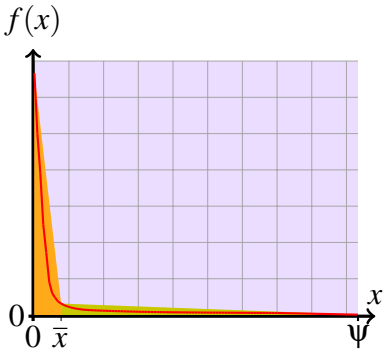
$$\bar{x} \leq x_r = \sum_{j=0}^{r} \sin\left(\pi \cdot \frac{\left\lfloor \frac{\overline{M}[j,2]}{2N} \right\rfloor + \frac{1}{2}}{N}\right). \tag{3.11}$$

Calculado o valor de r, obtemos o valor de p como sendo

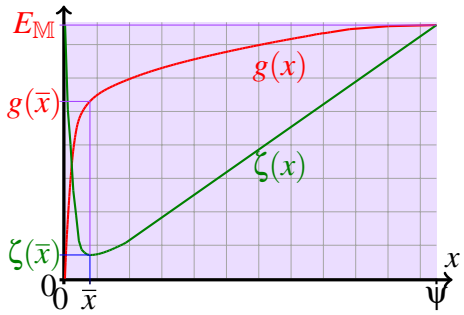
$$p = \frac{\sum_{i=0}^{r} \overline{M}[i,1] \cdot \sin\left(\pi \cdot \frac{\left\lfloor \frac{\overline{M}[i,2]}{2N} \right\rfloor + \frac{1}{2}}{N}\right)}{\sum_{i=0}^{2N^2-1} L_{\mathbb{M}}\left(\mathrm{mod}(i,2N), \left\lfloor \frac{i}{2N} \right\rfloor\right) \cdot \sin\left(\pi \cdot \frac{\left\lfloor \frac{i}{2N} \right\rfloor + \frac{1}{2}}{N}\right)}. \tag{3.12}$$

3.2.3 Clusterização

Juntamente com a estratificação, também tivemos cuidado de introduzir controles adicionais, dado que o método de Amostragem Hierárquica por Importância com Mosaicos de Penrose de Ostromoukhov e Donohue [47] apresenta alguns inconvenientes. Como o método de Ostromoukhov e Donohue [47] aplicado a um mapa de iluminação faz amostragem hierárquica por importância de uma região retangular contínua do plano com as dimensões do mapa, e função de importância constante por partes igual à luminância do mapa, tem-se frequentemente uma alta condensação de amostras em regiões de área muito pequena. Por exemplo, muitas vezes temos duas ou mais amostras numa área que corresponde a um pixel do mapa. Este fenômeno acontece porque nos mapas extraídos de ambientes reais a variação de luminância é muito grande. Como o objetivo é obter luzes direcionais que iluminem a cena, precisamos de amostras devidamente espalhadas para cobrir bem a região A (por este motivo não temos interesse em amostras cujos vizinhos estão a menos de 1 pixel de distância). Por fim, agrupamos estas amostras, aplicando um método de clusterização sugerido por Velho



(a) Função monótona não crescente de luminância do mapa *Kitchen*, $f(x)$.



(b) •: Função $\zeta(x)$ para o mapa *Kitchen*.
•: Integral $g(x)$ da função $f(x)$.

Figura 3-3: Funções $f(x)$, $g(x)$ e $\zeta(x)$ utilizada para estratificar o mapa *Kitchen*.

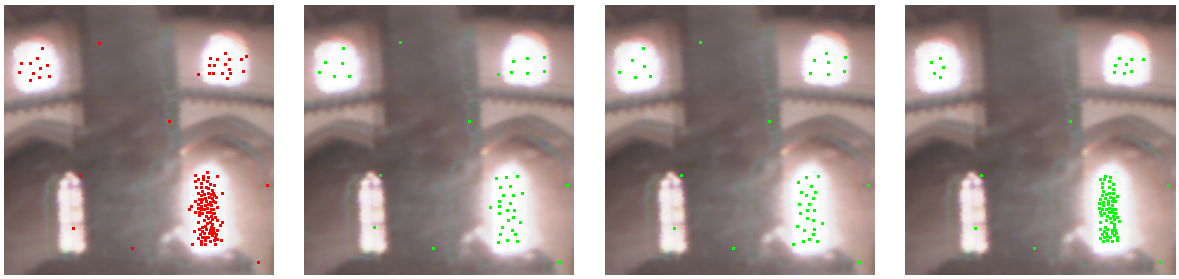
e Gomes [56], eliminando desta forma as amostras redundantes. Implementamos em nosso sistema a clusterização com três opções para a frequência F (veja exemplos na figura (3-4)):

$$\mathbf{F_1} = 1; \qquad \mathbf{F_2} = \int_{Vor(c_i)} L(x)dx; \qquad \mathbf{F_3} = \frac{\int_{Vor(c_i)} L(x)dx}{\text{Área } Vor(c_i)};$$

Dado um cluster com dois elementos $K = \{c_i, c_j\}$, o nível de quantização ótima utilizando a métrica do quadrado da distância geodésica sobre a esfera é:

$$c = \frac{F_{t,i}}{F_{t,i}+F_{t,j}}c_i + \frac{F_{t,j}}{F_{t,i}+F_{t,j}}c_j, \qquad t \in \{1,2,3\},$$

$$E(c_i,c_j) = \frac{F_{t,i}\cdot F_{t,j}^2 + F_{t,j}\cdot F_{t,i}^2}{(F_{t,i}+F_{t,j})^2}\|\arccos(c_i\cdot c_j)\|^2, \qquad t \in \{1,2,3\}.$$



(a) Sem clusterização. **(b)** Frequência F_1 **(c)** Frequência F_2 **(d)** Frequência F_3

Figura 3-4: Detalhes de uma clusterização do mapa *Galileo* com 304 amostras iniciais para 200 amostras clusterizadas.

A frequência F_1 leva em conta somente a distância entre as amostras e ignora a luminância das mesmas e a área da sua célula de Voronoi. É recomendada quando se deseja suprimir amostras próximas sem levar em conta sua luminosidade. Portanto, pode ser utilizada quando quisermos obter um conjunto de luzes direcionais a partir de um mapa de iluminação com alta concentração de luminância em regiões pequenas pelo método de Ostromoukhov.

A frequência F_2 é calculada como a luminância da célula de Voronoi da amostra. Pode ser utilizada quando temos interesse em juntar amostras com baixa luminância.

A frequência F_3 é dada pela luminância média da célula de Voronoi da amostra. Em termos de distribuição probabilística, é a mais adequada, pois leva em conta a distância das amostras como sua importância em função da luminância média.

3.2.3.1 Critérios de parada

O processo de clusterização usa dois critérios de parada.

- **Parada por número de clusters:** Dado um conjunto Ω com P amostras, a clusterização é realizada sucessivamente até atingir o valor $D \leq P$ desejado pelo usuário. No método de clusterização por pares de pontos a quantidade de clusters diminui numa unidade a cada interação, logo são realizadas $P - D$ aglomerações.
- **Parada por separação mínima:** Dado um conjunto Ω com P amostras e um limiar α de separação angular mínima, a clusterização é realizada até que a distância angular entre os clusters seja maior ou igual a α, isto é, até termos: $\alpha \leq \arccos(c_i \cdot c_j), \quad \forall i \neq j$.

3.3 Resultados

O *esquema híbrido* proposto neste capítulo requer o processamento do mapa de iluminação para gerar a iluminação híbrida que será utilizada na renderização. Desenvolvemos o software *Hybrid Sampling Machine (HSM)* [59] para realizar o pré-processamento do mapa de iluminação e gerar a iluminação que será utilizada pelo renderizador. Os experimentos foram executados no software *PBRT* [50].

Para trabalhar com mapas de pré-amostragem foi desenvolvido o plugin *PRESAMP* para o *PBRT* [50], que realiza amostragem por importância a partir de um mapa de pré-amostragem. Com o plugin *PRESAMP*, mais os plugins do *PBRT* é possível implementar os diversos esquemas de amostragem descritos anteriormente, utilizando-os separadamente ou combinados. Realizamos comparações numéricas utilizando quatro estratégias de renderização:

- **DL**: Aproximação do mapa de iluminação por luzes direcionais, através do método de Corte Mediano (**MC**) [18] e Amostragem com Mosaicos de Penrose (**OS**) [47];
- **IS**: Amostragem direta por importância utilizando o plugin *IS*, *PBRT* [50];
- **MPA**: Amostragem direta por importância utilizando mapas de pré-amostragem, obtidos com **MC** e **OS**, utilizando o plugin *PRESAMP*, *PBRT* [50];
- **HIB**: Método híbrido. Separação do mapa original e renderização utilizando luzes direcionais + amostragem por importância direta ou mapas de pré-amostragem.

Na tabela (3.1) colocamos alguns dos resultados obtidos para o cálculo de iluminação direta pelos métodos tradicionais, de pré-amostragem e o esquema híbrido proposto. A partir de um conjunto maior de testes elaboramos o gráfico da figura (3-5) que mostra a relação entre as três abordagens mencionadas anteriormente. Na figura (3-5) é possível observar que o *esquema híbrido* tem um desempenho notável para quantidades baixas de amostras. A figura (3-6) mostra dois resultados obtidos utilizando amostragem por importância (*I.S.*) e o *Esquema Híbrido* (*HIB.*), ambos com erro quadrado médio (*MSE*) e qualidade visual semelhantes porém tempos de renderização muito diferentes. Na figura (3-8) mostramos as renderizações de uma cena 3D com os parâmetros da tabela (3.1) e detalhes ampliados de uma região das mesmas, onde se pode ver a diferença de qualidade e os tempos de renderização.

Também realizamos testes com integradores de iluminação global tais como *photon mapping* (*mapeamento de fótons*) e *path tracing* (*traçado de caminhos*). Obtivemos bons resultados com estes dois métodos e os mesmos podem ser apreciados nas figuras (3-9) e (3-7).

Tabela 3.1: Renderizações do modelo *Killeroo* com o mapa *Galileo*. A Imagem de referência utilizada para o cálculo do erro foi renderizada com *IS* no *PBRT* [50], com 32768 (64x512) amostras. •: Amostragem por importância (IS), •: Mapas de pré-amostragem (MPA), •: Esquema Híbrido (HIB).

			Amostragem direta	Mapa de Pré-amostragem	Esquema Híbrido HIB		
Fig.	Tempo Seg.	Euc.MSE σ^2	IS Amostras	MPA Met(Pré-Amost):amostras	% Área Mapa *A*	Mapa *A* DL	Mapa *B* método:amostras
3-8a	156.4	0.021705	32				
	312.4	0.012404	64				
3-8c	604.3	.009375	128				
	1225.0	0.008154	256				
	129.0	0.025096		OS(1310):32			
	255.0	0.011824		OS(1310):64			
	511.4	0.009452		OS(1310):128			
	91.8	0.013426			0.864	OS:40	OS(360):8
	122.7	0.009693			0.864	OS:40	OS(360):16
	183.5	0.008793			0.864	OS:40	OS(360):32
3-8e	151.0	0.009033			0.864	OS:64	OS(360):16
3-8g	214.2	0.008343			0.864	OS:64	OS(360):32
3-8i	492.3	0.007975			1.275	OS:128	IS:64

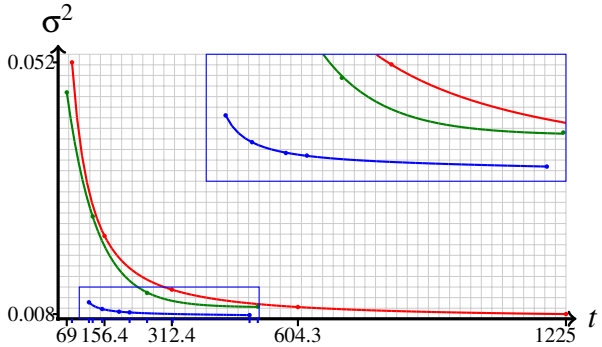
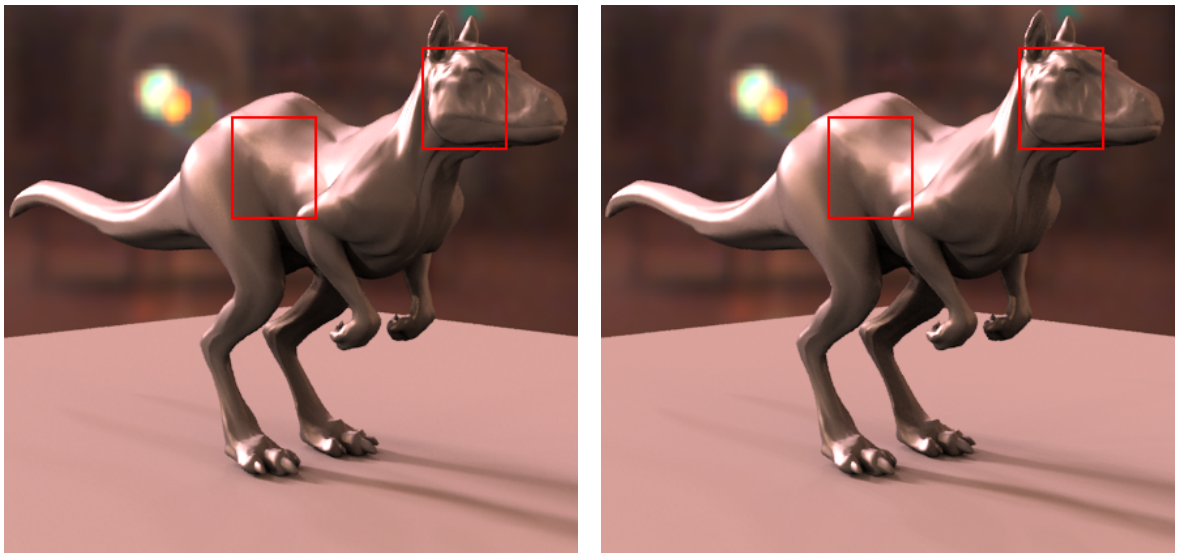


Figura 3-5: Aproximação por ajuste de curvas das funções de erro quadrado medio (MSE) euclidiano para alguns valores da tabela (3.1). •: Amostragem por importância com plugin *IS*; •: Amostragem por importância de Mapas de pre-amostragem, plugin *PRESAMP*; •: Método híbrido utilizando mapas de pre-amostragem + luzes direcionais.



(a) Amostragem por importância *I.S.* T = 604.3s **(b)** Esquema Híbrido *HIB.* T = 214.2s

Figura 3-6: Renderização com o Mapa *Galileo*.

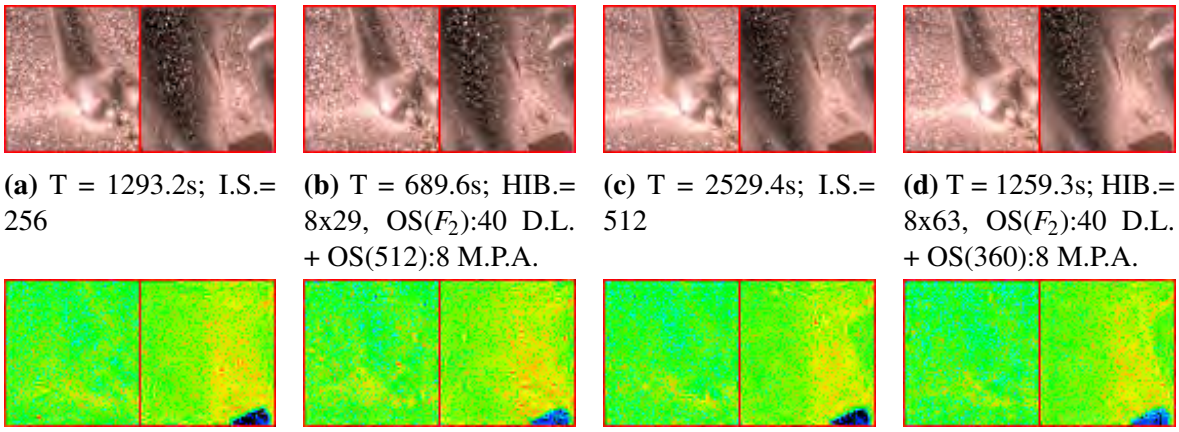
Na figura (3-9) podemos ver alguns exemplos renderizados com mapeamento de fótons. As figuras (3-9a), (3-9b) e (3-9c) são renderizações da mesma cena utilizando:

- Figura (3-9a): amostragem direta por importância com 64 amostras por raio,
- Figura (3-9b): 256 luzes direcionais geradas com o algoritmo de Corte Mediano,
- Figura (3-9c): *esquema híbrido* formado por 64 luzes direcionais obtidas pelo algoritmo de Ostromoukhov e Donohue [47] realizando clusterização com frequência F_2, mais um mapa de pré-amostragem do qual são utilizada 64 amostras por raio.

As figuras (3-9d), (3-9e) e (3-9f) foram renderizadas com amostragem direta por importância (*IS*) utilizando diferentes números de amostras. A figura (3-9i) foi renderizada utilizando um conjunto de 256 luzes direcionais. Nas figuras (3-9g) e (3-9h) a cena foi renderizada utilizando um *esquema híbrido* e mudando somente o número de amostras para antialiasing. Observe que o exemplo iluminado com luzes direcionais não apresenta granulosidade, mas apresenta sombras de contornos definidos. Por outro lado as imagens do *esquema híbrido* apresentam sombras suaves e pouca granulosidade e um tempo inferior a 50% das renderizações diretas de qualidade comparável.

Na figura (3-7) incluímos alguns resultados obtidos na renderização do modelo *killeroo* utilizando iluminação global com *path tracing*. Nas figuras (3-7a) e (3-7c) a renderização foi feita utilizando amostragem direta por importância do mapa de iluminação utilizando 256 e

512 amostras respectivamente. Nas figuras (3-7b) e (3-7d) foi utilizado um esquema híbrido para realizar a renderização utilizando luzes direcionais e mapas de pré-amostragem. As figuras (3-7e), (3-7f), (3-7g) e (3-7h) representam o erro quadrado médio (*MSE*) cometido na renderização das figuras (3-7a), (3-7b), (3-7c) e (3-7d) respectivamente. Para determinar o erro foi feita uma comparação com um modelo de referência renderizado com amostragem direta por importância pelo método de *path tracing* utilizando 32768 amostras por pixel. É possível observar que para valores de erro semelhantes o tempo de renderização do *esquema híbrido* é de aproximadamente 50% do tempo de renderização da amostragem direta.



(a) T = 1293.2s; I.S.= 256

(b) T = 689.6s; HIB.= 8x29, OS(F_2):40 D.L. + OS(512):8 M.P.A.

(c) T = 2529.4s; I.S.= 512

(d) T = 1259.3s; HIB.= 8x63, OS(F_2):40 D.L. + OS(360):8 M.P.A.

(e) Erro MSE = 0.237848

(f) Erro MSE = 0.214314

(g) Erro MSE = 0.119027

(h) Erro MSE = 0.110049

Figura 3-7: Renderização do modelo *Killeroo* com o Mapa *Galileo* utilizando iluminação global com *Traçado de Caminhos*.

3.4 Considerações

O *Esquema Híbrido* apresentado neste trabalho utiliza o algoritmo de amostragem hierárquica por importância, desenvolvido por Ostromoukhov e Donohue [47], e o algoritmo de amostragem por Corte Mediano, apresentado por Debevec [18], mas podem ser implementados outros algoritmos de amostragem.

Realizamos comparações dos métodos tradicionais, de geração de luzes direcionais a partir da amostragem do mapa, e amostragem direta por importância, com diversas configurações do *Esquema Híbrido* proposto. Em todas as situações o *Esquema Híbrido* foi igual ou mais eficiente que as técnicas tradicionais apresentadas. O grau de eficiência varia com a escolha dos parâmetros, o modelo de cena e o mapa utilizado.

Para conseguir um *Esquema Híbrido* eficiente tivemos que superar vários obstáculos, entre eles:

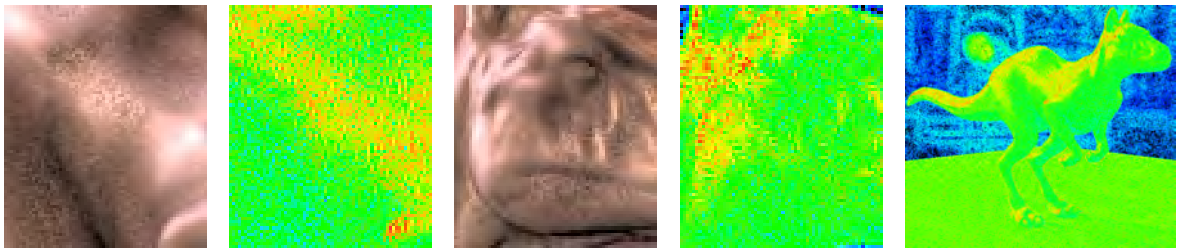
- Conseguir aproximações das regiões brilhantes do mapa utilizando poucas luzes direcionais. Implementamos um esquema de clusterização de amostras para evitar o aglutinamento de amostras causado pelos métodos de amostragem por importância.

- Obter distribuições de amostras e luzes direcionais na parametrização equiretangular, que fossem boas na representação esférica do mapa. A estes efeitos implementamos o cálculo de células de Voronoi na esfera para delimitar as regiões do mapa que contribuiriam com cada luz direcional.

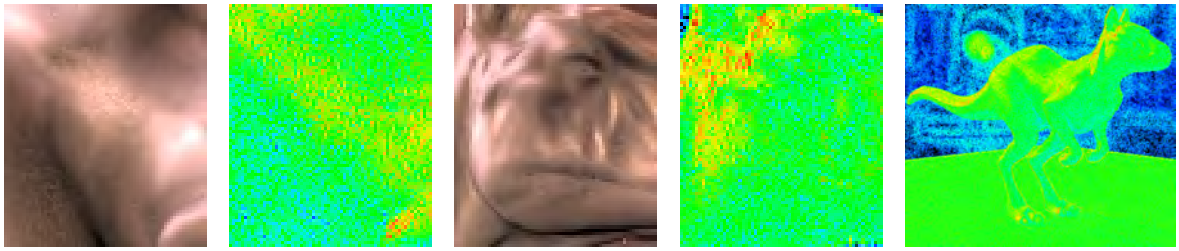
Também incluímos no software ferramentas de estimativa de parâmetros, como o cálculo automático de limiar de variação máxima, que permite separar o mapa de iluminação de forma automatizada, e tem fornecido excelentes resultados nas renderizações.

A proposta híbrida fornece resultados competitivos para o cálculo de iluminação direta, reduzindo o tempo de renderização em até 75% em certos casos. Também obtivemos resultados significativos para os principais métodos de iluminação global com eficiência de até 50% no tempo de renderização com o método de *path tracing* e até 75% com *photon mapping*.

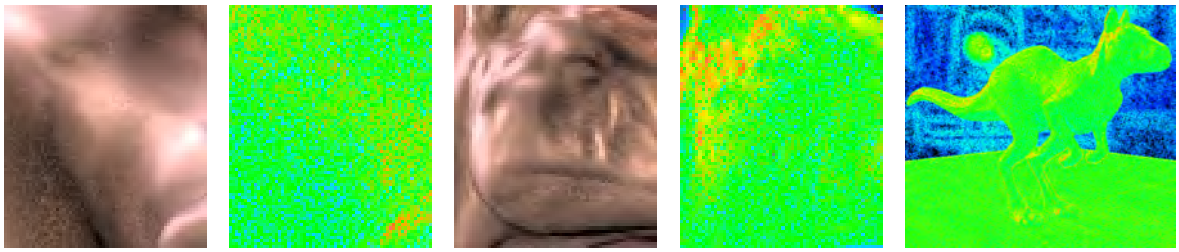
Em síntese o *Esquema Híbrido* fornece ao usuário, além da configuração automática, uma infinidade de possibilidades de configurações, mantendo um bom desempenho tanto em renderizações de iluminação direta quanto em métodos de iluminação global. Os processamentos realizados nos mapas de iluminação para gerar mapas de pré-amostragem serão utilizados nos próximos capítulos para realizar pré-processamentos da iluminação ao tratar do problema de inserir objetos sintéticos em cenas reais capturadas.



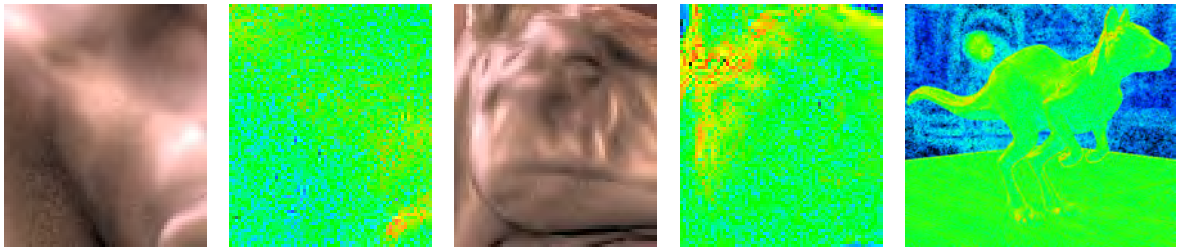
(a) Tempo = 156.4s; Método = *I.S.* **(b)** MSE = 0.021705



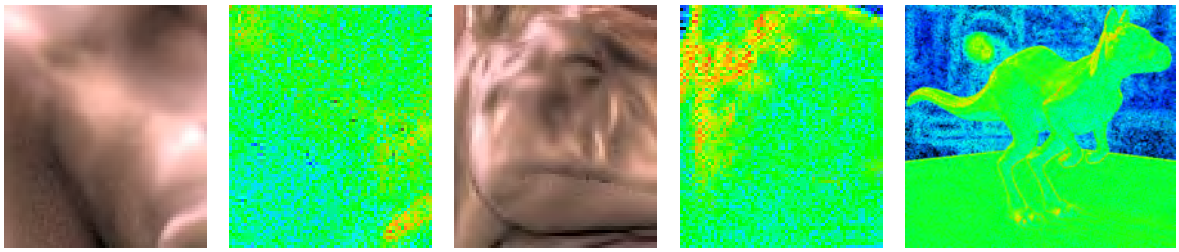
(c) Tempo = 604.3s; Método = *I.S.* **(d)** MSE = 0.009375



(e) Tempo = 151.0s; Método = *HIB.* **(f)** MSE = 0.009033



(g) Tempo = 214.2s; Método = *HIB.* **(h)** MSE = 0.008343



(i) Tempo = 492.3s; Método = *HIB.* **(j)** MSE = 0.007975

Figura 3-8: Detalhes da renderização do modelo *Killeroo* com o Mapa *Galileo*, correspondentes à tabela (3.1). Em todos os casos foram utilizados 4 raios por pixel para o anti-aliasing. Ao lado de cada detalhe colocamos uma imagem do erro utilizando um mapeamento logarítmico.

(a) T = 946.6s; I.S. = 64 amostras

(b) T = 889.7s; D.L.= 256 luzes com MC

(c) T = 910.1s; HIB.= OS(F_2):64 luzes + 64 amostras M.P.A.

(d) T = 525.9s; I.S. = 64 amostras, 4a. antialiasing

(e) T = 1379.4s; I.S. = 128 amostras, 8a. antialiasing

(f) T = 2129.6s; I.S. = 256 amostras, 8a. antialiasing

(g) T = 530.9s; HIB = OS(F_2):64 luzes + 64 amostras M.P.A.; 4a. antialiasing

(h) T = 1080.6s; HIB. = OS(F_2):64 luzes + 64 amostras M.P.A.; 8a.a.

(i) T = 1016s; D.L. = 256 luzes com MC; 8a. antialiasing

Figura 3-9: Renderização do modelo *Sponza* com o Mapa *Galileo* utilizando iluminação global com *Mapeamento de Fótons*.

Capítulo 4

Renderização foto-realista de objetos sintéticos em cenas reais

Para a indústria do cinema e outras que dependem de imagens foto-realistas, é indispensável contar com um pipeline eficiente que permita renderizar cenários baseados em iluminação real e completar cenas reais capturadas com elementos sintéticos de maneira foto-realista, i.e., de forma que os objetos inseridos artificialmente pareçam formar parte da cena real onde foram introduzidos. Usualmente muito tempo é gasto na combinação, ajuste e retoque digital das cenas reais e sintéticas, que em geral são renderizadas por separado e combinadas em uma imagem final. Como o ajuste manual nem sempre é fácil, o retoque e edição final resultam indispensáveis na maioria dos casos. Isto é aceitável em imagens estáticas, mas torna-se um problema em cenas dinâmicas, como um vídeo por exemplo.

Neste capitulo, estudaremos o problema de renderização foto-realista de objetos sintéticos imersos num cenário real, e vamos apresentar um método que permite produzir e renderizar cenas de realidade mista (i.e., cenas incluindo objetos reais e sintéticos) de uma maneira simples e direta. Como parte do trabalho desenvolvemos um software open source, *ARLuxrender*, para renderizar cenas de realidade mista. A partir deste momento iremos nos referir a uma cena real com objetos sintéticos pelos termos *cena aumentada, cena real aumentada* ou *cena de realidade mista*.

4.1 Produção de cenas de realidade mista

Renderizar uma cena onde parte dela é proveniente de um cenário real capturado e outra parte é modelada por computador é um problema usual na área de efeitos visuais para cinema. Em geral existem diversas abordagens para lograr uma renderização com estas características, mas todas elas têm alguns aspectos em comum. Por exemplo, é necessário capturar o cenário real e conhecer a posição, orientação e demais parâmetros da câmera utilizada. Também é preciso capturar ou modelar fielmente a iluminação existente na cena real, para assim poder iluminar os objetos sintéticos, e finalmente é preciso renderizar os novos elementos e combiná-los com o cenário real.

Motivados pelo trabalho de Paul Debevec [17], neste capítulo estudaremos o problema de renderização foto-realista de objetos sintéticos em cenas reais capturadas. Na abordagem que apresentaremos, tentamos lidar com alguns pontos críticos do processo de produção e renderização. No processo de renderização proposto por Debevec, a composição dos objetos sintéticos com o cenário real é feito numa etapa de pós-processamento, utilizando uma técnica de renderização diferencial. Para que a técnica diferencial forneça resultados satisfatórios é necessário estimar a refletância da cena local, próxima aos objetos sintéticos. Este processo é realizado de maneira iterativa e sua complexidade depende do tipo de *BRDF* com que a cena local é modelada. Debevec apresentou soluções para casos simples onde a cena local é representada por uma superfície difusa de cor constante. Nossa estratégia, embora não seja fisicamente correta, apresenta resultados foto-realistas na composição final sem necessidade de realizar o processo iterativo para estimar a *BRDF* nem combinar renderizações parciais da cena. É necessário destacar também que o processo que desenvolvemos pode ser inteiramente realizado com imagens de alta gama dinâmica, *HDR*, diferentemente de outros métodos que lidam unicamente com imagens *LDR*.

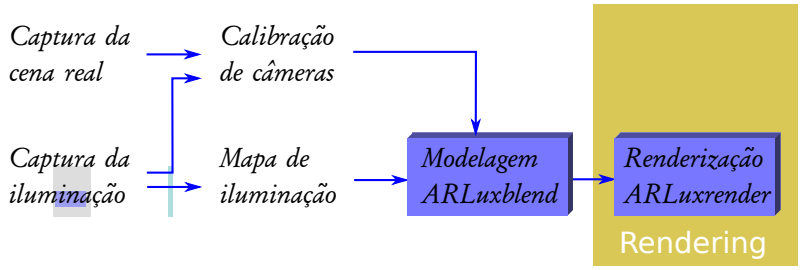


Figura 4-1: Framework de renderização foto-realista para cenas de realidade mista.

O processo de criação de uma cena de realidade mista pode ser descrito através de uma sequência de passos que devem ser seguidos até obter a renderização da cena final (figura (4-1)). Estes passos são:

1. Captura da imagem de fundo da cena real.
2. Captura da iluminação do ambiente real.
3. Calibração das câmeras principal e de captura de iluminação.
4. Montagem do mapa de iluminação *HDR* e da imagem de fundo *HDR* com as imagens obtidas.
5. Modelagem dos objetos sintéticos e da cena real local que interage com eles.
6. Renderização da cena com realidade mista (aumentada).

Antes de descrever o algoritmo de renderização, apresentaremos brevemente os passos prévios do processo de produção da cena de realidade mista, para esclarecer alguns aspectos na etapa de produção da cena.

4.1.1 Calibração de câmeras

Para realizar a modelagem da cena e ajustar a iluminação da mesma, é necessário conhecer a posição e orientação da câmera utilizada para capturar a cena e também os parâmetros da câmera utilizada para capturar a iluminação. Estes parâmetros podem ser calculados utilizando algum algoritmo de calibração de câmeras. Nos casos tratados nesta tese utilizamos o algoritmo de Tsai [54, 55] para realizar a calibração. Guardamos os dados da calibração para serem utilizados durante as etapas de modelagem e renderização da cena aumentada. O arquivo contêm o lookat, campo de visão (fov) e a distância focal da câmera principal. Também é guardada a matriz da transformação que determina a posição do mapa de iluminação. Todos os dados de posição estão no sistema de coordenadas utilizado. Utilizamos duas posições de câmera pois nem sempre será possível capturar a iluminação utilizando a posição da câmera principal. Ao utilizar duas posições de câmera é preciso lembrar-se de manter fixo o sistema de coordenadas do mundo para conseguir uma calibração correta, i.e., a posição do tabuleiro de calibração deve ser a mesma para ambas as câmeras. A figura (4-2) mostra uma calibração para duas posições de câmera.

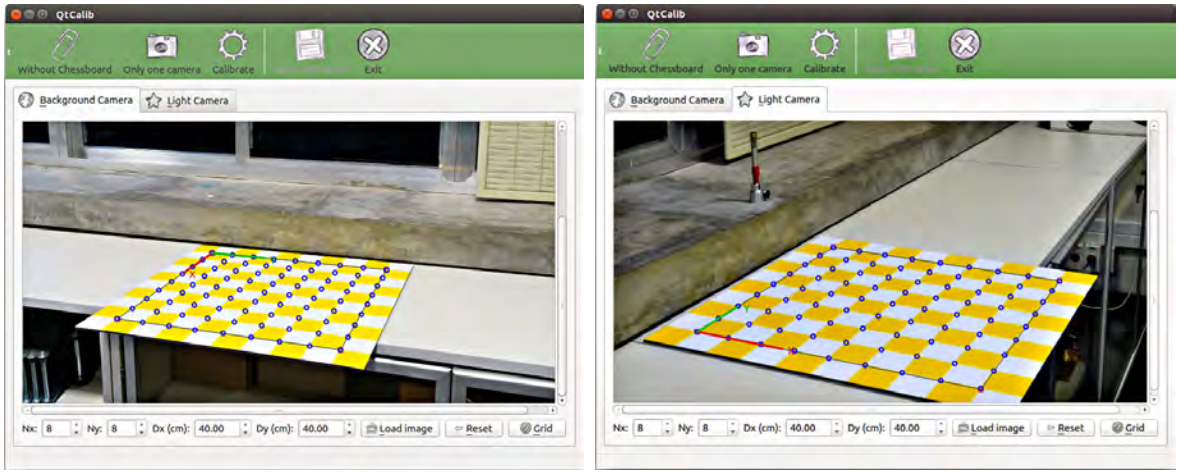


Figura 4-2: Calibração pelo método Tsai. A imagem da esquerda é utilizada para calibrar a câmera principal e a imagem da direita para calibrar a câmera de captura da iluminação.

4.1.2 Captura da iluminação do ambiente

Existem diversas formas de capturar e armazenar a informação de iluminação da cena. A forma clássica, utilizada por Debevec [17], consiste em tomar fotografias de uma esfera especular com diferentes tempos de exposição e montar uma imagem *HDR*, utilizando a técnica apresentada em 1997 por Debevec e Malik, [19]. Também é possível utilizar câmeras panorâmicas, como a *Ladybug* da *Point Grey*, que permitem capturar vídeo ou imagens omnidirecionais em *HDR*. Além destas duas abordagens, existem outras, como usar lentes olho de peixe para capturar uma serie de fotografias e compor elas numa imagem omnidirecional, técnica que exploraremos no próximo capítulo. Neste capítulo optamos por uma solução econômica e acessível, e nos decantamos pelo método de fotografar uma esfera especular.

Independentemente do método escolhido, a captura da iluminação deve ser realizada numa posição próxima à origem do sistema de coordenadas utilizado, para evitar distorções

(a) Câmera *Ladybug*. **(b)** Esfera especular. **(c)** Tabuleiro de calibração.

Figura 4-3: (c) Tabuleiro utilizado para marcar o sistema de coordenadas e realizar a calibração de câmera. A esfera especular precisa estar na origem do sistema de coordenadas.

perceptíveis nas sombras e reflexões produzidas pelo mapa de iluminação sobre os objetos sintéticos, figura (4-3c). Também é necessário que os objetos sintéticos inseridos na cena sejam posicionados próximos da origem do sistema de coordenadas para evitar as distorções citadas anteriormente. Uma descrição mais detalhada do processo de aquisição da iluminação pode ser encontrada no trabalho de Schulz et al. [1].

4.1.3 Modelagem da cena

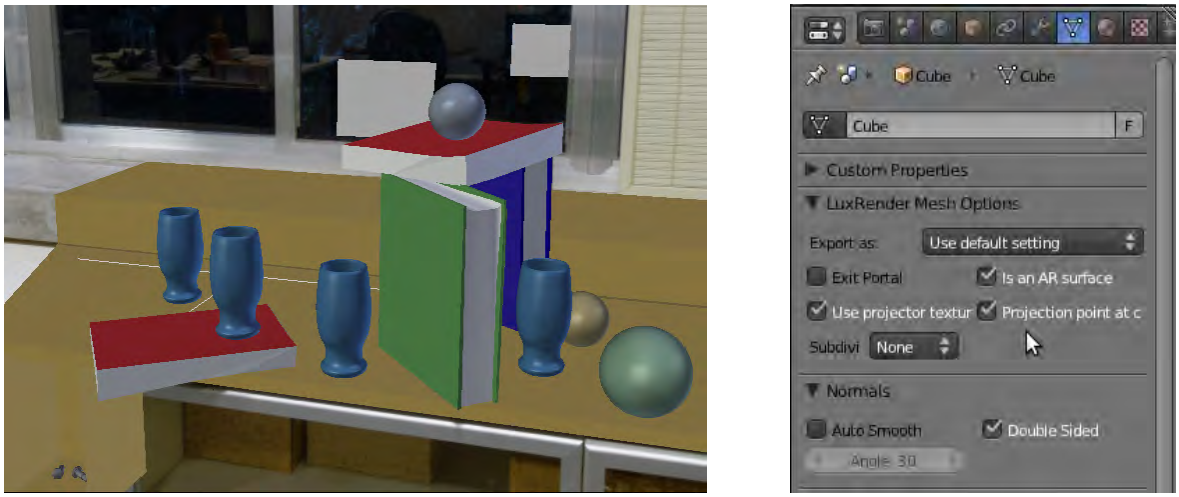
A modelagem da cena pode ser realizada em qualquer aplicativo de modelagem 3D. Optamos por utilizar software livre pela flexibilidade para desenvolver plugins e realizar modificações no código fonte. O algoritmo de renderização descrito na seção 4.2 foi implementado no branch *ARLuxrender* do *Luxrender*, um projeto de software livre baseado no *PBRT* [50]. Utilizamos *Blender* como software de modelagem e adaptamos o plugin *Luxblend* que exporta cenas modeladas em *Blender* para o formato de descrição de cena do *Luxrender*. O plugin *ARLuxblend*, exporta a cena modelada em *Blender* para a linguagem de descrição de cena do *ARLuxrender* e inclui os parâmetros adicionais necessários para agilizar o trabalho do artista durante a composição da cena.

Adicionalmente aos controles do *Luxblend*, o plugin *ARLuxblend* permite também:

- Definir os parâmetros da câmera e da iluminação através do arquivo de calibração;
- Criar e definir projetores de texturas, necessários para renderizar o fundo corretamente;
- Definir as propriedades das superfícies reais de suporte;
- Carregar a imagem de fundo da cena real, utilizada como fundo da renderização;

4.1.4 Renderização

O aspecto diferencial do framework que estamos apresentando é forma em que a renderização da cena aumentada é realizada. A técnica apresentada por Debevec [17], consiste em renderizar por separado os objetos sintéticos e depois adicionar os mesmos à cena real fazendo uma combinação da renderização com a imagem da cena real através de uma técnica diferencial. Este procedimento resolve o problema em casos simples onde a interação entre os objetos sintéticos e reais não é muito complexa. Nosso algoritmo não requer de varias etapas de renderização e combinação de resultados parciais, basta exportar a cena modelada



(a) Modelagem em Blender. **(b)** Painel de opções de malhas.

Figura 4-4: A cena do *Blender* é exportada para o formato de cena do *ARLuxrender* utilizando o plugin *ARLuxblend*.

com *ARLuxblend* e o renderizador *ARLuxrender* encarrega-se de renderizar, sem necessidade de pós-processamento.

4.2 Renderização foto-realista de cenas de realidade mista com mapas de iluminação

Uma cena de realidade mista é composta essencialmente por dois tipos de objetos: os que existem na cena real capturada, e os elementos sintéticos que desejamos adicionar à cena real. Para obter uma renderização foto-realista de uma cena de realidade mista é necessário calcular as interações entre os objetos sintéticos e os objetos reais, tais como sombras e reflexões, e inclui-las na renderização aumentada da cena real. Assim, um renderizador para cenas de realidade mista deve ser capaz de utilizar toda a informação disponível para contabilizar as interações faltantes de maneira consistente com as que já são conhecidas através da cena real capturada.

No caso estudado neste capítulo, a iluminação considerada vem de um mapa de iluminação omnidirecional $\mathbb{M}$. O sistema de orientação local do mapa de iluminação nem sempre coincide com a orientação do mundo utilizada na modelagem. Assim uma direção ω, em coordenadas locais de $\mathbb{M}$ (i.e., no sistema de coordenadas da luz), precisa ser transformada para o sistema de coordenadas do mundo (i.e., da cena). Denominaremos de *LTW* (*Light to World*) e WTL (*World to Light*) as transformações lineares que fazem as respectivas mu-

danças entre os sistemas de coordenadas do mapa de iluminação e o sistema de coordenadas do mundo. Neste capítulo, os mapas de iluminação utilizados apenas contêm informação relativa à direção e radiância a partir do ponto de captura, portanto, as transformações LTW e WTL se reduzem a rotações.

O sistema de renderização foto-realista para cenas de realidade mista é desenvolvido sobre a base de alguns conceitos e estruturas que não formam parte de outros renderizadores. A seguir descrevemos as ideias necessárias para construir o sistema de renderização.

4.2.1 Primitivos sintéticos e primitivos de suporte

Em cenas de realidade mista temos dois tipos de objetos compondo o cenário. Por um lado temos os objetos sintéticos inseridos na cena e temos também os objetos originais da cena real. Quando existe uma interação visível entre dois objetos, é necessário modelar os objetos para poder sintetizar esta interação. Também é importante saber em todo momento com que tipo de objeto se está lidando, sintético ou real. Assim, classificamos os elementos da cena com dois tipos de primitivos:

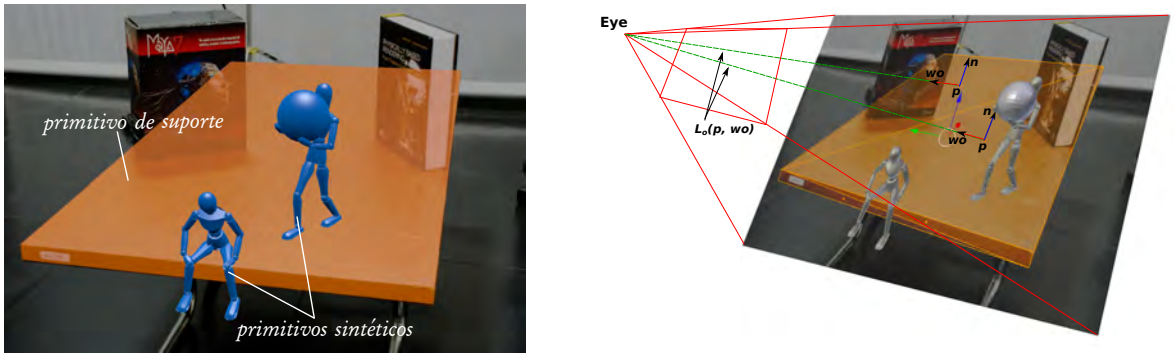
- ***Primitivos sintéticos:*** todos os objetos novos que são introduzidos na cena e não estão associados com nenhum objeto existente na cena original capturada são considerados como primitivos sintéticos. Os modelos de refletância para este tipo de objetos é conhecida, pois são definidos na modelagem. Assim, os cálculos de iluminação e interação com a cena para este tipo de objetos é a mesma utilizada para renderizar uma cena totalmente sintética com iluminação real, estudada no capítulo 3.

- ***Primitivos de suporte:*** são superfícies presentes na cena original capturada que apresentam alguma interação visível com os primitivos sintéticos adicionados, tais como sombras e reflexos. O cálculo de iluminação para estes primitivos não é trivial porque é preciso adicionar as contribuições provenientes dos objetos sintéticos mantendo as características da cena original nas áreas onde a interação é mínima ou nula. Para este tipo de primitivos será necessário estimar um modelo de refletância para realizar os cálculos de interação com os objetos sintéticos.

Quando criamos uma cena de realidade mista, não é necessário modelar todas as superfícies da cena real, senão somente aquelas que terão interação visível com os objetos sintéticos a serem inseridos pelo usuário. Estes elementos reais que interagem com os elementos sin-

téticos conformam a denominada cena local.

Nas implementações, utilizamos uma variável booleana *support* para especificar se um primitivo dado é de suporte (**true**) ou é um primitivo sintético (**false**).



(a) **(b)**

Figura 4-5: (a) Classificação dos primitivos da cena aumentada. (b) Os primitivos básicos para classificação são os triângulos que formam as malhas dos objetos. No exemplo o tampo da mesa é modelado com um plano que tem uma malha com dois triângulos.

4.2.2 Visibilidade e testes de oclusão

Para saber se uma determinada luz l_i deposita energia em um ponto p de uma superfície, é preciso saber se existe um raio contínuo desde o ponto p até a fonte de luz. Os raios obstruídos por algum objeto não depositam energia, determinando uma sombra no ponto p.

O teste de visibilidade convencional, *VisibilityTest(point p, dir* ω*)*, devolve um resultado **false** se o raio de sombra r intersecta algum objeto ao longo do raio com origem em p e direção ω da fonte de luz l_i (em um mapa de iluminação as amostras são luzes direcionais, logo l_i está no infinito), e **true** quando não há interseção. Um valor **true** indica que a luz associada ao raio de sombra r tem sua contribuição adicionada ao cálculo de iluminação do ponto p. Por outro lado um valor **false** indica que a luz associada ao raio r está sendo obstruída por algum objeto e portanto sua contribuição não é adicionada ao ponto p.

Nas cenas de realidade mista o teste de visibilidade convencional resulta insuficiente. Imagine um ponto p de uma superfície real de suporte (primitivo de suporte). Como todas as interações entre objetos reais estão contabilizadas na fotografia da cena real, não é necessário determinar sombras produzidas por outros objetos reais e sim as que são produzidas pelos objetos sintéticos da cena. Assim, estamos interessados em saber se o raio r que parte do ponto p intersecta algum objeto sintético antes de chegar à fonte de luz l_i. Por outra parte, para um ponto q sobre um objeto sintético, estamos interessados em determinar as possíveis

sombras produzidas por outros objetos sintéticos e também pela cena real. Neste caso, devemos determinar se o raio r que parte do ponto q intersecta algum objeto, sintético ou de suporte, antes de chegar à fonte de luz l_i.

Como se pode notar, os testes de visibilidade dependem do tipo de superfície para o qual desejamos realizar o cálculo de visibilidade. Se a origem p do raio de sombra pertence a um objeto sintético, o teste de visibilidade considera todos os objetos como intersectáveis e retorna um valor **false** quando intersecta algum objeto. Se o ponto p corresponde a um objeto real de suporte, o teste de visibilidade retornará **false** somente se a primeira interseção é com um objeto sintético. Para saber qual tipo de teste deve ser efetuado, modificamos o teste de visibilidade introduzindo uma variável booleana *null_shp_isect* que ajusta o resultado do teste. A tabela (4.1) indica os resultados obtidos com o teste de visibilidade segundo o valor da variável *null_shp_isect* e o tipo de objeto da primeira interseção.

Tabela 4.1: Teste de visibilidade para cenas de realidade mista.

VisibilityTest(point p, dir ω, bool null_shp_isect)		
Primeira interseção	***ignore_support***	
Tipo de objeto	**true**	**false**
Sintético	**false**	**false**
Suporte	**true**	**false**
Sem interseção	**true**	**true**

4.2.3 Cálculo de iluminação direta

As características visuais de um objeto da cena são definidas pelo material que o compõe. O material fornece uma descrição da aparência e das propriedades de interação com a luz em cada ponto da superfície. Esta descrição é dada pela *BSDF* (*bidirectional scattering distribution function*). Esta função nos diz quanta energia é refletida a partir de uma direção de entrada ω_i numa certa direção de saída ω_o, e também quanta energia é transmitida através da superfície. A quantidade refletida no ponto p é caracterizada pela *BRDF* (*bidirectional reflectance distribution function*), denotada como $f(p, \omega_o, \omega_i)$.

A equação de espalhamento (4.1), utilizada no cálculo da iluminação, diz que a radiância $L_o(p, \omega_o)$ que emana de um ponto p na direção ω_o é a soma da radiância $L_e(p, \omega_o)$ emitida pela superfície no ponto p mais a integral da radiância entrante sobre a esfera $\mathbb{S}^2$, $L_i(p, \omega_i)$, multiplicado pela *BSDF* f para cada direção ω_i e o cosseno do ângulo entre a normal n_p à

superfície em p e a direção ω_i na esfera $\mathbb{S}^2$.

$$L_o(p,\omega_o) = L_e(p,\omega_o) + \int_{\mathbb{S}^2} f(p,\omega_o,\omega_i) L_i(p,\omega_i) \left\langle \omega_i, n_p \right\rangle \mathrm{d}\omega_i, \tag{4.1}$$

A equação (4.1) também vale para a fotografia utilizada como fundo para construir a cena de realidade mista, pois a mesma pode ser considerada como o resultado de uma renderização, neste caso, feita pela própria câmera fotográfica.

Não vamos modelar objetos reais emissivos, logo consideramos $L_e(p,\omega_o) = 0$ para todo ponto p pertencente a uma superfície de suporte. Devido ao alto custo computacional para determinar uma solução da equação de espalhamento (4.1), a integral do lado direito é aproximada utilizando estimativas de Monte Carlo [50]. Portanto

$$L_o(p,\omega_o) = \frac{1}{N} \sum_{j=1}^{N} \frac{f(p,\omega_o,LTW(\omega_j)) \cdot \mathbb{M}(\omega_j) \cdot \max\{0, \left\langle LTW(\omega_j), n_p \right\rangle\}}{p(\omega_j)}, \tag{4.2}$$

onde $\left\langle LTW(\omega_j), n_p \right\rangle > 0$ quando $LTW(\omega_j)$ está no hemisfério visível de p, e $p(\omega_j)$ é a probabilidade de amostrar a direção ω_j no mapa de iluminação $\mathbb{M}$. O valor da probabilidade $p(\omega_j)$ é dado por

$$p(\omega_j) = \frac{Luminancia(\mathbb{M}(\omega_j))}{\int_{\mathbb{S}^2} Luminancia(\mathbb{M}(\omega))\mathrm{d}\omega}.$$

A estimativa equação (4.2) é utilizada quando p pertence a uma superfície sintética. Para pontos sobre superfícies de suporte, a equação (4.2) será calculada através de uma estimativa diferente. Fazemos isto porque a radiância de um ponto p pertencente a uma superfície real já é conhecida na imagem capturada como fundo. Somente estamos interessados em levar em consideração a probabilidade de amostragem da iluminação para estimar as sombras (i.e., a probabilidade dos raios de sombra ser bloqueados). Seguindo esta ideia formulamos a seguinte estimativa para uma superfície real de suporte:

$$L_o(p,\omega_o) = \frac{1}{N} \sum_{j=1}^{N} Background(p,\omega_o) \cdot LightScale(n_p) \cdot \max\{0, \left\langle LTW(\omega_j), n_p \right\rangle\}, \tag{4.3}$$

onde

$$LightScale(n_p) = \frac{\int_{\mathbb{S}^2} Luminancia(\mathbb{M}(\omega_s))\mathrm{d}\omega_s}{\int_{\mathbb{S}^2} Luminancia(\mathbb{M}(\omega_j))\max\{0, \langle LTW(\omega_j), n_p \rangle\}\mathrm{d}\omega_j}. \tag{4.4}$$

Nas estimativas (4.3), as amostras ω_j são selecionadas de acordo à probabilidade $p(\omega_j)$ definida para as estimativas de Monte Carlo usadas com primitivos sintéticos. Desta forma, durante o processo de amostragem do mapa de iluminação as amostras com maior radiância serão selecionadas com maior frequência, e no caso de seus raios serem obstruídos por algum objeto sintético, determinarão uma sombra no ponto p com uma intensidade proporcional à probabilidade de amostragem.

Qualquer objeto da cena é modelado por uma malha de triângulos, logo os primitivos geométricos de superfícies são triângulos. Como é inviável calcular o valor *LightScale* para cada raio r que intercepta um ponto p de uma superfície real de suporte, optamos por um pré-processamento. Calculamos o valor $LightScale(n_p)$ para cada triângulo da cena que pertence a uma superfície real de suporte, utilizando a normal, n_p, do triângulo. Cada valor *LightScale* calculado é guardado numa variável no seu primitivo correspondente. Desta forma, durante a renderização para calcular um valor $BackgroundColor(p, \omega_o)$ basta acessar a fotografia de fundo da cena, carregada como textura, para obter $L_o(p, \omega_o)$ e procurar no primitivo do ponto p pelo valor *LightScale* que foi calculado no pré-processamento inicial.

4.2.4 Pré-processamento da iluminação para superfícies reais

Como o mapa de iluminação é representado por uma imagem podemos computar a integral da equação (4.4) somando sobre os pixels da imagem do mapa. Esta abordagem é demorada, especialmente quando a quantidade de primitivos de suporte é grande, portanto implementamos uma alternativa que transforma o mapa num conjunto de luzes direcionais cuja radiância total é equivalente à radiância total do mapa. Assim em lugar de somar sobre o mapa, aplicamos a soma integral sobre o grupo de luzes direcionais para ter uma estimativa rápida da iluminação. O mapa é discretizado aplicando o método de corte mediano descrito por Debevec [18]. Também é possível aplicar outros métodos como por exemplo o de Ostromoukhov [47], estudado por Zang em [61]. Normalmente discretizamos um mapa utilizando $1024 = 2^{10}$ luzes, o que permite obter um incremento de desempenho de $2^9 = 512$ vezes para mapas de 1024×512 pixels. A equação (4.4) calculada sobre o conjunto $L_d(i)$ de K luzes

direcionais que aproxima o mapa de iluminação é:

$$LightScale(n) = \frac{\sum_{i=1}^{K} L_d(i)}{\sum_{i=1}^{K} L_d(i) \cdot \max\{0, \langle LTW(\omega_i), n\rangle\}}. \tag{4.5}$$

4.3 Ray tracing para cenas de realidade mista

Para compreender o processo de *ray tracing* que vamos propor para cenas de realidade mista é necessário entender como funciona a *equação de transporte de luz* (*LTE*), formulada por Kajiya [34], que descreve os fenômenos de interação global entre as fontes luminosas e as superfícies da cena.

É possível descrever a partir da *LTE* a radiância incidente no ponto p_0 desde o ponto p_1, onde p_1 é o primeiro ponto sobre uma superfície ao longo do raio partindo de p_0 na direção $p_1 - p_0$:

$$\begin{aligned} L(p_1 \to p_0) = L_e(p_1 \to p_0) &+ \int_A L_e(p_2 \to p_1) f(p_2 \to p_1 \to p_0) G(p_2 \leftrightarrow p_1) \mathrm{d}A(p_2) \\ &+ \int_A \int_A L_e(p_3 \to p_2) f(p_3 \to p_2 \to p_1) G(p_3 \leftrightarrow p_2) \\ &\quad f(p_2 \to p_1 \to p_0) G(p_2 \leftrightarrow p_1) \mathrm{d}A(p_3) \mathrm{d}A(p_2) + \cdots \end{aligned}$$

Cada termo do lado direito da equação representa um caminho de comprimento crescente. Por exemplo, o terceiro termo é ilustrado na figura (4-6). Este caminho tem 4 vértices, conectados por três segmentos. A contribuição total de todos os caminhos de longitude

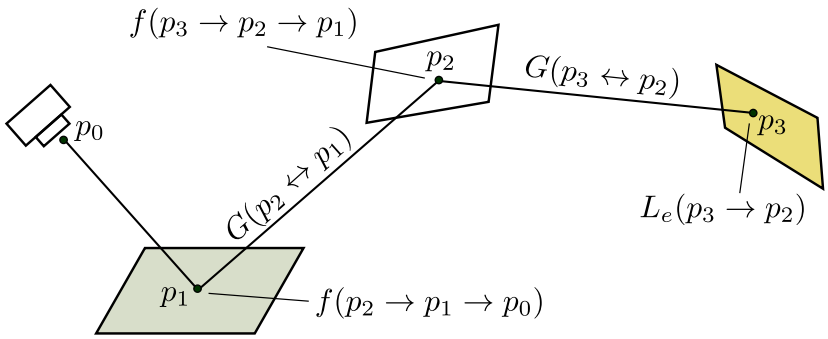


Figura 4-6: A integral de todos os pontos p_2 e p_3 em superfícies da cena da a contribuição total de radiância, partindo de p_1 na direção de p_0, dada por dois segmentos de caminho. As componentes do integrando são: a radiância emitida luz L_e, os termos geométricos entre os vértices G, e o espalhamento das *BSDFs*, f.

4 é dada pelo terceiro termo. O somatório infinito da equação pode ser escrito da seguinte maneira:

$$L(p_1 \to p_0) = \sum_{i=1}^{\infty} P(\bar{p}_i), \tag{4.6}$$

onde $P(\bar{p}_i)$ é a radiância espalhada ao longo de todos os caminhos $\bar{p}_i$ com $i+1$ vértices, $\bar{p}_i = (p_0, p_1, \cdots, p_i)$ com p_0 posicionado na câmera e p_i na fonte de luz. Assim

$$P(\bar{p}_i) = \underbrace{\int_A \int_A \cdots \int_A}_{i-1} L_e(p_i \to p_{i-1}) T(\bar{p}_i) \mathrm{d}A(p_2) \cdots \mathrm{d}A(p_i),$$

onde

$$T(\bar{p}_i) = \prod_{j=1}^{i-1} f(p_{j+1} \to p_j \to p_{j-1}) G(p_{j+1} \leftrightarrow p_j)$$

descreve a porção de radiância da fonte de luz que chega até a câmera depois de sofrer difusão em todos os vértices ao longo do caminho.

Dada a equação (4.6) e um comprimento fixo i, para calcular uma estimativa de Monte Carlo da radiância que chega ao ponto p_0 através dos caminhos de comprimento i, é preciso amostrar um conjunto de vértices com uma densidade de amostragem apropriada, e estimar $P(\bar{p}_i)$ usando os vértices amostrados. Uma descrição detalhada da *LTE* e a derivação do algoritmo de *path tracing* para simulação foto-realista pode ser encontrado no livro *"Physically Based Rendering: from theory to implementation"* de M. Pharr e G. Humphreys [48, 50].

4.3.1 *ARpath*: path tracing para cenas de realidade mista

O algoritmo de *path tracing* calcula uma estimativa do somatório da equação (4.6), construindo para cada raio que parte da câmera, um caminho contínuo linear por partes onde o primeiro vértice, p_0, encontra-se na câmera, o último vértice, p_i, está sobre uma fonte de luz e os vértices intermediários, p_k, encontram-se sobre os objetos da cena.

A implementação do algoritmo de *path tracing* que utilizamos resolve a equação de iluminação realizando a construção incremental do caminho, começando pelo vértice p_0 na câmera. Em cada vértice, a *BSDF* é amostrada para gerar uma nova direção. O próximo vértice p_{k+1} é encontrado traçando um raio desde p_k na direção amostrada e encontrando a primeira intersecção com algum objeto da cena.

Como temos 2 tipos de primitivos, sintéticos e de suporte, um caminho $\bar{p}_i = (p_0, \cdots, p_i)$ será classificado segundo seus vértices como:

- **Caminho real:** os vértices p_k, $k = 1, \cdots, i-1$ estão sobre superfícies reais de suporte.
- **Caminho sintético:** os vértices p_k, $k = 1, \cdots, i-1$ estão sobre a superfície de objetos sintéticos.
- **Caminho misto:** alguns vértices são sintéticos e outros reais.

Desta forma, a *equação de transporte* pode ser particionada da seguinte forma:

$$L(p_1 \to p_0) = \sum_{i=1}^{\infty} P_r(\bar{p}_i) + P_s(\bar{p}_i) + P_m(\bar{p}_i), \tag{4.7}$$

onde $P_r(\bar{p}_i)$ representa os caminhos reais, $P_s(\bar{p}_i)$ os caminhos sintéticos, e $P_m(\bar{p}_i)$ os caminhos mistos.

Para cada vértice p_k com $k = 1, \cdots, i-1$ devemos calcular a radiância espalhada no ponto p_k na direção $p_{k-1} - p_k$. Este cálculo é efetuado através de estimativas de Monte Carlo: usamos a estimativa tradicional (4.2) se p_k pertence a uma superfície sintética e a estimativa dada pela equação (4.3) se p_k pertence a uma superfície real de suporte.

Caminhos reais não precisam ser calculados, pois já foram computados na fotografia usada como fundo de cena. Entretanto, como a cena local foi modelada, devemos renderizá-la novamente, e sua radiância deve coincidir com a fotografia original. Para fazer isto agrupamos toda a informação de radiância de um caminho real no vértice p_1 na forma de iluminação direta, com a finalidade de evitar cálculos nos vértices seguintes.

Tanto os caminhos sintéticos quanto os mistos são calculados de maneira similar, contabilizando a contribuição da iluminação direta em cada vértice do caminho. A diferença está na estimativa de Monte Carlo utilizada em cada caso. A figura (4-7) ilustra os tipos de

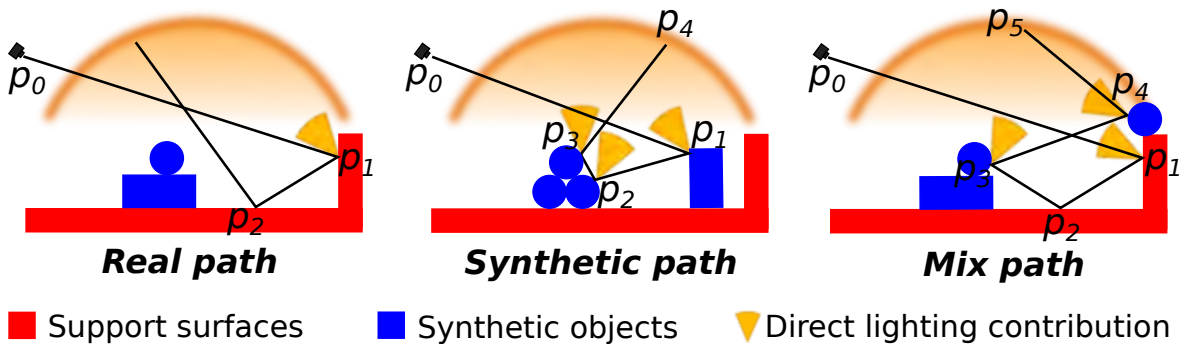


Figura 4-7: Tipos de caminhos e cálculo da iluminação. Os cones amarelos significam que a iluminação direta foi calculada no vértice p_i correspondente.

caminhos e o cálculo que é realizado nos vértices em cada caso.

Na prática somente é preciso saber se num momento dado da construção os vértices do caminho são todos reais de suporte ou se existe pelo menos um vértice que pertence a uma superfície sintética. Utilizamos então uma variável booleana *path_type* para especificar se o caminho que está sendo construído contém um vértice sintético ou não. O valor da variável é atualizado a cada novo vértice que é adicionado ao caminho. Os algoritmos 1 e 2 ilustram a implementação do *path tracing* para cenas de realidade mista.

4.4 Resultados e considerações

Testamos o framework proposto na produção de cenas estáticas e também em animações com fundo e iluminação estáticos.

Não realizamos comparações de tempo de processamento pois as mudanças no algoritmo de path tracing não incrementaram o tempo de renderização para cenas aumentadas, quando comparado com os algoritmos de path tracing tradicionais. Na cena da figura (4-10) as superfícies reais de suporte são modeladas com 45000 triângulos e o tempo de pré-processamento foi de 3s. num tempo total de renderização de 1000s.

Na figura (4-9) apresentamos uma comparação entre cenas de realidade mista e cenas reais semelhantes. É possível observar que as reflexões no livro e na esfera são coerentes com a cena real fotografada. Na figura (4-11d) podemos observar como parte da cena real é distorcida pelos copos sintéticos devido ao efeito de refração da luz. Também é possível observar os reflexos produzidos pelos copos sintéticos sobre a capa do livro que está apoiado na mesa. Para capturar estes reflexos foi necessário modelar o livro, tal como pode ser observado na figura (4-4). Exemplos adicionais que ilustram a qualidade dos efeitos logrados são apresentados na figura (4-12).

O framework proposto atende aos propósitos de fotorrealismo considerados e gera resultados com alto grau de realismo. Como é de se esperar existem cenas problemáticas onde o realismo fica comprometido. Estes problemas estão relacionados ao fato de estarmos reconstruindo uma cena 3D a partir de uma representação fotográfica 2D na qual não temos informação sobre a profundidade. Este tipo de problemas pode ser visto quando na cena real existem inter-reflexões visíveis e queremos adicionar um objeto sintético que bloqueia alguma de estas inter-reflexões. Estes problemas podem ser evitados se a cena for estudada

e composta adequadamente. O inconveniente é que exige uma modelagem mais delicada e alguns conhecimentos adicionais por parte do artista.

Outro dos inconvenientes que notamos é que, quando trabalhamos com objetos sintéticos com brilho elevado surgem problemas enquanto a uso de mapas de iluminação.

Algorithm 1 ARPath Integrator

$L \leftarrow 0$; ▷ Radiance contribution
$r_0 \leftarrow ray(p_0, -\omega_0)$; ▷ Camera ray
$path\_type \leftarrow false$; ▷ Path type
$T \leftarrow 1$; ▷ Path throughput factor
for $i \leftarrow 0, MaxLength$ **do**
    $p_{i+1} \leftarrow SceneIntersect(r_i)$;
    $T \leftarrow T \cdot ThroughputAtenuation(p_i, p_{i+1})$;
    if $p_{i+1} = INFINITY$ **then** ▷ r_i not instersect the scene
        if $i = 0$ **then**
            $L \leftarrow$ compute background color from photograph;
        else
            if $path\_type = true$ **then**
                $\omega \leftarrow WorldToLight(p_{i+1} - p_i)$;
                $L \leftarrow L + T \cdot \mathbb{M}(\omega)$;
            end if
        end if
        break;
    else
        if $type(p_{i+1}) \neq SUPPORT$ **then**
            $path\_type \leftarrow true$;
        end if
        if $i = MaxLength$ **then** ▷ Terminate the path
            break;
        end if
        $n_p \leftarrow$ normal at p_{i+1};
        $L \leftarrow L + T \cdot DirectLighting(p_{i+1}, n_p, (p_i - p_{i+1}), path\_type, i)$;
        $r_{i+1} \leftarrow SampleNewDirection(p_{i+1})$; ▷ Sample BSDF to get new path direction
        $T \leftarrow T \cdot f(p_{i+1}, -dir(r_i), dir(r_{i+1}))$;
        if $Prob() < END\_PROBABILITY$ **then** ▷ Continues the path construction?
            break; ▷ Terminate the path
        else
            $T \leftarrow T / Prob()$; ▷ Increase path contribution
        end if
    end if
end for
return L;

Algorithm 2 Direct Lighting Estimator

Procedure DirectLighting(p, n_p, ω_o, $path\_type$, $RayDepth$)
$Ld \leftarrow 0$;
$mis \leftarrow UseMultipleImportanceSampling(p)$;
$\omega_i \leftarrow SampleLightDir(\mathbb{M})$;
$\omega_p \leftarrow LightToWorld(\omega_i)$;
if $type(p) = SUPPORT$ **then**
    if ($path\_type = true$) **OR** ($RayDepth = 0$) **then**
        if ($mis = false$) **OR** ($RayDepth = 0$) **then**
            if $VisibilityTest(p, \omega_p, true) = true$ **then**
                $Ld += Background(p, \omega_o) \cdot LightScale(p) \cdot \max\{0, \langle \omega_p, n_p \rangle\}$;
            end if
        else ▷ ($mis = true$) **AND** ($RayDepth > 0$)
            if $VisibilityTest(p, \omega_i, false) = true$ **then**
                $Ld += Background(p, \omega_o) \cdot LightScale(p) \cdot \max\{0, \langle \omega_p, n_p \rangle\}$;
                $Ld += \frac{f(p, \omega_o, \omega_p) \mathbb{M}(\omega_i) \langle \omega_p, n_p \rangle}{pdf(\omega_i)}$;
            end if
        end if
        if $mis = true$ **then** ▷ Trace a 2nd shadow ray by sampling the BSDF
            $\omega_i \leftarrow SampleBSDFDir(p, \omega_o)$;
            if $VisibilityTest(p, \omega_i, false) = true$ **AND** $RayDepth > 0$ **then**
                $\omega \leftarrow WorldToMap(\omega_i)$;
                $Ld \leftarrow Ld + \mathbb{M}(\omega)$;
            end if
        end if
    end if
else ▷ $type(p) = SYNTHETIC$
    if $VisibilityTest(p, \omega_i, false) = true$ **then**
        $Ld \leftarrow Ld + \frac{f(p, \omega_o, \omega_p) \mathbb{M}(\omega_i) \langle \omega_p, n_p \rangle}{pdf(\omega_i)}$
    end if
    if $mis = true$ **then** ▷ Trace a 2nd shadow ray by sampling the BSDF
        $\omega_i \leftarrow SampleBSDFDir(p, \omega_o)$;
        if $VisibilityTest(p, \omega_i, false) = true$ **then**
            $\omega \leftarrow WorldToMap(\omega_i)$;
            $Ld \leftarrow Ld + \mathbb{M}(\omega)$;
        end if
    end if
end if
return Ld
end procedure

Figura 4-8: Cenas de realidade mista. Os exemplares do modelo *killeroo* e as esferas são sintéticos.

(a) Cena real. **(b)** Cena aumentada. **(c)** Cena com esfera real. **(d)** Reflexos.

Figura 4-9: Comparação entre cenas reais e cenas de realidade mista. (b) A esfera foi adicionada com renderização. (d) *Inferior*: a renderização do reflexo da fig. (b); *Superior*: fotografia do reflexo da esfera real (c).

(a) Modelagem. **(b)** Fotografia da cena real. **(c)** Renderização da cena aumentada.

Figura 4-10: Exemplo de cena de realidade mista com superfícies reais de suporte curvas. A cabeça foi modelada na cena aumentada para incluir as sombras projetadas por duas esferas sintéticas de metal adicionadas a cena.

(a) Imagem da cena real.

(b) Renderização aumentada.

(c) Detalhes da figura (b).

(d) Detalhes da figura (b).

Figura 4-11: (b) Renderização aumentada. As esferas, os copos e os dois parafusos foram introduzidos na cena. (c) Esfera artificial e sua interação com a cena real.

(a)

(b)

Figura 4-12: Exemplos de cenas de realidade mista. O fundo real é o mesmo para as imagens (a) e (b). É possível apreciar efeitos de oclusão de objetos (boneco e esfera em (a)), e reflexões indiretas sobre os entre as esferas e livros em (a). Em (b) é possível observar o reflexo parcial do boneco na janela ao fundo.

Capítulo 5

Mapas de iluminação *RGB-D* e usos em renderizações de realidade mista

No capítulo 4, estudamos o problema de renderização foto-realista de objetos sintéticos imersos num cenário real. No presente capítulo, continuaremos estudando o mesmo problema desde uma ótica diferente. O problema a ser tratado é: *"Renderizar objetos sintéticos em cenas capturadas e armazenadas em imagens omnidirecionais HDR de maneira foto-realista"*. Ou seja, temos uma imagem omnidirecional *HDR* que representa uma cena do mundo real e desejamos introduzir novos elementos sintéticos nesta cena, usando a própria imagem omnidirecional tanto como fundo de cena quanto como mapa de iluminação para renderizar os objetos sintéticos. Lembre-se que no caso analisado no capítulo 4 trabalhamos com um mapa de iluminação capturado próximo da posição de inserção dos objetos sintéticos, e o fundo da cena era capturado de forma separada desde uma posição de câmera diferente à do mapa de iluminação.

Para resolver este problema formulamos um novo tipo de mapa de iluminação que contém além da radiância informação geométrica da cena. Assim entre as contribuições deste capítulo temos uma nova estrutura para mapas de iluminação, o mapa de iluminação *RGB-D* e um framework para construir um mapa desde tipo a partir de um mapa de iluminação convencional. Outra contribuição é um algoritmo de renderização que explora a informação de um mapa de iluminação *RGB-D* para resolver o problema de renderização de objetos sintéticos de maneira foto-realista dentro do próprio mapa de iluminação.

Os mapas de iluminação *RGB-D* (i.e., panoramas omnidirecionais *RGB-D* em *HDR*) também podem ser usados em renderizações com fundo capturado de maneira separada, como

o problema estudado no capítulo 4, com o benefício adicional de poder introduzir vários objetos sintéticos em posições arbitrarias da cena e conseguir renderizações de qualidade.

5.1 Mapas de iluminação com profundidade

Em este capítulo vamos explorar um novo tipo de representação para mapas de iluminação, o mapa de iluminação com profundidade, ou mapa de iluminação *RGB-D*, proposto em [23]. Este tipo de mapa contém tanto a radiância quanto informação de distribuição espacial (i.e., profundidade) da iluminação. Na abordagem tradicional, um mapa de iluminação pode ser considerado como um conjunto de luzes direcionais, de tal forma que cada pixel do mapa representa uma luz direcional.

Em este novo tipo de mapa que estamos propondo, o mesmo fornece informações sobre a estrutura geométrica do ambiente, por isso, podemos considerá-lo como um conjunto de luzes pontuais em vez de luzes direcionais. Como veremos ao longo do capítulo, esta abordagem posicional faz do mapa de iluminação uma ferramenta poderosa para propósitos de renderização. Isto é devido a que, conhecendo a profundidade podemos reconstruir a posição de onde provem a luz e realizar cálculos de sombras e reflexões mais complexas e precisas para qualquer objeto da cena, sem necessidade do mesmo estar próximo à origem de captura do mapa *RGB-D* como acontece com os mapas de iluminação tradicionais.

Um mapa de iluminação *RGB-D* (mapa de iluminação com profundidade) pode ser construído a partir de um mapa de iluminação *HRD* tradicional, adicionando um canal com a informação de profundidade, como ilustramos na figura (5-1). O canal de profundidade pode ser obtido a partir de uma renderização da geometria reconstruída do ambiente (renderizando a profundidade em lugar de cor) ou através de equipamentos de captura especiais (e.g. Scanners 3D).

Na primeira parte deste capitulo discutiremos como obter um mapa de iluminação *RGB-D* a partir de um mapa de iluminação tradicional através de um processo de modelagem. Para tais fins construímos um framework de produção que foi testado com panoramas que capturamos e também usamos o framework com panoramas disponíveis na internet para validar a ferramenta.

Para organizar o resto do texto, definiremos a notação a ser utilizada neste capítulo para no referir a um mapa de iluminação *RGB-D*. Uma amostra do mapa de iluminação *RGB-D*

(a) Canal de radiância. **(b)** Canal de profundidade.

Figura 5-1: Mapa de iluminação com profundidade. (b) O canal de profundidade é utilizado para reconstruir a posição das luzes.

será denotada como $\mathbb{M}(\omega_i, z_i)$, onde ω_i é a direção da amostra de luz no mapa (a conversão entre direções e posições de pixels no mapa é feito segundo as equações da parametrização escolhida para o mapa, ver apêndice A). O valor escalar z_i denota a distância da amostra de luz desde a origem do panorama. Assim, a posição da amostra de luz no espaço local do mapa é dada pelo ponto $z_i \cdot \omega_i$. Para efeitos de alguns cálculos, por exemplo escolher amostras para realizar uma estimativa de Monte Carlo, não será necessário ter em conta a posição 3D da amostra e somente importará a posição da amostra no mapa (i.e., ω_i). Neste caso usaremos a notação $\mathbb{M}(\omega_i)$ adotada para os mapas tradicionais.

5.2 O framework

O framework que apresentamos neste capitulo tem seu foco em produções panorâmicas foto-realistas que exigem a combinação de elementos sintéticos e capturados. Parte do framework é semelhante ao apresentado no capítulo 4, porém o fato de usar um mapa de iluminação com informação de profundidade exige alguns processos adicionais. De modo geral, o framework é composto das seguintes partes:

1. Captura do ambiente e construção do panorama
2. Calibração do Panorama
3. Construção da malha do ambiente
4. Produção do roteiro e filme
5. Renderização final

Alguns dos passos apresentados podem ser realizados de varias maneiras, como já foi comentado no capítulo 4. Como princípio geral para esta tese, escolhemos implementar o framework usando apenas a tecnologia e equipamentos que podem ser acessíveis para a maioria dos artistas. Entretanto, devemos enfatizar, que existem equipamentos e tecnologia que podem facilitar em grande medida a execução de algumas partes deste framework. Há equipamento profissional que permite realizar a etapa de captura, reconstruindo e calibrando o ambiente em um único passo. Por exemplo, a malha que modela o ambiente pode ser obtida diretamente por meio de scanners 3D. Soluções semiautomáticas mais acessíveis, potencialmente mais interessantes para o artista independente, também podem ser alcançadas com ferramentas de uso diário, como Kinect, sobre o qual existem trabalhos como o *KinectFusion* [46]; ou ainda com uma única câmera em movimento, como descrito no [45].

5.3 Captura do ambiente real e construção do panorama

Existem diversos métodos e equipamentos especiais que permitem capturar e montar uma imagem *HDR* omnidirecional. Por exemplo, é possível usar uma câmera panorâmica *HDR*, como a *Ladybug* da *Point Grey*, capaz de capturar o panorama *HDR* através de um único disparo. Como o framework é flexível, o usuário pode escolher a melhor técnica segundo as necessidades específicas de uma produção. Escolhemos usar uma câmera semiprofissional com lente olho de peixe e um software *open source* para montagem de panoramas. As etapas básicas para produzir um panorama *HDR* com este método são:

- Colocar a câmera em uma posição estratégica no ambiente real;
- Capturar sequencias de fotografias com diferentes tempos de exposição, e alinhamentos para preencher todo o campo visual de $360^\circ \times 180^\circ$;
- Montar imagens *HDR* com a sequencia de imagens obtidas para cada vista parcial;
- Montar o panorama omnidirecional com todas as vistas parciais *HDR*;

As câmeras fotográficas podem capturar e armazenar radiância do ambiente, porém dificilmente podem armazenar todos os valores de radiância existentes em uma cena real. Felizmente, existem trabalhos como o do Debevec e Malik [19], que permitem recuperar o espectro de radiância de uma cena usando uma câmera fotográfica como equipamento de

captura. Entretanto, o método de Debevec e Malik envolve ajustes de calibração da câmera e exige a captura de uma sequencia de fotografias com diferentes exposições da cena para alimentar o algoritmo.

Adotamos o método de Debevec e Malik para produzir vistas *HDR* parciais a partir de uma sequencia de fotos com diferentes tempos de exposição. Usamos uma sequencia de 9 fotos por cada orientação da câmara, com um total de 7 orientações diferentes para obter o campo de visão completo de $360^\circ \times 180^\circ$ desde o ponto de vista da câmera. As 7 vistas *HDR* parciais obtidas são usadas para montar um panorama *HDR* equiretangular. Para testar as nossas ideias usamos uma câmera Nikon DX2S com uma lente olho de peixe Nikon DX10.5mm, e uma cabeça *Manfrotto PanoHead* (figura (5-2)). Para a montagem das imagens *HDR* das vistas parciais foi usado o software *Luminance HDR* e, para montar o panorama usamos o software *Hugin*, ambos os projetos de software livre disponíveis na Internet.

Baseamos a nossa técnica de captura do panorama no estudo apresentado por Kuliyev [38]. Os grandes estúdios de Hollywood costumam usar soluções mais robustas e custosas para realizar a captura da iluminação e geometria do ambiente (por exemplo, nuvens de pontos). Nossa proposta é apresentar soluções acessíveis a um público mais amplo.

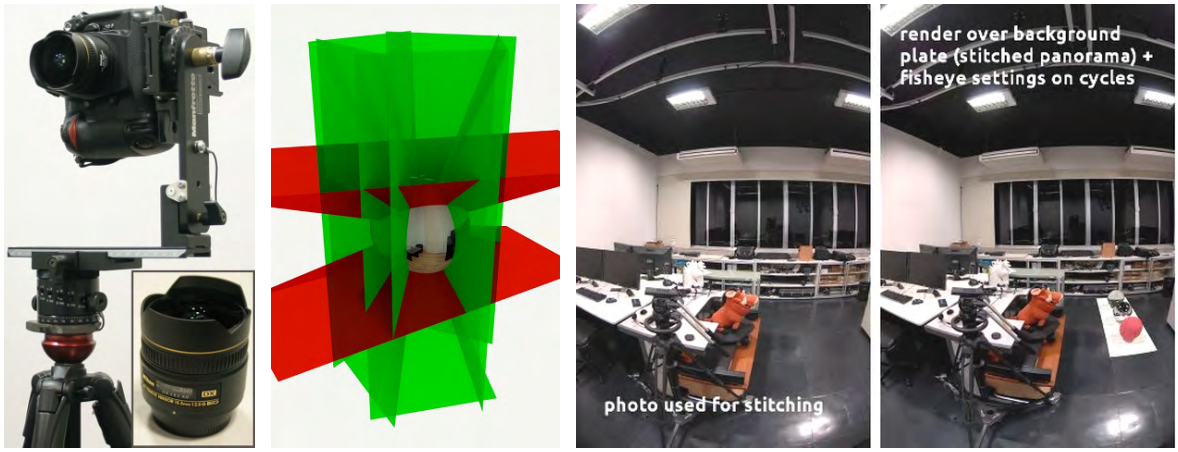
5.3.1 Captura de fotografias

Usamos uma lente olho de peixe (Nikon 10.5 mm DX) para aumentar o campo de visão por cada fotografia tomada e reduzir o número de fotografias necessárias para cobrir o campo de visão completo. Usamos 7 fotografias para cobrir todo o campo de visão, organizadas de acordo a ilustração da figura (5-2b).

5.3.2 Captura do Polo Sul - *Nadir*

O polo sul é conhecido pelos problemas que apresenta quando se trata de montar um panorama. Muitos artistas consideram a captura do *nadir* opcional, uma vez que o tripé obstrui parte do ambiente nesta direção. Isso pode ser resolvido, colocando de maneira estratégica elementos sintéticos durante a reconstrução modelagem do ambiente (seção 5.5.3).

Para produzir uma captura completa do ambiente, podemos tirar a fotografia do correspondente ao polo sul sem o tripé segurando a câmera com as mãos. Isto pode introduzir



(a) Equipamento usado para capturar o ambiente completo.

(b) Partição do espaço utilizado para montar o panorama.

(c) Esq: Fotografia real da cena, **Dir:** Imagem renderizada com *Cycles* utilizando o panorama esférico com uma lente olho de peixe virtual.

Figura 5-2: (a) Sistema de captura: *Manfrotto Pano Head*, *Nikon DX2S* e lente *Nikon 10.5mm*. (b) Para a lente *Nikon* 10.5mm: 5 fotos em torno do eixo z, 1 foto do Zenith e 1 foto do nadir. (c) Comparação de uma imagem tomada a partir da câmera real e uma renderização do panorama simulando a uma câmera virtual com a mesma lente.

anomalias e imprecisões durante a montagem do panorama. Para minimizar esse problema, criamos uma máscara que aplicamos à imagem correspondente ao polo sul, para utilizar somente a informação que não foi coletada pelas outras vistas capturadas com o tripé. Para determinar de maneira precisa a máscara, foi montado um panorama sem o polo sul, depois foi criada uma cena virtual onde o panorama é usado como fundo. Renderizamos uma vista parcial desta cena com uma câmera virtual que emula a câmera real que aponta na mesma direção que a câmera usada para capturar o polo sul. A área preta na renderização representa a informação que está faltando e tem que ser extraída da fotografia do polo sul. Tivemos que criar uma lente olho de peixe virtual para emular a lente 10.5mm usada. A distorção da lente olho de peixe pode ser calculada com a equação (5.1):

$$FOV_{equisolid} = 4 \cdot \arcsin \cdot \left(\frac{framesize}{focallength \cdot 4}\right) \tag{5.1}$$

Na figura (5-2c) mostramos uma comparação de uma imagem tomada a partir da câmera real e uma renderização do panorama simulando uma lente olho de peixe. Esta implementação foi feita em *Cycles*, um renderizador de traçado de raios para *Blender*.

5.4 Calibração do panorama

Um panorama equiretangular é uma representação da esfera $\mathbb{S}^2$ no plano. Uma imagem equiretangular induz um referencial em $\mathbb{S}^2$. As partes superior e inferior da imagem equiretangular representam os polos da esfera $\mathbb{S}^2$. No entanto, a imagem equiretangular pode não estar alinhada com as direções do mundo real. Em outras palavras, o polo norte da esfera $\mathbb{S}^2$ associada à imagem equiretangular pode não coincidir com o zênite do mundo real.

Uma segunda questão é a escala do espaço paramétrico. O espaço representado no panorama equiretangular está normalizado em torno do ponto de vista da câmera. A fim de reconstruir o espaço tridimensional, é necessário para ancorar um ponto no espaço representado, onde a distância é conhecida.

Vantagens adicionais do sistema de calibração:

1. Permite mover áreas importantes da cena e afasta-las dos polos.
2. Não é necessário fazer um alinhamento perfeito do tripé durante a captura
3. Funciona com panoramas publicados na Internet.
4. Permite alinhar o panorama com a orientação no mundo, facilitando a reconstrução do ambiente.

5.4.1 Alinhamento do horizonte

Para determinar o alinhamento horizontal do panorama, vamos localizar o horizonte através de entradas por parte do usuário, que pode marcar elementos conhecidos. Sobre a imagem equiretangular do panorama, o usuário deve selecionar 4 pontos p_0, p_1, p_2 e p_3 que representam os vértices de uma forma retangular no espaço do mundo e que estão sobre o piso, i.e., sobre um plano perpendicular aos polos (ver figura (5-3)).

Os pontos selecionados definem vetores cujos produtos vetoriais determinam os eixos **x** e **y** do plano horizontal. O produto vetorial de **x** e **y** determina o eixo **z** (onde se encontram os polos norte e sul). Isso deve ser suficiente para determinar a orientação global da imagem, porém a entrada manual de dados introduz erros no cálculo dos eixos **x** e **y**. Em vez de usar os dados introduzidos, calculamos novamente o eixo **y** como o produto vetorial dos eixos **z**

e **x**, para obter uma base ortonormal.

$$\begin{aligned} x &= (\vec{p}_0 \times \vec{p}_1) \times (\vec{p}_3 \times \vec{p}_2) \\ y &= (\vec{p}_1 \times \vec{p}_2) \times (\vec{p}_0 \times \vec{p}_3) \end{aligned} \quad \Bigg| \quad \begin{aligned} z &= x \times y \\ y &= z \times x \end{aligned} \tag{5.2}$$

Para garantir que o resultado é satisfatório reprojetamos a linha do horizonte e dos eixos na imagem para fornecer um feedback visual para o usuário, permitindo um ajuste fino do retângulo de calibração. O retângulo escolhido para a calibração define os eixos do mundo e o alinhamento do plano do piso. Isso facilita o trabalho nas fases de reconstrução do ambiente existente e na modelagem 3D de novos elementos.

(a) **(b)**

Figura 5-3: (a) Sistema de alinhamento do panorama. Os 4 pontos no piso determinam os eixos de orientação para o ambiente. As linhas azul, vermelha e verde ilustram o corte dos planos *xy* (horizonte), *xz* e *yz* respectivamente. (b) Representação esférica do panorama (a).

5.4.2 Escala do mundo

Uma vez que a orientação do mapa foi calculada podemos projetar o retângulo selecionado no piso do mundo 3D. Precisamos, no entanto, de mais dados - a orientação por si só não é suficiente. Todo valor positivo para a altura da câmera irá produzir uma escala de reconstrução, que pode não coincidir com a escala do ambiente real.

Temos um sistema dual com as dimensões do retângulo e a altura da câmera. Assim, deixamos para o usuário decidir qual deles usar segundo os dados disponíveis - a altura da câmera ou as dimensões do retângulo. As dimensões do retângulo não são utilizadas nas operações de reconstrução posteriores. Em vez disso, sempre calculamos a altura da câmera para o parâmetro de entrada fornecido.

5.4.3 Considerações adicionais

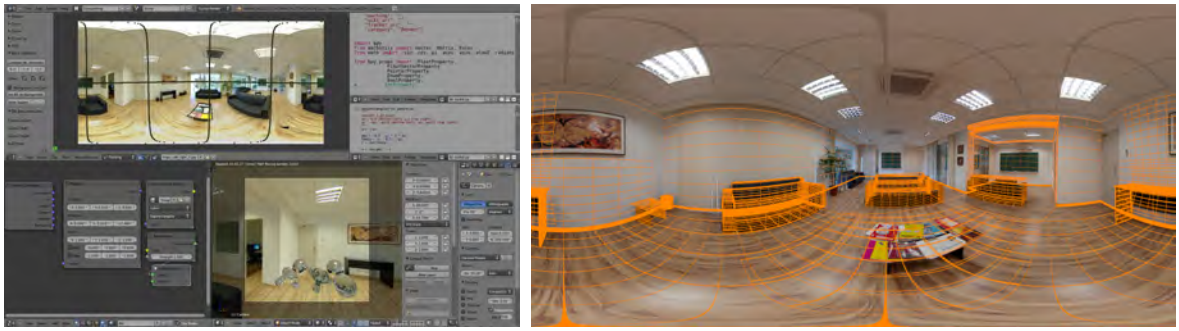
Quando não é possível reconhecer um retângulo no piso, o plano de orientação pode ser definido por pares de eixos que convergem no horizonte. Em ambientes de arquitetura e fácil reconhecer características arquitetônicas (bordas de janelas, linhas divisórias de paredes e teto) e usá-las como guias para a calibração e orientação do panorama.

A solução adoptada explora características planares da cena. O retângulo de calibração não precisa estar no piso. Pode muito bem ser parte de uma parede ou do teto. No entanto, em geral existem mais elementos para usar como pontos de referência no piso. Esta é também a razão pela qual o ambiente é reconstruído a partir do piso, como mostraremos na seção 5.5.

5.5 Construção da malha ambiente

Existem diferentes razões para querer modelar o ambiente. Com a reconstrução completa do espaço capturado no panorama podemos ter maior liberdade para movimentar a câmera e podemos calcular corretamente as sombras e reflexos dos objetos sintéticos inseridos na cena. No entanto, a imagem equiretangular por si só não é um mapa 3D do ambiente. O panorama equiretangular somente é uma parametrização da esfera. O mapa *HDR* tem informação de radiância, e um pixel pode ser localizado parcialmente no espaço (conhecemos a direção no mundo para cada pixel no mapa), mas nos falta informação adicional para determinar a posição dos pixels no espaço 3D (i.e., conhecemos a direção da iluminação, mas não a posição da fonte de luz). Se formos capazes de estimar a profundidade dos pixels da imagem omnidirecional, seremos capazes de reconstruir o espaço original representado pelo panorama. Assim, seremos capazes de renderizar interações de luz mais complexas, tais como reflexões especulares e calcular sombras precisas.

As malhas de ambiente servem para múltiplos propósitos em nosso trabalho: (a) Na fase de renderização, o ambiente é usado para determinar a posição da luz no mundo para os raios de reflexão (ver seção 5.7.2); (b) A profundidade da cena precisa ser calculada e armazenada no panorama *HDR*, pois é utilizada pelo renderizador para resolver testes de visibilidade (ver seções 5.1 e 5.6.2); (c) Parte do meio ambiente modelado serve como superfície de suporte para depositar as sombras e reflexões dos elementos sintéticos renderizados no panorama (consulte a seção 5.6.1); (d) elementos modelados podem ser transformados, mantendo intactas suas coordenadas de textura associadas ao panorama. Desta forma é possível produzir



(a) Toolkit IBL para Blender. **(b)** Malha do ambiente modelada sobre o panorama.

Figura 5-4: Toolkit IBL, permite calibrar o panorama e reconstruir a geometria do ambiente.

efeitos tais como deformação em partes do ambiente, por exemplo quando existe interação física com elementos sintéticos (ver seção 5.5.4); (e) Finalmente, na produção artística, é útil ter elementos estruturais para a animação/simulação física e para a oclusão visual dos elementos sintéticos.

Existem diferentes dispositivos e técnicas que permitem obter a malha ambiente. O método mais avançado envolve câmeras que capturam profundidade para gerar uma nuvem de pontos que pode ser associada com a fotografia capturada. Este método traz muitas melhorias de precisão e velocidade para o pipeline. Essas são características essenciais para produzir filmes de longa-metragem que exigem a captura de múltiplos ambientes para uma produção. Outro método consiste em capturar diferentes panoramas desde posições próximas, e usar um algoritmo de reconstrução de visão computacional para determinar a estrutura 3d com base na paralaxe entre as diferentes vistas. A desvantagem deste método é que ele requer pelo menos dois panoramas por cena e falhará se a principal área capturada não tem características distintas para identificar saliências.

Neste capítulo vamos explorar um método de modelagem manual mais acessível, que não requer nenhum dispositivo de captura especial ou algoritmo. Esta abordagem é, ao mesmo tempo económica e didática. Reconstruiremos a cena por partes. Devido às implicações do sistema de calibração implementado, o piso é a região que melhor conhecemos. Por isso, começamos por modelar as projeções de objetos no plano do piso. A figura (5-4) mostra o plug-in de Blender para calibrar o panorama e modelar a malha do ambiente.

5.5.1 Reconstrução do piso

Um sistema com a região do piso definido no espaço de imagem e pontos estruturais adicionais ajuda a construir elementos geométricos básicos. Dois pontos podem ser utilizados para definir os cantos de um quadrado. Três pontos podem delimitar o perímetro de um círculo ou definir um lado e a altura de um retângulo.

5.5.1.1 Círculos

Um círculo fica determinado pelo seu centro e o raio. São poucos os casos onde teremos o centro de um circulo visível e demarcado no panorama. Portanto, determinamos os círculos a partir de 3 pontos da circunferência. Mesmo se o circulo está parcialmente obstruído este método pode ser aplicado satisfatoriamente.

$$\begin{aligned} c_0 &= \tfrac{p_0+p_1}{2}, c_1 = \tfrac{p_1+p_2}{2} \\ \vec{v_0} &= (p_0.y - p_1.y, p_1.x - p_0.x, 0) \\ \vec{v_1} &= (p_1.y - p_2.y, p_2.x - p_1.x, 0) \end{aligned} \qquad \left| \qquad \begin{aligned} p_{center} &= c_0, \vec{v_0} \cap c_1, \vec{v_1} \\ radius &= ||p_0 - \vec{p}_{center}|| \end{aligned} \right. \tag{5.3}$$

5.5.1.2 Retângulos

Objetos que têm uma geometria retangular (por exemplo pés de uma mesa, baús, caixas) podem ser reconstruídos a partir de uma base retangular projetada no chão. A menos que o material seja transparente, ou unicamente estruturais (por exemplo, fios), eles terão, pelo menos, um canto ocluído pelo seu próprio corpo tridimensional. Neste caso, a base retangular precisa ser reconstruída a partir de 3 pontos.

Do ponto de vista matemático, três pontos podem definir um retângulo de varias maneiras. Na solução escolhida, o usuário precisa definir um dos lados do retângulo através de dois pontos e fixar a altura com o terceiro ponto. Desta forma, mesmo que o ponto não determine um ângulo reto quando projetada no chão com o lado retângulo definido, podemos usá-lo para calcular a distância entre os lados opostos do retângulo e garantir uma reconstrução perfeita da forma original.

A forma final é construída no espaço do mundo com a escala, orientação e posição definidas pela fórmula anterior. No espaço local preservamos uma geometria quadrada de dimensões unitárias para ajudar o artista a ajustar o tamanho real da geometria, definindo diretamente as escalas.

$$
\begin{array}{ll|ll}
\vec{v_{side}} & = p_1 - p_0 & height & = DistancePointLine(p_2, p_0, \vec{v_{side}}) \\
width & = ||\vec{v_{side}}|| & center & = c_0 + \frac{\vec{v_0} * height}{2} \\
\vec{v_0} & = (p_0.y - p_1.y, p_1.x - p_0.x, 0) & \alpha & = \vec{v_{side}} \cdot \vec{v_{(1,0,0)}} \\
c_0 & = \frac{p_0 + p_1}{2} & scale & = (width, height, 1)
\end{array}
\tag{5.4}
$$

5.5.1.3 Polígonos

Qualquer outro simplexo pode ser traçado para delimitar as fronteiras do piso ou a projeção de outros elementos da cena sobre o piso (por exemplo, a seção de uma coluna no piso). Os pontos selecionados no panorama são convertidos para o mundo 3D usando a altura da câmera da mesma forma que fizemos com as outras estruturas geométricas.

5.5.2 Mapeamento do fundo

Para reconstruir elementos que não estão apoiados no piso, é necessário ter uma maneira de editar a malha enquanto vemos sua projeção na imagem panorâmica. Tradicionalmente, isto é feito usando imagens individuais de vistas laterais do objeto a ser modelado, usadas como imagem de fundo para guiar a reconstrução, [64]. O mesmo mapeamento utilizado durante a renderização do fundo precisa ser replicado no viewport 3D do software de modelagem. Finalmente, a imagem do ambiente é mapeada esfericamente como um elemento de fundo, permitindo que ele seja explorado com uma câmera virtual com frustum limitado (FOV $<$ 180°), comumente suportado em qualquer software 3D.

A implementação priorizou uma abordagem não intrusiva no software para garantir que ela possa ser replicada, independentemente da suíte escolhida pelo estúdio/artista. Após cada ciclo de renderização da vista 3d, armazenamos o buffer de cor (com o alfa) e rodamos um Shader GLSL no espaço da imagem. Este shader recebe o buffer de cor original da vista 3D, o panorama e a inversa da matriz de transformação. Há duas razões para se passar a matriz de transformação: (a) utilizou-se a implementação clássica de shader GLSL no espaço da tela [53], que redefine as matrizes de projeção e modelview, a fim de desenhar um retângulo em toda a tela para que o programa shader possa funcionar como um Fragment Shader sobre ele. As matrizes são então resgatadas antes da nova vista ser configurada, e a inversa da

matriz é calculada na CPU; (b) É preciso levar em conta a orientação do mundo do panorama calibrado (ver seção 5.4.1).

$$M_{MVP}^{-1} = ((M_{ModelView} \cdot M_{Environment}) \cdot M_{Projection})^{-1} \tag{5.5}$$

O shader (algoritmo 5.1) realiza uma transformação do espaço de tela para o espaço de imagem do panorama e usa o canal alpha do buffer (o buffer de profundidade) para determinar onde desenhar o panorama. As coordenadas da textura do fundo são calculadas com a função $equirectangular(normalize(world))$ onde *world* é obtido com a função $glUnprojectGL(coords)$ que usa a inversa da matriz *Model View Projection* para converter as coordenadas de imagem para coordenadas do mundo.

O fundo é consistente, mesmo para diferentes lentes, frustums e orientações de câmera. Essa técnica libera o artista para criar inteiramente imerso no espaço 3D sem os problemas do espaço equiretangular do panorama.

Algoritmo 5.1: Shader para mostrar o fundo.

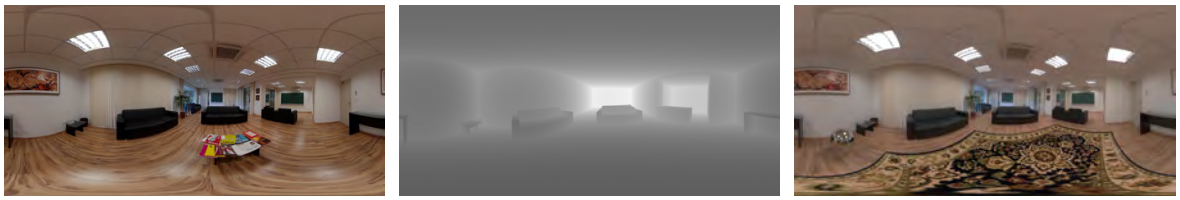
```
#version 120
uniform sampler2D color_buffer;
uniform sampler2D texture_buffer;
uniform mat4 projectionmodelviewinverse;
#define PI   3.14159265

vec3 glUnprojectGL(vec2 coords) {
  float u = coords.s * 2.0 - 1.0;
  float v = coords.t * 2.0 - 1.0;
  vec4 view = vec4(u, v, 1.0, 1.0);
  vec4 world = projectionmodelviewinverse * vec4(view.x, view.y, -view.z, 1.0);
  return vec3(world[0] * world[3], world[1] *  world[3], world[2] * world[3]);
}
vec2 equirectangular(vec3 vert) {
  float theta = asin(vert.z);
  float phi = atan(vert.x, vert.y);
  float u = 0.5 * (phi / PI) + 0.25;
  float v = 0.5 + theta/PI;
  return vec2(u,v);
}
void main(void)
{
  vec2 coords = gl_TexCoord[0].st;
  vec4 foreground = texture2D(color_buffer, coords);
  vec3 world = glUnprojectGL(coords);
```

```
  vec4 background = texture2D(texture_buffer, equirectangular(normalize(world)));
  gl_FragColor = mix(background, foreground, foreground.a);
}
```

5.5.3 Remoção de elementos do ambiente real

Alguns elementos presentes no ambiente podem ser indesejáveis na composição final. Um exemplo é a mesa de centro presente no meio da cena que usamos para montar nosso panorama de exemplo. Para remover esta mesa inserimos um tapete sintético na cena de maneira que o mesmo sobrepõe completamente a mesa no espaço da imagem panorâmica.

Na figura (5-5) é possível observar o tapete renderizado. Note também que não foi necessário modelar a mesa no mapa de profundidade (veja a figura central). Assim, se o objeto não existe no mapa de profundidade, é como se ele nunca existiu no mundo real. A mesma técnica de preenchimento pode ser usada para manipular a falta de capturas correspondentes ao polo Sul (ver seção 5.3.2).



(a) Panorama original. **(b)** Canal de profundidade. **(c)** Panorama objetos removidos.

Figura 5-5: Exemplo de eliminação de objetos durante o processo de reconstrução do ambiente. (b) O canal de profundidade gerado omite a mesa que foi removida (omitida) na reconstrução. (c) Um panorama com novos elementos e um tapete sintético utilizado para cobrir a área em que originalmente estava a mesa.

5.5.4 Projeção de coordenadas de ambiente

Estamos usando um método para transformar o ambiente que deforma a malha de suporte criada usando a imagem como referência. Uma vez que o artista está satisfeito com a precisão da malha do objeto a ser transformado, ela pode armazenar as coordenadas originais da imagem Panorama (UV) de cada vértice na própria malha. Daquele ponto em diante, quaisquer mudanças na posição vértices pode ser realizada como se a afetar os elementos do ambiente originais. Por exemplo, podemos simular uma bola pesada (elemento sintético) saltando em um sofá (elemento do ambiente) e animar a deformação das almofadas do sofá

para acomodar o peso da bola. Na figura (5-6) é possível observar um sofá modelado a partir da imagem de fundo e a deformação de malha sob a esfera sintetizada.

O renderizador deve ser capaz de utilizar esta informação para usar sempre a informação de luz dada pelas coordenadas armazenadas em vez da posição atual dos vértices. Isso funciona semelhante ao mapeamento UV tradicional e de texturas. Na verdade, este pode ser usado para mapeamento de toda a imagem nas malhas da cena (usando as coordenadas originais como UV e passando uma versão LDR do panorama como textura) nos casos em que o renderizador não pode ser portado para suportar os recursos de realidade aumentada implementados no integrador *ENVPath*, seção 5.8. Isto também pode ser usado para duplicar os elementos de cena em novos locais. A pintura pode se mover a partir de uma parede para outra, ladrilhos do piso pode ser utilizado para esconder um tapete presentes no meio ambiente, e assim por diante.

Para o integrador *ENVPath* o UV por si só não é suficiente. Os vértices no espaço de imagem são representados por uma direção, mas precisamos também, ser capazes de armazenar a profundidade original do ponto. Portanto armazenamos em uma camada de dados personalizada não o UV/direção, mas um vetor com a posição original de cada vértice. Aproveitamos que tanto formato de arquivo interno do Blender e o formato de malha do renderizador podem lidar com dados personalizados. O renderizador suporta malhas em formato ply, por isso, estendemos este formato com os parâmetros *wx*, *wy* e *wz*. Os dados precisam ser armazenados como no sistema de espaço do mundo (e não em coordenadas locais) para permitir que a malha sofra transformações ao nível do objeto.

Figura 5-6: Deformação de objetos reais através de coordenadas de textura ambiente. A geometria deformada é texturizada usando as coordenadas associadas à malha original, antes do processo de deformação.

5.5.5 Modelando além das áreas visíveis

Mesmo com a posição da câmera sendo estática podemos precisar de mais informações da cena além das que foram originalmente capturadas no panorama. Por exemplo, se uma esfera sintética reflexiva é colocada em algum lugar visível da cena, os reflexos que vemos na mesma podem ser de áreas obstruídas no panorama, tais como a parte de traz de objetos vistos no panorama. Outro caso é realizar movimentos de translação com a câmera no mundo 3D. Uma nova posição da câmera pode revelar superfícies que estavam obstruídas antes, e das quais não há nenhuma informação presente no panorama.

Há três soluções que consideramos para este problema: (1) Se o elemento ocluído não é relevante para a narrativa o mesmo pode ser simplesmente ignorado, como se nunca estivesse presente no mundo real (por exemplo, um objeto perdido debaixo do sofá, invisível a partir da posição da câmera); (2) Em outros casos, o artista precisa mapear uma textura para o elemento modelado e criar um material como faria em um pipeline de processamento normal. (3) Uma malha com coordenadas de textura provenientes de outras áreas do ambiente pode ser usada para preencher algumas lacunas (ver seção 5.5.4).

5.6 Pipeline de renderização para cenas mistas

O pipeline de renderização que apresentaremos a seguir é semelhante ao apresentado na seção 4.2. Entretanto, as novas informações geométricas de profundidade do mapa de iluminação, fazem necessário introduzir algumas modificações no processo de cálculo de interações da luz com os objetos para resolver alguns dos problemas do método anterior.

5.6.1 Classificação dos primitivos: sintéticos, de suporte e ambiente

Da mesma forma que o algoritmo apresentado no capítulo 4, o sistema de rendering proposto para mapas de iluminação com profundidade requer uma classificação especial dos primitivos da cena, pois os cálculos das contribuições luminosas e testes de visibilidade são heterogéneos e dependem da interação entre a cena real e os objetos sintéticos. Na figura (5-7) são ilustrados os primitivos utilizados.

- ***Primitivos sintéticos:*** os objetos que foram introduzidos na cena. Eles não existem na cena original capturada. Os cálculos de iluminação para estes objetos são realizados como nos renderizadores tradicionais.
- ***Primitivos de suporte:*** são superfícies presentes na cena original capturada que apresentam alguma interação visível com os primitivos sintéticos, tais como sombras e reflexos. O cálculo de iluminação para estes primitivos não é trivial porque é preciso adicionar as contribuições provenientes dos objetos sintéticos mantendo as características da cena original nas áreas onde a interação é mínima ou nula.
- ***Primitivos de ambiente:*** são todas as superfícies da cena original reconstruída que precisam ser levadas em conta no cálculo de sombras e reflexões para os outros tipos primitivos. Estes primitivos não precisam de nenhum cálculo de iluminação, pois como não interagem com os novos objetos sintéticos, sua radiância pode ser obtida diretamente a partir do mapa de iluminação.

Os níveis de detalhe na reconstrução geométrica do ambiente vão depender das necessidades da renderização final. Por exemplo, numa cena sem objetos altamente especulares, o ambiente pode ser simplificado para conter unicamente as características necessárias para uma iluminação correta, tais como o posicionamento correto de luzes, janelas e outras áreas luminosas que determinam as sombras e a iluminação principal da cena.

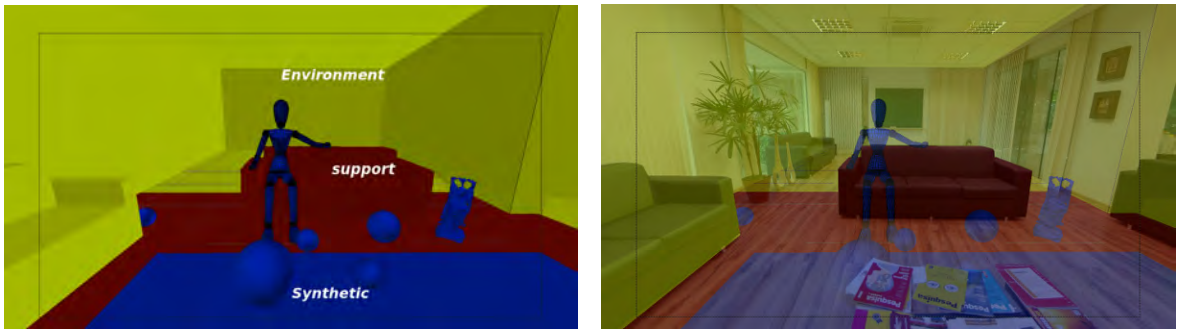


Figura 5-7: Classificação dos primitivos da cena de realidade mista. Os primitivos básicos para classificação são os triângulos que formam as malhas dos objetos. Em verde a malha ambiente, em vermelho os primitivos de suporte e em azul os primitivos sintéticos.

5.6.2 Visibilidade

Em cenas de realidade mista, onde temos objetos de *suporte* e *sintéticos*, o teste de visibilidade tradicional não permite resolver todos os tipos de interação entre os objetos da cena necessários para um cálculo correto das sombras neste tipo de cenas, como foi discutido na seção 4.2.2.

Utilizamos uma abordagem semelhante à proposta no capítulo 4. Entretanto, agora temos 3 tipos de primitivos: *sintéticos*, de *suporte* e de *ambiente*. Não realizaremos testes de visibilidade para um ponto p sobre a malha de ambiente, devido a que a mesma está associada com a estrutura de suporte geométrico do mapa de iluminação e será utilizada unicamente durante os testes de interseção para determinar as posições corretas das fontes de iluminação, e obter sombras e reflexões posicionadas corretamente.

O teste de visibilidade é aplicado somente durante o cálculo da iluminação direta. As demais interseções raio/objeto do algoritmo de ray tracing são efetuadas da maneira tradicional, considerando todos os primitivos da cena.

Como o mapa de iluminação tem informação de profundidade, cada amostra ω_i gera uma luz com uma posição l_i definida no mundo. O teste de visibilidade é realizado para o ponto p e a direção $\omega = l_i - p$ formada entre o ponto considerado e a posição da amostra luminosa no mundo. Assim, implementamos uma função $VisibTest(p, l_i, ignore\_support)$ para realizar o teste de visibilidade. A variável booleana *ignore_support* tem o papel de ajustar o resultado do teste se estivermos desconsiderando uma primeira interseção com um objeto de suporte. Uma interseção com om objeto da malha ambiente é equivalente a não ter interseção se comparado com o teste efetuado na seção 4.2.2.

A tabela (5.1) mostra os possíveis resultados do teste, de acordo a interseção determinada e o parâmetro *ignore_support* fornecido. Se for detectada uma interseção que está depois do ponto l_i, o resultado é verdadeiro, pois o ponto l_i é visível desde p.

Tabela 5.1: Teste de visibilidade para cenas aumentadas.

$VisibTest(p, l_i, ignore\_support)$		
Primeira interseção	***ignore_support***	
Tipo de objeto	**true**	**false**
Sintético	**false**	**false**
Suporte	**true**	**false**
Ambiente	**true**	**true**

5.6.3 Estimativas de iluminação direta

Tal como no capítulo 4, não consideramos objetos reais emissivos, logo $L_e(p,\omega_o) = 0$ para todo ponto p pertencente a uma superfície real de suporte. Se restringirmos os cálculos somente à radiância proveniente das fontes de iluminação L_d, a equação de espalhamento (4.1) é simplificada para

$$L_o(p,\omega_o) = \int_{\mathbb{S}^2} f(p,\omega_o,\omega_i) L_d(p,\omega_i) |\cos\theta_i| d\omega_i. \tag{5.6}$$

Como foi visto no capítulo 3, o cálculo da equação (5.6) em uma cena iluminada por um mapa de iluminação tem um custo computacional elevado. Em geral a estratégia adotada é aproximar o valor de (5.6) através de estimativas de Monte Carlo. Desta forma temos

$$L_o(p,\omega_o) = \frac{1}{N}\sum_{j=1}^{N} f(p,\omega_o,\omega_j)\frac{L_d(p,\omega_j)}{p(\omega_j)}\|\cos\theta_j\| \tag{5.7}$$

onde $\langle \omega_j, n_p \rangle > 0$, e $p(\omega_j)$ é a probabilidade de amostrar a direção ω_j no mapa de iluminação. O valor da probabilidade $p(\omega_j)$ é dado por

$$p(\omega_j) = \frac{Luminancia(\mathbb{M}(\omega_j))}{\int_{S^2} Luminancia(\mathbb{M}(\omega_s))d\omega_s}.$$

O cálculo da contribuição $L_o(p,\omega_o)$ depende do tipo de primitivo do ponto p. Assim, para cada tipo de primitivo, temos uma forma particular de calcular a contribuição da iluminação direta.

- ***Primitivos sintéticos:*** $L_o(p,\omega_o)$ é calculado com o estimador de Monte Carlo tradicional (5.7);
- ***Primitivos de ambiente:*** como os mesmos não têm interação direta visível com os objetos sintéticos, podemos atribuir a eles a radiância original armazenada no mapa de iluminação. Assim, a contribuição de iluminação direta é obtida diretamente do mapa de iluminação *RGB-D* como

$$L_o(p,\omega_o) = \mathbb{M}(\omega_p/|\omega_p|, |\omega_p|), \tag{5.8}$$

onde $\omega_p = WTL(p)$, e a função $WTL(p)$ transforma p do sistema de coordenadas do

mundo para o sistema de coordenadas da luz (mapa de iluminação).

- ***Primitivos de suporte:*** $L_o(p,\omega_o)$ é calculado utilizando o seguinte estimador

$$\frac{1}{N}\sum_{j=1}^{N}\mathbb{M}\left(\frac{\omega_p}{|\omega_p|},|\omega_p|\right)\cdot ES(p,n_p),\cdot\left\langle\frac{\bar{\omega}_j}{|\bar{\omega}_j|},n_p\right\rangle, \tag{5.9}$$

onde $ES(p,n_p)$ é um número real que indica o percentual de contribuição do mapa de iluminação no ponto p. Dedicaremos a seguinte secção para mostrar como é obtido o valor do fator $ES(p,n_p)$ e a estimativa de iluminação direta para primitivos de suporte.

5.6.3.1 Iluminação direta para primitivos de suporte

Superfícies de suporte precisam ser parcialmente renderizadas para incluir sombras e reflexões provenientes das interações diretas e indiretas com objetos sintéticos da cena. Se um ponto p sobre uma superfície de suporte não recebe nenhuma contribuição adicional de sombras ou reflexões por parte dos objetos sintéticos, seu valor de radiância final deve convergir para ser o mesmo que o que foi capturado e armazenado no mapa de iluminação *RGBD*. Como um ponto p sobre uma superfície de suporte representa um ponto do mundo real do qual conhecemos a radiância, pois a mesma foi capturada e armazenada no mapa de iluminação *RGBD*, é possível descrever a radiância observada em p como

$$L_o(p,\omega_o)=\mathbb{M}(\omega_p/|\omega_p|,|\omega_p|), \tag{5.10}$$

onde $\omega_p = WTL(p)$, ($WTL(p)$ transforma p do mundo para coordenadas do espaço da luz), e o termo $L_o(p,\omega_o)$ é proveniente da equação (5.6) de espalhamento da luz.

Para obter a igualdade da equação (5.10) utilizamos um fator pré-calculado

$$ES(p,n_p)=\frac{\int_{S^2}Lum(\mathbb{M}(\omega_s,z_s))\mathrm{d}\omega_s}{\int_{S^2}Lum(\mathbb{M}(\omega_j,z_j))\left\langle\frac{\bar{\omega}_j}{|\bar{\omega}_j|},n_p\right\rangle\mathrm{d}\omega_j}, \tag{5.11}$$

onde $\bar{\omega}_j = LTW(z_j\omega_j)-p$, e a função $LTW(z_j\omega_j)$ transforma $z_j\omega_j$ de coordenadas no espaço da luz para coordenadas do mundo.

O fator escalar $ES(p,n_p)$ representa o percentual de contribuição luminosa proveniente do mapa de iluminação *RGB-D* no ponto p, e é usado no estimador da equação (5.6). Assim, o estimador para primitivos de suporte é

$$L_o(p,\omega_o) = \frac{1}{N}\sum_{j=1}^{N} \mathbb{M}\left(\frac{\omega_p}{|\omega_p|}, |\omega_p|\right) \cdot ES(p,n_p) \cdot \left\langle \frac{\bar{\omega}_j}{|\bar{\omega}_j|}, n_p \right\rangle,$$

apresentado na equação (5.9) da seção 5.6.3. A escolha das direções ω_j são feitas por usando a função de probabilidade $p(\omega_j)$ descrita anteriormente. Desta forma, regiões de maior radiância serão escolhidas com maior frequência, e se seus raios forem ocluídos por algum objeto sintético, sua contribuição não será adicionada, gerando uma sombra no ponto p com intensidade proporcional à luz bloqueada.

O valor $ES(v,n_v)$ é calculado para todos os vértices v que pertencem a malhas que representam superfícies de suporte. Assim, durante a renderização, calculamos o fator escalar para um ponto p com coordenadas baricêntricas (a_1,a_2,a_3) no triângulo $T(v_1,v_2,v_3)$ como

$$ES(p,n_p) = a_1 \cdot ES(v_1,n_1) + a_2 \cdot ES(v_2,n_2) + a_3 \cdot ES(v_3,n_3). \tag{5.12}$$

5.6.3.2 Cálculo do fator de escala para BSDFs em primitivos de suporte

As duas integrais presentes na equação (5.11) podem ser calculada somando a contribuição de cada ponto de luz, i.e. somando a contribuição de todos os pixels do mapa de iluminação *RGBD*. Este processo tem um custo computacional elevado, pois depende da resolução do mapa (8k ou maior). Para simplificar as contas, o mapa de iluminação é discretizado e representado por um número limitado de luzes pontuais tais que a radiância total dos mesmos seja equivalente à radiância do mapa. Assim, no lugar de somar sobre todos os pixels do mapa de iluminação, os cálculos são realizados somando as contribuições sobre o pequeno grupo de luzes pontuais para obter uma estimativa da iluminação.

A discretização do mapa de iluminação *RGBD* é realizada com o algoritmo de corte mediano descrito em [18]. Calculando a equação (5.11) sobre a representação discretizada do mapa de iluminação através de um conjunto $L_d(i)$ de K luzes pontuais, obtemos

$$ES(p,n) = \frac{\sum_{i=1}^{K} L_d(i)}{\sum_{i=1}^{K} L_d(i)\,\langle \omega_i, n\rangle}, \quad with \;\; \langle \omega_i, n\rangle > 0, \tag{5.13}$$

onde

$$\omega_i = \frac{LTW(L_d(i)) - p}{|LTW(L_d(i)) - p|}.$$

Note que, diferentemente ao caso apresentado no capítulo 4, no caso de um mapa de

iluminação *RGB-d* a posição p assim como a posição da 3D da luz são fundamentais para determinar o valor de $ES(p,n)$.

5.7 Renderização com mapas de iluminação *RGBD*

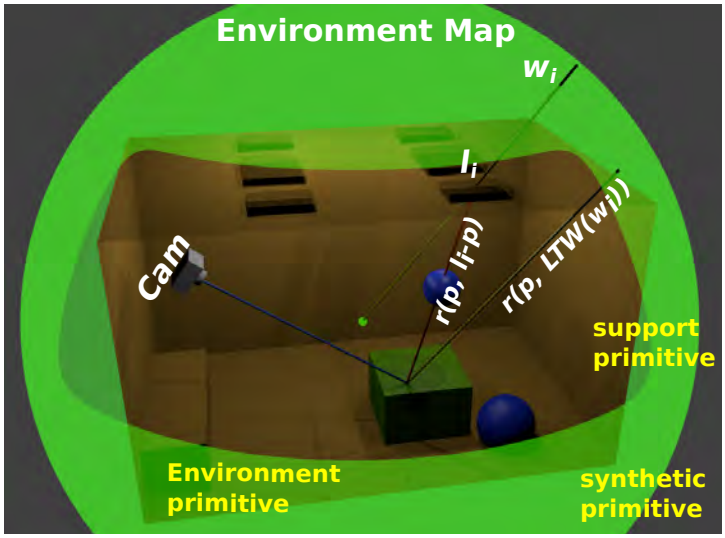
A principal deficiência dos algoritmos tradicionais de renderização com mapas de iluminação é que todos os cálculos de iluminação são realizados considerando o mapa de iluminação como um conjunto de luzes direcionais. Esta abordagem é suficientemente boa para algumas situações, particularmente quando as áreas de maior luminosidade do mapa (luzes principais) encontram-se muito distantes da cena no mundo real (por exemplo o sol). Por outra parte em cenas onde isto não acontece ou temos planejado incluir objetos sintéticos reflexivos, é necessário capturar a iluminação numa posição próxima da posição onde será inserido o objeto sintético, caso contrario os erros nas sombras e reflexões dos objetos sintéticos chamarão a atenção. Se for necessário introduzir vários objetos em diferentes posições da cena, o uso de um único mapa de iluminação convencional poderá ser insuficiente para conseguir o grau de realismo desejado, como pode ser apreciado na figura (5-16).

Como foi mencionado previamente, o proposito deste capitulo é modelar e sintetizar a cena usando um mapa de iluminação de alta resolução. Este mapa de iluminação é utilizado para renderizar o fundo da cena, aplicar texturas e modelar a geometria do ambiente real para fornecer uma distribuição da luz e das sombras que leve em conta a distribuição espacial.

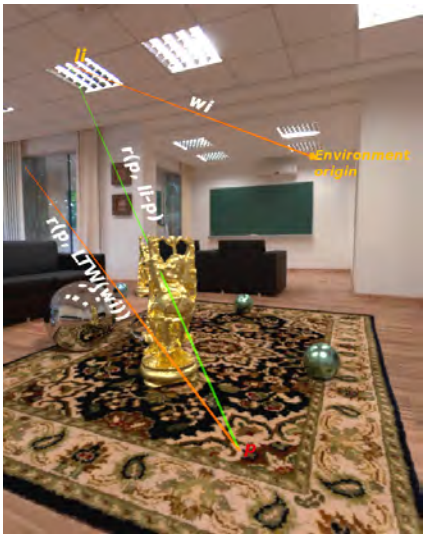
5.7.1 Cálculo de Sombras

Para cada raio proveniente da câmera que intersecta com a cena num ponto p sobre uma superfície, o integrador escolhe uma direção ω_i usando amostragem por importância do mapa de iluminação *RGBD* para calcular a contribuição de iluminação direta no ponto p. Seguidamente, o renderizador realiza um teste de visibilidade para determinar se a amostra de luz escolhida é ou não visível desde o ponto p.

No esquema tradicional de renderização, que usa o mapa de iluminação como um conjunto de luzes direcionais (cada pixel do mapa representa uma luz direcional), o teste de visibilidade é calculado para o raio $r(p,LTW(\omega_i))$, com origem em p e direção $LTW(\omega_i)$ com ω_i no sistema de coordenadas do espaço da luz. Esta abordagem introduz diversos erros quando o ponto p encontra-se afastado do ponto onde o mapa de iluminação foi capturado,



(a) Cálculo de visibilidade no integrador *ENVPath*.



(b) Exemplo do cálculo de visibilidade ilustrado na cena.

Figura 5-8: O raio $r(p, LTW(\omega_i))$ usado pelo esquema direcional não está ocluído, assim a luz l_i contribui com a iluminação do ponto p. O raio $r(p, l_i - p)$ utilizado pela nova abordagem posicional é obstruído por um objeto sintético e neste caso fornece uma estimativa correta, baseada na estrutura geométrica do mundo.

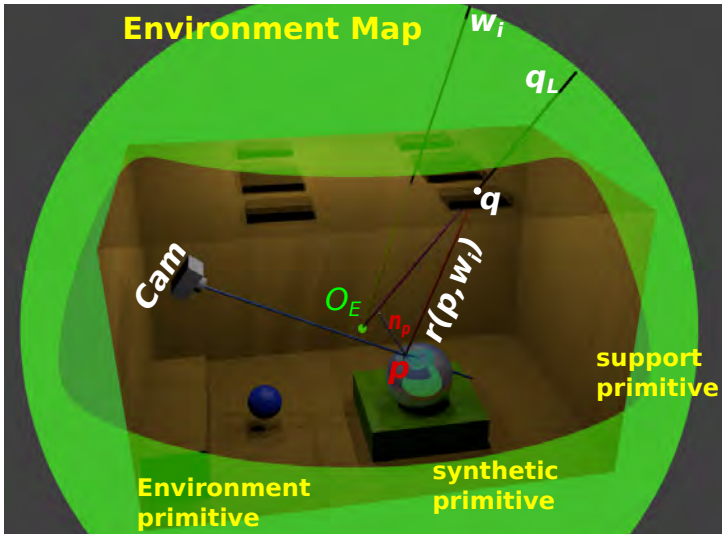
pois $LTW(\omega_i)$ só acomoda a rotação do vetor ω_i do espaço da luz para o espaço do mundo. Note que usando um mapa de iluminação tradicional todas as sombras terão a mesma orientação, pois somente é levada em conta a direção da iluminação.

Nosso algoritmo de renderização funciona de forma diferente. Devido às propriedades do mapa de iluminação *RGBD*, os testes de visibilidade conseguem determinar e utilizar a posição da amostra de luz do mapa e não somente sua direção.

O novo teste de visibilidade usa a direção ω_i da amostra de luz e a multiplica pelo valor de profundidade z_i correspondente, obtendo o ponto $z_i \cdot \omega_i$ nas coordenadas do espaço da luz (mapa de iluminação). O ponto $z_i \cdot \omega_i$ é transformado para o sistema de coordenadas do mundo onde obtemos $l_i = LTW(z_i \cdot \omega_i)$ ($z_i \cdot \omega_i$ é um ponto e não um vetor). Finalmente, o cálculo de visibilidade é realizado para o raio $r(p, l_i - p)$, com origem p e direção $l_i - p$ como é ilustrado na figura (5-8).

5.7.2 Cálculo de reflexões

Dado um ponto p sobre uma superfície, o integrador escolhe uma direção ω_i amostrando a *BRDF* (Bidirectional Reflectance Distribution Function) da superfície para adicionar as contribuições indiretas de reflexões (e refrações) no ponto p ao longo da direção ω_i. Para



(a) Cálculo de reflexões no integrador *ENVPath*.

(b) Exemplo do cálculo de reflexão ilustrado na cena.

Figura 5-9: Como pode ser visto, o raio $r(p,\omega_i)$ intersecta a cena no ponto q, sobre a malha ambiente. A radiância correspondente ao ponto q está armazenada na direção q_L no mapa de iluminação e não na direção ω_i fornecida pelo renderizador.

fazer isto, calculamos o ponto de interseção do raio $r(p,\omega_i)$, com origem p e direção ω_i, com a cena. Note que a interseção existe, pois exigimos que a o ambiente seja completamente modelado, mesmo que seja de maneira simplificada. Se o ponto de interseção q encontra-se sobre uma superfície sintética ou uma superfície de suporte, sua contribuição deve ser adicionada ao cálculo de contribuição de reflexão. Caso contrário, se a interseção é com a malha de ambiente, transformamos q de coordenadas do mundo para coordenadas do espaço do mapa de iluminação como $q_L = WTL(q)$. Finalmente a contribuição dada pela direção q_L no mapa de iluminação *RGBD* é usada para o cálculo de reflexão. A figura (5-9) ilustra esta ideia.

5.7.3 Coordenadas de textura do ambiente

As coordenadas de textura do ambiente foram implementadas para dar suporte a efeitos de rendering tais como deformação do ambiente original quando existe uma interação física com objetos sintéticos. Outra aplicação é a copia de textura de uma região para outra. Isto pode ser uma ferramenta poderosa para texturizar superfícies de suporte em regiões ocluídas do mapa de iluminação. Na figura (5-10) mostramos como a ferramenta pode ser utilizada para aplicar deformações nos objetos de suporte quando os mesmos interagem com objetos sintéticos.

Figura 5-10: As duas esferas sobre o sofá foram adicionadas à cena. O sofá é real e foi capturado e armazenado no mapa de iluminação com profundidade. Ao introduzir as esferas, deformamos parte da modelagem do sofá para simular a interação com as esferas. Na modelagem original do ambiente foi feito o casamento das coordenadas de textura da geometria com o mapa de iluminação, assim posteriores deformações geométricas do ambiente arrastam a textura.

5.8 ENVPath: path tracing para cenas de realidade mista com mapas de iluminação *RGB-D*

O algoritmo de path tracing desenvolvido, é semelhante ao apresentado no capítulo 4 e resolve a equação de transporte de luz realizando a construção incremental do caminho. Entretanto, como veremos a seguir, algumas adaptações são necessárias. Um caminho começa no vértice p_0 na câmera. Em cada vértice p_k, a *BSDF* é amostrada para gerar uma nova direção. O vértice p_{k+1} é encontrado traçando um raio desde p_k na direção amostrada e determinando a primeira intersecção com algum objeto da cena. O processo é interrompido quando a interseção com a cena é um vértice p que pertence a um *primitivo ambiente*, ou quando o algoritmo passa no teste de *Roleta Russa* que interrompe o caminho.

Em cada vértice p_k com $k = 1, \cdots, i-1$ calculamos a radiância espalhada no ponto p_k na direção $p_{k-1} - p_k$. Quando o ponto p_k pertence a uma superfície sintética ou de suporte, o calculo de espalhamento é realizado utilizando as equações (5.7) e (5.9) da seção 5.6.3 respectivamente. Caminhos nos quais p_1 pertence à malha ambiente são calculados diretamente com a equação (5.8).

Os caminhos são classificados como **caminho real**, **caminho sintético** o **caminho misto** segundo a natureza dos seus vértices, da mesma forma que apresentamos no capítulo 4.

O último vértice de um caminho, p_i, pertence a uma fonte de luz (em nosso caso é alguma amostra do mapa de iluminação). Quando usamos os mapas de iluminação tradicionais, os pontos p_i ficam determinados quando um raio não intersecta com a cena, e nesse caso é associado com uma luz direcional armazenada no mapa. No caso de um mapa de iluminação *RGB-D* os pontos p_i são determinados como pontos que estão sobre superfícies de ambiente.

É neste momento que a malha ambiente é utilizada. Na verdade nenhum raio escapa do interior da malha ambiente mais a malha de suporte, pois a união delas é uma envoltura que representa a estrutura geométrica do mapa de iluminação.

Nos caminhos reais não é necessário realizar cálculos de contribuições, pois sua radiância já está armazenada no mapa de iluminação $\mathbb{M}$ que é utilizado também como fundo da cena. Para este tipo de caminhos agrupamos toda a informação de radiância no vértice p_1 na forma de iluminação direta, a fim de evitar cálculos nos vértices seguintes.

Os caminhos sintéticos e mistos são calculados de maneira similar, contabilizando a contribuição da iluminação direta em cada vértice do caminho. A diferença está na estimativa de Monte Carlo utilizada em cada caso, dependendo do tipo de primitivo do vértice considerado. A figura (5-11) ilustra os tipos de caminhos e o cálculo que é realizado nos vértices em cada caso.

Para implementar o algoritmo, não é necessário saber a priori o tipo de caminho. De igual forma que no algoritmo 1, basta conhecer se os vértices do caminho parcial são todos de suporte ou se existe pelo menos um vértice que pertence a uma superfície sintética. A informação é manipulada dentro do algoritmo através de uma variável booleana *path_type*.

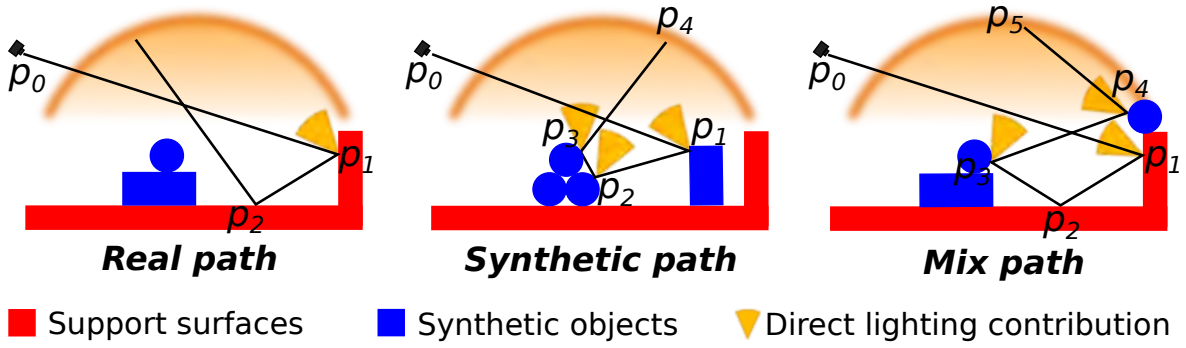


Figura 5-11: Classificação de tipos de caminhos. Os cones amarelos indicam que a contribuição de iluminação direta no vértice p_i foi calculada e será adicionada ao cálculo da contribuição global do caminho.

5.9 Resultados e considerações

Como explicamos durante o capítulo, o aspecto principal do processo é integrar elementos sintéticos na imagem omnidirecional capturada de maneira foto-realista. Nas figuras (5-12) e (5-15) é possível observar elementos sintéticos integrados com o panorama original. Em cada caso a iluminação e as sombras dos objetos sintéticos sobre os elementos do ambiente são

Algorithm 3 ENVPath Integrator

$L \leftarrow 0$; ▷ Radiance contribution
$r_0 \leftarrow ray(p_0, -\omega_0)$; ▷ Camera ray
$path\_type \leftarrow false$; ▷ true if path contains vértices from synthetic primitives
$T \leftarrow 1$; ▷ Path throughput factor
for $i \leftarrow 0$; $MaxLength$ **do**
    $p_{i+1} \leftarrow SceneIntersect(r_i)$;
    $T \leftarrow T \cdot ThroughputAtenuation(p_i, p_{i+1})$;
    if $type(p_{i+1}) = ENVIRONMENT$ **then**
        if ($path\_type = true$) **OR** ($i = 0$) **then**
            $\omega_{p_{i+1}} \leftarrow WorldToLight(p_{i+1})$;
            $L \leftarrow L + T \cdot \mathbb{M}\left(\frac{\omega_{p_{i+1}}}{|\omega_{p_{i+1}}|}, |\omega_{p_{i+1}}|\right)$; ▷ equação (5.8)
        end if
        break; ▷ Terminate the path
    end if
    if $type(p_{i+1}) \neq SUPPORT$ **then** ▷ $type(p_{i+1}) = SYNTHETIC$
        $path\_type \leftarrow true$;
    end if
    if $i = MaxLength$ **then**
        break; ▷ Terminate the path
    end if
    $L \leftarrow L + T \cdot DirectLighting(p_{i+1}, path\_type, i)$; ▷ algoritmo 4
    $r_{i+1} \leftarrow SampleNewDirection(p_{i+1})$; ▷ Sample BSDF to get new path direction
    $T \leftarrow T \cdot f(p_{i+1}, -dir(r_i), dir(r_{i+1}))$; ▷ Include BSDF attenuation
    if $Prob() < END\_PROBABILITY$ **then** ▷ Continues the path construction or not ?
        break; ▷ Terminate the path
    else
        $T \leftarrow T / Prob()$; ▷ Increase path contribution
    end if
    if $type(p_{i+1}) = SUPPORT$ **then**
        $T \leftarrow T \cdot SupportAttenuation(p_i, p_{i+1})$;
    end if
end for
return L;

Algorithm 4 Direct Lighting Estimator

Procedure DirectLighting(p, ω_o, $path\_type$, $RayDepth$)
$Ld \leftarrow 0$;
$mis \leftarrow UseMultipleImportanceSampling(p)$;
$\omega_i \leftarrow SampleLightDir(\mathbb{M})$;
$l_i \leftarrow LightToWorld(\omega_i)$;
$\omega_p \leftarrow Normalize(l_i - p)$;
if $type(p) = SUPPORT$ **then**
    if ($path\_type = true$) **OR** ($RayDepth = 0$) **then**
        if ($mis = false$) **OR** ($RayDepth = 0$) **then**
            if $VisibTest(p, l_i, true) = true$ **then**
                $Ld \leftarrow Ld + ES(p, n_p)\mathbb{M}(\omega_i)\langle\omega_p, n_p\rangle$; ▷ compute original background color
            end if
        else
            if $VisibTest(p, l_i, false) = true$ **then**
                $Ld \leftarrow Ld + ES(p, n_p)\mathbb{M}(\omega_i)\langle\omega_p, n_p\rangle$; ▷ compute original color

                $Ld \leftarrow Ld + \frac{f(p,\omega_o,\omega_p)\mathbb{M}(\omega_i)\langle\omega_p,n_p\rangle}{pdf(\omega_i)}$; ▷ add light from indirect reflection
            end if
        end if
        if $mis = true$ **then** ▷ Trace a 2nd shadow ray by sampling the BSDF.
            $\omega_i \leftarrow SampleBSDFDir(p, \omega_o)$;
            $r \leftarrow ray(p, \omega_i)$;
            $q \leftarrow SceneIntersect(r)$;
            if $type(q) = ENVIRONMENT$ **AND** $RayDepth > 0$ **then**
                $\omega \leftarrow ConvertWorldPosToMap(q)$;
                $Ld \leftarrow Ld + \mathbb{M}(\omega)$;
            end if
        end if
    end if
else ▷ $type(p) = SYNTHETIC$
    if $VisibTest(p, l_i, false) = true$ **then**

        $Ld \leftarrow Ld + \frac{f(p,\omega_o,\omega_p)\mathbb{M}(\omega_i)\langle\omega_p,n_p\rangle}{pdf(\omega_i)}$;
    end if
    if $mis = true$ **then** ▷ Trace a 2nd shadow ray by sampling the BSDF.
        $\omega_i \leftarrow SampleBSDFDir(p, \omega_o)$;
        $r \leftarrow ray(p, \omega_i)$;
        $q \leftarrow SceneIntersect(r)$;
        if $type(q) = ENVIRONMENT$ **then**
            $\omega \leftarrow ConvertWorldPosToMap(q)$;
            $Ld \leftarrow Ld + \mathbb{M}(\omega)$;
        end if
    end if
end if
return Ld;
end procedure

calculadas utilizando o panorama omnidirecional *HDR* original como mapa de iluminação *RGB-D*. Foi necessário modelar a malha ambiente com o plugin *IBLtoolkit* para gerar o canal de profundidade para o mapa.

O cálculo de reflexões especulares pode ser apreciado com maior detalhe na esfera metálica da figura (5-16). Na figura comparamos os reflexos produzidos pelo mapa com estrutura *RGB-D* com os reflexos produzidos utilizando um mapa convencional, sem informação posicional. As esferas e o tapete são sintéticos e são renderizados com ambos os métodos. Mas, a presença das malhas do ambiente original faz com que a reflexão seja consistente, especialmente na transição entre objetos sintéticos (por exemplo, tapete) e elementos do ambiente (por exemplo, piso de madeira).

A calibração correta da cena e o cálculo correto da posição das sombras e reflexos ajudam proporcionar um maior realismo aos objetos sintéticos adicionados à cena. Na figura (5-13) é possível observar as esferas e o tapete que foram inseridos na cena. A imagem mostra como uma parte do panorama (frustum limitado) seria visualizada numa tela convencional.

Finalmente, exploramos alguns movimentos de translação da câmera. Na figura (5-14) é possível observar uma parte da cena renderizada desde duas posições de câmera diferentes. O resultado é satisfatório, desde que o ambiente tenha uma modelagem correta. Para pequenas mudanças de câmera não é mesmo um problema.

Também testamos o framework com panoramas de terceiros, disponíveis em Internet. Para ele reconstruímos o canal de profundidade através da modelagem da malha do ambiente e adicionamos novos elementos sintéticos, para logo realizar a renderização. Obtivemos resultados satisfatórios, figura (5-12). Na figura (5-17) são apresentadas duas vistas parciais do panorama com elementos sintéticos adicionados e renderizados com o framework desenvolvido.

Nos casos abordados nesta capitulo, a malha de ambiente foi modelada pelo artista, com base na calibração e dados visuais do panorama omnidirecional *RGB*. Posteriormente renderizamos a profundidade da malha base e adicionamos a informação no canal de profundidade do panorama. No capítulo 7 descreveremos como podemos obter uma malha quando conhecemos a informação de profundidade do panorama omnidirecional. Assim, teremos um pipeline mais completo, onde podemos trabalhar a partir de um panorama com ou sem profundidade.

Figura 5-12: Imagens renderizadas partir de um panorama da Internet para validar o framework. O panorama original é cortesia de Sam Schad, www.blendedskies.com.

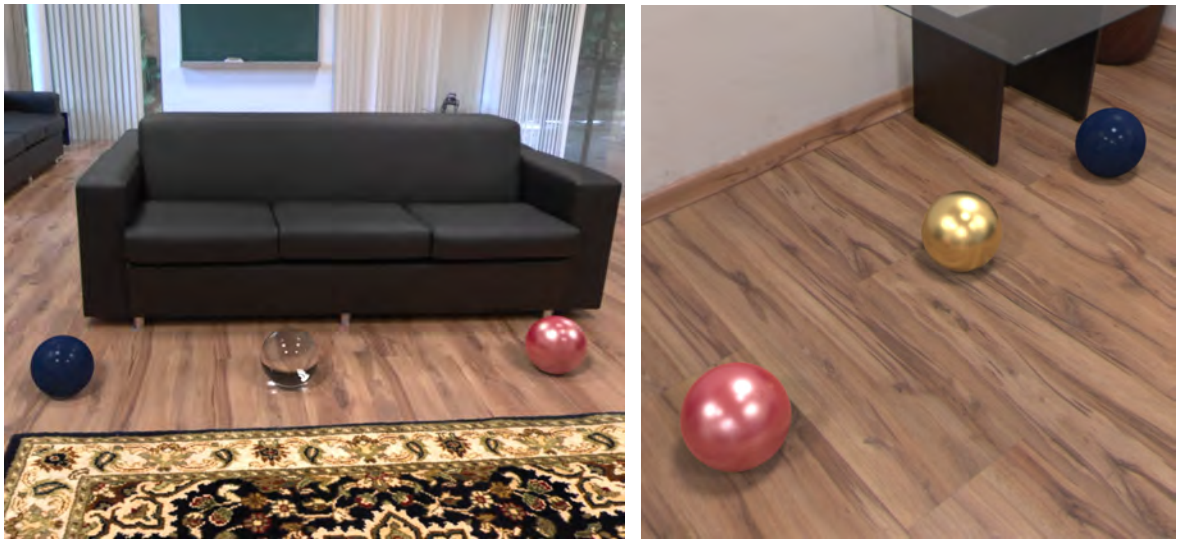
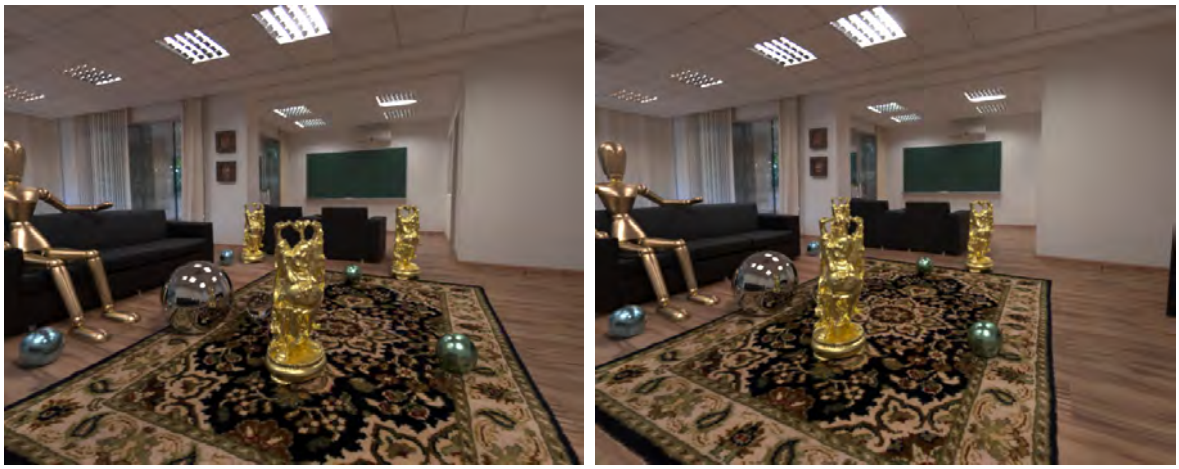


Figura 5-13: Sombras usando um mapa de iluminação *RGB-D*. A bola vermelha tem sombras diferentes da bola azul.



(a) Câmera posicionada na origem do mapa de iluminação.

(b) Câmera deslocada da origem do mapa de iluminação.

Figura 5-14: Efeitos do deslocamento de câmera. Uma parede parcialmente visível em (a) é ocluída em (b).

Figura 5-15: A imagem superior é o panorama omnidirecional completo com realidade mista. Nas imagens inferiores observamos duas vistas da mesma cena renderizadas com uma lente olho de peixe.

(a) Renderização com um mapa tradicional. **(b)** Renderização com um mapa *RGB-D*.

Figura 5-16: Comparação entre as reflexões produzidas pelos mapas de iluminação tradicionais (mapa direcional) (a); e mapas de iluminação *RGB-D* (b).

(a)

(b)

Figura 5-17: Renderizações do panorama omnidirecional com realidade mista feitas com uma câmera perspectiva. Os objetos sintéticos incluídos na cena são iluminados com o panorama e renderizados sobre o fundo proveniente do panorama.

Capítulo 6

Panoramas omnidirecionais com múltiplas camadas - *POMC*

Nos capítulos 3, 4 e 5 estudamos o problema de renderização foto-realista com mapas de iluminação omnidirecionais *RGB* e *RGB-D*. Neste capitulo apresentaremos uma nova perspectiva para trabalhar com informação omnidirecional.

Como temos visto até agora, muitos trabalhos se preocupam com diversos aspectos dos mapas de iluminação omnidirecionais. Alguns procuram formas eficientes de amostrar a iluminação, outros em reproduzir o aspecto de variância espacial da iluminação, mas não exploram a fundo maneiras de utilizar esta informação contida no mapa para reproduzir completamente o ambiente capturado, tal como fizemos no capítulo 5. No capítulo 5 foi possível resolver vários problemas frequentes no contexto de renderização foto-realista de cenas de realidade mista, mas nem por isso podemos nos dar por satisfeitos. Como mencionamos, ainda existem problemas relacionados com a falta de informação complementar das regiões ocluídas ao momento da captura do mapa de iluminação. Em geral estas informações podem ser irrelevantes em termos de da sua contribuição para a iluminação, mas são necessárias para reproduzir efeitos indiretos de iluminação, tais como reflexões, e também no momento de oferecer liberdade de movimento para a câmera.

Foram anunciados recentemente vários tipos de equipamentos que vão permitir nos próximos anos realizar a capturas de informação geométrica e de radiância da cena de maneira relativamente simples. Porém nada é de graça! Este tipo de tecnologia pode ser de acesso restrito em muitos casos, como a *Lytro Immerge* anunciada recentemente, que promete soluções a inúmeros problemas no pipeline de produção para cinema. O grande problema de

capturar toda a informação disponível no ambiente de maneira densa, é o de armazenamento e manipulação posterior dos dados. Se uma fotografia de frustum limitado em *HDR* de alta resolução ocupa uns quantos megabytes de espaço, tente imaginar imagens *HDR* omnidirecionais de alta resolução amostradas de maneira densa numa região do espaço. Os problemas que vislumbramos para o futuro estão relacionados com a forma de acessar a informação de maneira eficiente e usá-la convenientemente.

Os novos desafios agora são os de procurar representações adequadas para estruturar os dados brutos capturados ou renderizados e processa-los adequadamente para seu posterior uso em computação gráfica. Tendo em vista os avanços recentes das tecnologias de captura de imagem e profundidade, vamos apresentar uma nova variedade de panoramas, os panoramas omnidirecionais *RGB-D* com múltiplas camadas (*POMC*), que buscam estruturar adequadamente a informação contida na cena e codifica-la num panorama omnidirecional que minimiza a redundância da informação capturada. Estes panoramas também abrem uma nova gama de aplicações na área de visualização interativa, tais como: visualização estéreo calculada a partir de um único panorama *POMC*, manipulação de disparidade e efeitos de paralaxe. Esta representação também proporciona novas e melhores soluções para problemas de renderização com iluminação baseada em imagem, *IBL* (*image-based lighting*), como os tratados nos capítulos anteriores.

O conceito de panoramas omnidirecionais com múltiplas camadas *POMC*, pode ser pensado como uma evolução do panorama com profundidade proposto nos trabalhos de Felinto et al. [23, 63], e como será mostrado posteriormente, permite resolver alguns dos problemas relacionados à movimentação de câmara e posicionamento de objetos sintéticos em cenas reais capturadas, tais como oclusão/desoclusão de objetos e variação da iluminação indireta em função da posição de câmera que não podem ser resolvidos com os panoramas atuais. Em Felinto et al. [23] alguns problemas têm que ser resolvidos com ajuda do usuário, sendo necessário copiar e colar texturas de uma região da cena para outra, a fim de preencher a informação de regiões ocluídas do panorama que são refletidas pelos objetos sintéticos. Estes problemas em particular podem ser minimizados e até eliminados com o uso de um *POMC*.

6.1 A função plenóptica

Adelson e Bergen [2] definiram a *função plenóptica* como o feixe de raios visível a partir de qualquer ponto do espaço, a qualquer momento e em todos os comprimentos de onda. A função plenóptica descreve toda a energia que chega num ponto do espaço a partir de todas as direções. Adelson e Bergen [2] ainda formalizaram uma descrição funcional da função plenóptica, dada por:

$$\rho = P(\phi, \theta, \lambda, V_x, V_y, V_z, t), \tag{6.1}$$

onde (V_x, V_y, V_z) é a posição do observador, ϕ e θ são as coordenadas esféricas da direção de observação, λ é o comprimento de onda da luz e a variável t descreve o tempo.

Na equação (6.1), os parâmetros ϕ e θ indicam uma direção de observação a partir do ponto (V_x, V_y, V_z), logo podemos substituir estes parâmetros pela direção ω associada e reformular a função plenóptica como:

$$\rho = P(\omega, \lambda, V_x, V_y, V_z, t), \tag{6.2}$$

onde $\omega \in \mathbb{S}^2$. Esta forma da função plenóptica será útil para as formulações que apresentaremos nas próximas seções.

Pode-se entender a *função plenóptica* como sendo uma função capaz de descrever todas as possíveis vistas de uma determinada cena. Dizemos que temos uma amostra completa *função plenóptica* se, para um ponto coberto pela função, toda a abóboda que está à sua volta pode ser vista, e uma amostra incompleta quando apenas uma parte desta abóbada é visível.

Vários autores afirmam que todos os algoritmos de renderização baseada em imagens (*IBR* image based rendering) se resumem a uma abordagem particular da função plenóptica. Desta maneira, um possível enunciado para a maioria dos problemas de *IBR* pode ser descrito da seguinte forma: *"dado um conjunto de amostras (completas ou incompletas) da função plenóptica, o objetivo da solução proposta consiste em gerar uma representação contínua desta função"*, [42].

6.1.1 Panoramas omnidirecionais e a função plenóptica

Se a função *plenóptica* é construída para um único ponto de vista e um instante de tempo, sua dimensionalidade é reduzida de 7 para 2. Este é o princípio usado em mapeamento de reflexão, [6], onde a vista do ambiente desde uma posição fixa é representada por uma imagem 2D. Assim, do ponto de vista da função plenóptica, um panorama *HDR* omnidirecional tradicional é uma amostra completa da função plenóptica para um ponto no espaço e um tempo determinado. Quando adicionamos uma componente de profundidade ao panorama, podemos resolver alguns problemas inerentes ao processo de iluminação correta da cena e em geral conseguimos, além da amostra pontual completa, uma aproximação parcial da função plenóptica numa vizinhança da amostra completa. No capítulo 5 introduzimos o conceito de *Panorama Omnidirecional com Profundidade* [23], como sendo um mapa esférico completo onde para cada direção ω, além da radiância, também é conhecida a distância até a primeira interseção com um objeto da cena.

Em casos simples é possível reconstruir a função plenóptica completamente a partir de um ponto no espaço. Por exemplo, imagine um quarto completamente vazio com luzes no teto, e cujas superfícies sejam de um material lambertiano. Neste caso um panorama omnidirecional *RGB-D* capturado no ponto médio do quarto fornecerá toda a informação de radiância necessária para reproduzir uma amostra completa da função plenóptica em qualquer ponto do interior do quarto. Claro que isto é um exemplo muito limitado e meramente ilustrativo. Em geral os objetos de uma cena não são todos compostos por materiais lambertianos e quase sempre existem pontos cegos na cena. Isto não deve nos desalentar. Em geral, dependendo do tipo de aplicações, o interesse recai sobre certas propriedades da cena. Às vezes estamos interessados em saber a distribuição da radiância, para aproximar a iluminação desde diferentes pontos da cena. Nestes casos um mapa com profundidade capturado desde um ponto que maximize a radiância visível, i.e., desde onde nenhuma luz importante seja bloqueada, permitirá uma estimativa plausível da iluminação. Por outra parte, se o interesse é renderizar cenas mistas com objetos sintéticos reflexivos, é importante dispor de uma amostra completa da função plenóptica na posição do objeto sintético, para reproduzir os reflexos corretamente. A solução tradicional nestes casos é ter mapas omnidirecionais adicionais capturados nos pontos de inserção dos objetos especulares na cena real.

Para obter uma formulação matemática consistente para os panoramas estendidos, começaremos apresentando uma versão expandida da função plenóptica. A formulação da função

plenóptica restrita ao ponto $p = (x_0, y_0, z_0)$ no tempo t_0 e sobre todas as longitudes de onda λ, ou seja $P(\omega, \lambda, x_0, y_0, z_0, t_0)$, será denotada simplesmente por $P(\omega)$. A formulação funcional esta restrição da função plenóptica é definida como

$$P : \mathbb{S}^2 \to C, \tag{6.3}$$

onde C é um espaço de cor onde será representada a iluminação correspondente à avaliação da função plenóptica nos parâmetros estabelecidos. Nesta tese trabalharemos com o espaço de cor *RGB* de modo que consideraremos $C \subset \mathbb{R}^3$.

Na prática, não trabalhamos diretamente com direções na esfera $\mathbb{S}^2$, mas sim através de parametrizações da mesma, portanto podemos representar a função P como uma composição de funções

$$\begin{aligned} &P = f \circ g, \\ &g : \mathbb{S}^2 \to U \subset \mathbb{R}^2 \\ &f : U \subset \mathbb{R}^2 \to C \end{aligned} \tag{6.4}$$

onde g é uma parametrização da esfera $\mathbb{S}^2$.

A descrição funcional anterior corresponde ao um panorama omnidirecional tradicional, onde para cada direção ω a função $P(\omega)$ devolve o valor de radiância nessa direção.

Para representar um panorama omnidirecional *RGB-D*, além da radiância na direção ω, é necessário conhecer a distância ao longo de ω até o primeiro objeto da cena. Para lograr isto, vamos estender o contradomínio da função plenóptica de maneira tal que além da radiância também devolva a distância até o objeto que emite aquela radiância. Para um ponto $p = (x_0, y_0, z_0)$ e tempo t_0 a versão estendida da função plenóptica será formulada como

$$\mathcal{P} : \mathbb{S}^2 \to C \times \mathbb{R}^n, \tag{6.5}$$

com $\mathcal{P} = (P, Q)$, e tal que

$$\begin{aligned} &P : \mathbb{S}^2 \to C, \\ &Q : \mathbb{S}^2 \to \mathbb{R}^n, \end{aligned} \tag{6.6}$$

onde P é a função plenóptica usual, e Q é a função de propriedades estendidas. Em particular, para um panorama omnidirecional *RGB-D*, temos $Q : \mathbb{S}^2 \to \mathbb{R}$, onde Q é uma função de distância.

Figura 6-1: Panorama omnidirecional *RGB-D*. Na imagem colorida é possível observar a radiância do panorama, enquanto a imagem em tons de cinza representa a profundidade de cada direção.

6.2 Panoramas omnidirecionais com múltiplas camadas

Suponhamos que temos armazenada a função plenóptica de uma região Ω do espaço. Isto significa que em cada ponto $p \in \Omega$ é possível reconstruir uma amostra completa da função plenóptica (i.e., um panorama omnidirecional). É provável que muitos pontos da cena sejam visíveis desde qualquer ponto da região Ω. Outros pontos da cena são vistos em sub-regiões de Ω. Assim, se pensarmos em termos de visibilidade, muita informação pode resultar redundante. Se considerarmos cenas reais, é possível que muitos dos pontos visíveis em toda a região Ω tenham pouca ou nenhuma variação de radiância, quando visualizados desde diferentes pontos da região Ω (isto acontece, por exemplo, com pontos sobre superfícies difusas). Assim, também existe alguma redundância em termos da radiância percebida. Começaremos por pensar no problema da redundância respeito à visibilidade. A eliminação de redundância na informação de radiância implica em um processamento adequado dos dados e a estimativa de propriedades dos materiais da cena, para ré-sintetizar parcialmente a função plenóptica e calcular as diferenças de radiância.

Dado um *panorama omnidirecional com profundidade* de uma cena real, é natural que existam descontinuidades entre os valores de profundidade, dado que a cena geralmente contem objetos a diferentes distâncias do observador (figura (6-4b)). Em geral os pontos de descontinuidade na função de distância encontram-se na borda dos objetos. Usando as descontinuidades da função distância como guia, podemos dividir a esfera $\mathbb{S}^2$ em regiões V_k, tais que todos os pontos de descontinuidade da função de distância estejam localizados nas fronteiras das regiões V_i e nunca no interior das mesmas.

Até o momento falamos de *panoramas omnidirecionais com profundidade*. Eles têm a propriedade de que para o ponto de captura do mesmo, além da radiância incidente desde todas as direções, também conhecemos a profundidade da cena em todas as direções. Se

(a) Câmara na origem do panorama. **(b)** Câmara deslocada da origem do panorama.

Figura 6-2: Efeitos do movimento de câmara. Um Puff oclui um canto da sala na imagem (a), enquanto que ao mover a câmera o canto da sala já não é ocluído pelo Puff, porém não temos informação para renderizar a região desocluída na imagem (b).

renderizamos uma cena cuja câmara está posicionada na origem do panorama, não temos problemas pois dispomos de suficiente informação para gerar uma imagem de saída que nada mais é do que uma porção limitada do panorama (figura (6-2)). Se desejarmos renderizar uma vista desde uma posição de câmera diferente à origem do panorama, podemos ter sérios problemas, mesmo contando com a profundidade, dado que alguma parte da cena que estava ocluída pode ficar visível, mas um *panorama omnidirecional com profundidade* não tem armazenada esta informação (figura (6-2a)). Em alguns casos podemos reconstruir uma visualização correta da cena, desde que a região captada pela câmara não contenha descontinuidades de profundidade (figura (6-3)), mas em outros casos podemos ter desoclusão de objetos ocultos e a falta de esta informação no panorama impedirá de reconstruir uma visualização convincente, (figura (6-2b)).

Como os problemas de movimentar a câmera estão ligados à falta de informação adici-

(a) Câmara na origem do panorama. **(b)** Câmara deslocada da origem do panorama.

Figura 6-3: Efeito do movimento de câmara. Quando a região enquadrada pela câmara não apresenta descontinuidades de profundidade, um panorama com profundidade é suficiente para conseguir uma renderização.

onal tanto de profundidade quanto de cor nas regiões circundantes aos pontos de descontinuidade da *função plenóptica* restrita à profundidade, pensamos numa maneira alternativa de formular um panorama, de forma tal que contenha informação necessária para reconstruir a *função plenóptica* não somente no ponto de captura (x,y,z) mas numa vizinhança Ω deste ponto. Para isto introduzimos o conceito de *panorama omnidirecional com múltiplas camadas*, *POMC*.

Um panorama omnidirecional *RGB-D* com múltiplas camadas, *POMC*, é definido formalmente como um conjunto de funções $\mathcal{P}_i$ de $V_i \subset \mathbb{S}^2$ para um espaço de atributos $\mathcal{A} = A_0 \times \ldots \times A_l$ (cor, profundidade, radiância, propriedades de reflexão, e outros), tais que o domínio de todas elas fazem uma cobertura da esfera unitária $\mathbb{S}^2$, e as funções $\mathcal{P}_i$ restritas ao atributo de profundidade, $\mathcal{P}_i|_d$, são contínuas.

Definimos um *POMC* $\mathcal{M}$ como uma coleção de funções sobre um conjunto de atributos $\mathcal{A} = A_0 \times \ldots \times A_l$, da seguinte maneira

$$\mathcal{M} = \{\mathcal{P}_0, \ldots, \mathcal{P}_k\} \;\; k \in \mathbb{N}, \tag{6.7}$$

onde

$$\mathcal{P}_i : V_i \subset \mathbb{S}^2 \to \mathcal{A}, \quad 0 \leq i \leq k, \tag{6.8}$$

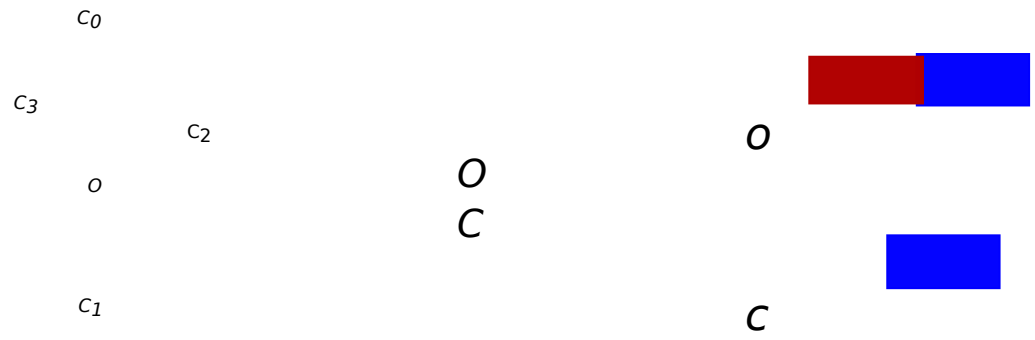
com funções $\mathcal{P}_i = (\mathcal{P}_i|_0, \ldots, \mathcal{P}_i|_l)$, que satisfazem

$$\begin{aligned} &\mathcal{P}_i|_d : V_i \subset \mathbb{S}^2 \to A_d, \quad \text{é} \quad C^0, \quad 0 \leq i \leq k, \\ &\textstyle\bigcup_{i=0}^{k} V_i = \mathbb{S}^2, \end{aligned} \tag{6.9}$$

onde A_d representará o atributo de profundidade (depth).

Observamos que no caso em que $V_i \cap V_j = \varnothing$, $\forall i \neq j$, estamos ante um *panorama omnidirecional RGB-D* como o apresentado em [23, 63]. Em um *panorama omnidirecional com múltiplas camadas* que permite reconstruir a função plenóptica em Ω em geral teremos $V_i \cap V_j \neq \varnothing$, para alguns $i \neq j$, especialmente quando V_i e V_j representem regiões contíguas na cena, desde o ponto de vista da captura do panorama.

Resgatando as equações que definem um panorama omnidirecional (6.4), temos que cada função $\mathcal{P}_i$ pode ser definida como



(a) Estrutura do *PMC*. **(b)** Ilustração de uma cena armazenada num *PMC*.

Figura 6-4: (a): Camadas C_3, C_2, C_1, C_0 ordenadas segundo a profundidade crescente desde o centro O do panorama. Em linhas pontilhadas o raio de cobertura do *POMC*, dentro do qual a *função plenóptica* pode ser reconstruída com o *POMC*. (b): Ilustração de duas vistas da cena, uma desde o centro do panorama O, e outra desde uma posição diferente C. Objetos podem ser ocluídos ou aparecer ao mudar o ponto de vista.

$$\begin{aligned} \mathcal{P}_i &= f_i \circ g_i : V_i \subset \mathbb{S}^2 \to \mathcal{A} \\ g_i &: V_i \subset \mathbb{S}^2 \to U_i \subset \mathbb{R}^2 \\ f_i &: U_i \to \mathcal{A}, \end{aligned} \tag{6.10}$$

onde $g_i = g_{|_{V_i}}$ é a função de parametrização planar da esfera restrita a V_i, enquanto que a função $f_i = (f_i|_0, \ldots, f_i|_l)$ mapeia os atributos de V_i a partir da parametrização escolhida.

Para recuperar a posição 3D a partir de uma posição $(s,t) \in U_i$ usaremos a função

$$\begin{aligned} &h_i : U_i \subset \mathbb{R}^2 \to \mathbb{R}^3; \\ &h_i(s,t) = f_i|_d(s,t) \cdot g_i^{-1}(s,t) \end{aligned} \tag{6.11}$$

onde $f_i|_d$ é a função que devolve a profundidade da camada i da cena, e g_i^{-1} a função que devolve o ponto na esfera unitária $\mathbb{S}^2$ correspondente ao parâmetro (s,t). Adicionalmente podemos aplicar uma transformação afim *LocalToWorld* de translação e rotação de maneira global ao *POMC* para posicionar e orientar o panorama no mundo.

6.2.1 Representação de panoramas omnidirecionais com múltiplas camadas

Para os efeitos teóricos pretendidos, decidimos codificar as informações do *POMC* de uma maneira simples e intuitiva, sem fazer muito caso de optimizações. No futuro pretendemos explorar o uso de formatos de imagem já estabelecidos no mercado como o *EXR* que tem

muita flexibilidade e oferece opções de compressão de dados. Nos casos pertinentes comentaremos sobre o tipo de melhoras possíveis de serem incorporados no esquema apresentado.

Escolhemos descrever as informações do panorama num arquivo *XML* que contém as informações gerais do panorama, tais como sua posição, orientação, número de camadas, e informações contidas em cada camada. Deste modo a descrição do panorama adquire uma forma semelhante a uma descrição de cena. De modo geral cada camada pode conter varias informações, sendo as mais básicas um arquivo de imagem onde está codificada a informação de cor da camada e um arquivo de imagem em ponto flutuante que contém a informação de profundidade. Adicionalmente, é possível armazenar outras informações, por camada, como por exemplo informação das direções normais, propriedades dos materiais que compõem a camada, dados de radiância para ser utilizados em iluminação, entre outros. A seguir apresentamos um exemplo de Panorama em Camadas básico, utilizado para produzir alguns dos exemplos apresentados nesta tese.

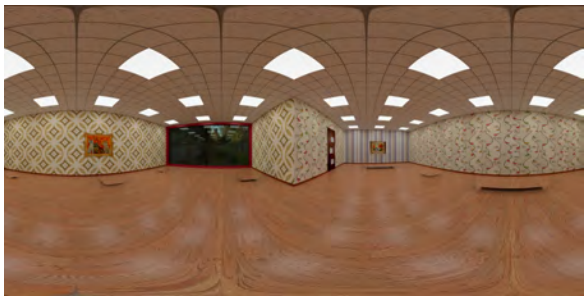
```
<?xml version="1.0" encoding="iso-8859-1"?>
<!-- Panorama: anotacoes -->
<Panorama>
    <TransformGroup> <!-- Affine transformation LocalToWorld -->
        <Translate vector='0 0 1.15'/>
        <Rotate axis='0 0 1' angle='90'/>
        <Scale vector='-1 1 1' />
    </TransformGroup>
    <Mapping type='LatLongMapping'/>
    <Freedom radius="0.5" />
    <Layer>
        <Textures mapname="Background.png" depthname="Background_d.exr" normals=""/>
    </Layer>
    <Layer>
        <Textures mapname="Foreground.png" depthname="Foreground_d.exr" normals=""/>
    </Layer>
</Panorama>
```

Embora simples, o arquivo *XML* contêm as informações necessárias para realizar uma visualização adequada desde pontos de vista dentro do raio de reconstrução do panorama. Contêm as informações de localização espacial do panorama, posição e orientação. Com estas informações e a informação de profundidade de cada camada é possível reconstruir a estrutura do ambiente se assim o desejarmos. A informação de cor contida nas camadas pode ser utilizada para gerar visualizações do panorama e realizar cálculos de iluminação quando tivermos informação de radiância em *HDR*, entre outras aplicações. No entanto devemos

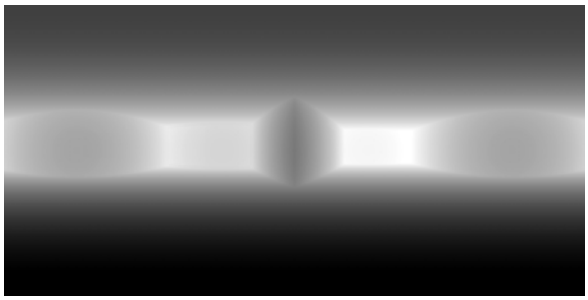
lidar com o tamanho da informação contida nas camadas. Como é esperado, dispor de uma vasta informação do ambiente ao ponto de poder estimar vistas desde novas posições de câmera, implica em um custo computacional e de armazenamento de informação. Trabalhos anteriores na área [29,40], notaram que armazenar amostras completas da função plenóptica de maneira densa pode resultar inviável ou impedir seu uso em aplicações em tempo real. A estratificação e eliminação de redundância são etapas cruciais para tornar viável o uso de uma amostragem local da função plenóptica. O panorama com camadas fornece uma representação adequada para realizar esta estratificação e simplificação da função plenóptica numa vizinhança de um ponto.

6.2.2 Atributos das camadas de um *POMC*

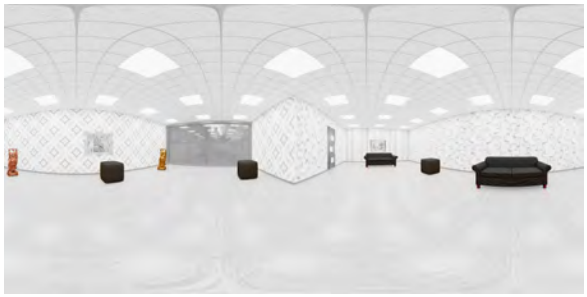
Os atributos definem a estrutura e até mesmo a segmentação das camadas do *POMC*. Entre os atributos, um de vital importância é o de profundidade, utilizado para definir as propriedades mínimas de continuidade das camadas do panorama. Podemos ter muitos atributos no *POMC*, alguns dos quais podem ser obtidos por medições durante o processo de captura ou pela análise e processamento posterior dos dados. Algumas propriedades também podem ser sintéticas, i.e., atribuídas pelo usuário do panorama. A quantidade e qualidade dos atributos



(a) Radiância da camada de fundo do *POMC*.



(b) Valor de profundidade da camada de fundo.



(c) Radiância da camada dos objetos do *POMC*.

(d) Valor de profundidade da camada de objetos.

Figura 6-5: Panorama com múltiplas camadas. O panorama tem duas camadas, a camada de fundo (a), e a camada dos objetos (c), cada uma com atributos de radiância e profundidade.

definirão os possíveis usos do *POMC*. Em muitos casos, um atributo obtido no processo de captura pode ser analisado e reestruturado em novos atributos do *POMC*, como por exemplo o caso de uma análise da radiância para inferir propriedades reflexivas dos materiais pode resultar em um novo atributo de refletância. Alguns dos atributos atribuídos às camadas podem ser:

$$Camada = \begin{cases} \text{Radiância;} \\ \text{Albedo difuso;} \\ \text{Profundidade a cena;} \\ \text{Iluminação difusa pré-calculada;} \\ \text{Índices de materiais;} \\ \text{Propriedades de reflexão;} \\ \text{Propriedades de transmitância;} \\ \text{Propriedades emissivas da camada (para separar} \\ \text{superfícies e luzes de área);} \\ \text{Sombras;} \\ \text{etc.} \end{cases} \tag{6.12}$$

Para ilustrar estas ideais, na figura (6-5) mostramos um *POMC* com duas camadas (fundo e objetos) e dois atributos básicos, radiância e profundidade.

6.2.3 Codificação da informação de uma camada

Como vimos, a informação contida em uma camada representa uma amostra completa ou incompleta da função plenóptica no ponto (x, y, z), isto é, um panorama omnidirecional completo ou parcial. Esta informação precisa ser armazenada em um arquivo digital. Uma forma de fazer isto é codificar a informação das camadas em imagens digitais. Para isto, é necessário encontrar uma parametrização adequada $g_i : V_i \subset \mathbb{S}^2 \to U_i \subset \mathbb{R}^2$ da camada, na qual o suporte planar U_i seja tal que permita uma codificação eficiente da imagem $f_i : U_i \to \mathcal{A}$ em uma digital I. Como pode ser visto na seção A.1 do apêndice A, uma condição desejada é que U_i seja um retângulo.

Existem na literatura diversos mapeamentos da esfera $\mathbb{S}^2$ sobre um suporte planar, a partir as quais podemos obter diferentes representações digitais para armazenar um panorama

omnidirecional. Nesta tese optamos por utilizar os mapeamentos equiretangular e mapa de cubo, pois entre outras coisas, eles têm suporte retangular. As fórmulas e detalhes técnicos adicionais sobre estes mapeamentos podem ser consultados no apêndice A.

Em ambos os casos, o suporte geométrico da imagem é um retângulo, uma imagem com razão de aspecto 2:1 no formato equiretangular e razão de aspecto 3:2 para o mapa de cubo. Como qualquer dos dois formatos implica em realizar cortes na esfera para planificar a mesma, isto pode induzir a que objetos de uma camada sejam divididos pelos cortes na parametrização, figura (6-13b). Isto não representa maiores problemas, mas requer alguns cuidados especiais nas implementações computacionais, como será visto ao longo dos próximos capítulos.

Tanto no formato equiretangular quanto mapa de cubo, temos que o contradomínio U da parametrização $g : \mathbb{S}^2 \to U \subset \mathbb{R}^2$ é um retângulo

$$U = [a,b] \times [c,d] = \{(x,y) \in \mathbb{R}^2 : a \leq x \leq b \quad \text{e} \quad c \leq y \leq d\}, \tag{6.13}$$

alinhado com os eixos. Desta forma, para camadas que cobrem completamente $\mathbb{S}^2$, como por exemplo uma camada de fundo, uma discretização U' de U mediante um reticulado bidimensional fornece o suporte para construir a imagem digital da camada.

Camadas que não cobrem completamente $\mathbb{S}^2$, i.e., $V_i \subsetneq \mathbb{S}^2$, podem ter um suporte planar U_i com forma arbitraria ou até mesmo varias componentes conexas, figura (6-5c). No entanto, para obter uma representação digital, podemos tomar um domínio retangular $R_i = [x_0, y_0] \times [x_1, y_1]$ que contém U_i. Logo, tomamos uma discretização R_i' de R_i com um reticulado bidimensional e construímos a imagem digital I onde assignamos valores aos pixels (x,y) que pertencem a U_i

$$I(x,y) = \begin{cases} f_i(x,y) & \text{se} \quad (x,y) \in U_i, \\ \text{valor especial} & \text{se} \quad (x,y) \notin U_i. \end{cases} \tag{6.14}$$

No caso que não seja possível codificar com um valor especial os pixels que não pertencem à discretização do domínio U_i, é possível construir uma imagem digital da função característica

$\mathcal{X}_{U_i}$ de U_i para as amostras discretas R'_i,

$$\mathcal{X}_{U_i}(x,y) = \begin{cases} 1 & \text{se} \quad (x,y) \in U_i, \\ 0 & \text{se} \quad (x,y) \notin U_i, \end{cases} \tag{6.15}$$

e usar a mesma em conjunto com $I(x,y)$ para recuperar somente os pixels do domínio U_i quando for necessário. A imagem digital da função característica costuma ser chamada de máscara, e pode ser armazenada juntamente com a imagem I num canal adicional, como por exemplo o canal alfa.

Nas parametrizações equiretangular e mapa de cubo, o conjunto imagem $g(\mathbb{S}^2) = U$ é um retângulo $U = [a,b] \times [c,d]$, de forma tal que para estas parametrizações temos fórmulas que permitem fazer a conversão entre um pixel (x,y) da imagem digital e a correspondente direção $\omega \in \mathbb{S}^2$ e vice-versa. Para que estas fórmulas sejam válidas, é importante que as coordenadas do pixel (x,y) sejam dadas em função de uma imagem digital omnidirecional completa.

Assim, a priori, podemos escolher o suporte retangular R_i de uma camada de forma que $R_i = g(\mathbb{S}^2)$, isto é, R_i é o retângulo resultante de aplicar a parametrização g à esfera $\mathbb{S}^2$ completa. Esta escolha simplifica as transformações entre pixels e direções na esfera, mas pode introduzir severos problemas de armazenamento devido a que o suporte R_i pode ser muito maior que a região U_i da camada, figura (6-5). Na seguinte seção veremos como podemos comprimir a informação, escolhendo suportes retangulares menores que o suporte máximo $R_i = g(\mathbb{S}^2)$ mantendo intactas as funções de transformação entre pixels e direções.

6.2.3.1 Compressão

Numa camada frontal, em geral se espera que a área de objetos seja inferior ao 100% do mapa. Por exemplo, os sofás, os Budas e os pufes da cena apresentada na figura (6-6a) estão na camada frontal e cobrem apenas 3% da cena completa na representação equiretangular. Ao invés de armazenar um panorama completo com muita informação irrelevante, podemos otimizar isto guardando porções retangulares de imagem que contenham os objetos da camada a ser codificada, e agrupa-los numa imagem mais compacta, como é ilustrado na figura (6-6b). Como a nossa representação não guarda informação posicional de maneira explícita, senão por meio da parametrização escolhida, é necessário guardar no cabeçalho da

(a) Camada de objetos panorama.

(b) Compressão da camada dos objetos numa imagem compacta.

Figura 6-6: (a) A camada de objetos ocupa uma pequena área na parametrização original.(b) Reorganizando os dados da camada em uma imagem compacta, é possível eliminar as áreas sem informação da parametrização original.

imagem uma lista com algumas informações dos retalhos retangulares alocados na imagem compacta, tais como o offset do retalho na imagem compacta, o offset no panorama original e o tamanho do retalho. Se pretendermos ir mais longe, é possível guardar camadas diferentes em resoluções diferentes. Isto é conveniente por exemplo em cenas onde temos um objeto de interesse com muitos detalhes na camada frontal e um background que está longe é não requer uma resolução comparável à do objeto mais próximo.

Para uma camada i cujo mapa associado seja $\mathbb{M}_i$, a versão codificada e comprimida será denotada por $\bar{M}_i$. O mapa $\bar{M}_i$ contém retalhos R_j extraídos do mapa original $\mathbb{M}_i$. Conhecendo as dimensões da cada retalho $R_j \subset \bar{M}_i$, e as coordenadas do seu vértice superior esquerdo no mapa de origem $\mathbb{M}_i$ e no mapa comprimido $\bar{M}_i$, é possível recuperar o mapa $\mathbb{M}_i$ a partir do mapa $\bar{M}_i$.

Considere um mapa comprimido $\bar{M}_i$ com k retalhos $R_j = [0,w_j] \times [0,h_j],\ j = 0,\cdots,k$. Para cada retalho R_j são conhecidos os parâmetros: $T_j = \{(u_j,v_j),(x_j,y_j),(w_j,h_j)\}$, onde

$$\begin{aligned} &(u_j,v_j) && \text{é a posição } (0,0) \text{ de } R_j \text{ no mapa de origem } \mathbb{M}_i, \\ &(x_j,y_j) && \text{é a posição } (0,0) \text{ de } R_j \text{ no mapa comprimido } \bar{M}_i, \\ &(w_j,h_j) && \text{são as dimensões de } R_j. \end{aligned} \tag{6.16}$$

A relação entre os mapas para um retalho R_j é dada por

$$\mathbb{M}(u_j+s,v_j+t) = \bar{M}_i(x_j+s,y_j+t), \quad \forall (s,t) \in [0,w_j] \times [0,h_j]. \tag{6.17}$$

Na codificação podem ser utilizados também *MIPs maps* para as camadas. Os *MIPs maps* permitem trabalhar na resolução adequada para o tipo de vista que pretendemos ge-

rar. Por exemplo, é possível que numa tela gigante seja necessário trabalhar com a camada na resolução máxima, mas em um dispositivo móvel 25% da resolução máxima pode ser suficiente.

6.2.4 Construção e classificação de camadas

Da formulação dada pela equação (6.8), é possível observar que a escolha das camadas não é única. Em geral existe um número mínimo de camadas, que dependerá das descontinuidades de profundidade da cena desde o ponto de observação escolhido como origem do panorama.

Um aspecto importante ao estruturar um panorama com camadas, será definir adequadamente as camadas, pois isto facilitará o uso posterior do panorama. Podemos pensar em vários tipos de classificadores para determinar as camadas do panorama. Uma estratégia de classificação pode ser mais ou menos relevante que outra segundo a aplicação pretendida para o panorama. A seguir apresentaremos duas estratégias de classificação para gerar camadas, a primeira guiada pela profundidade da cena e a segunda guiada pelas propriedades topológicas e geométricas.

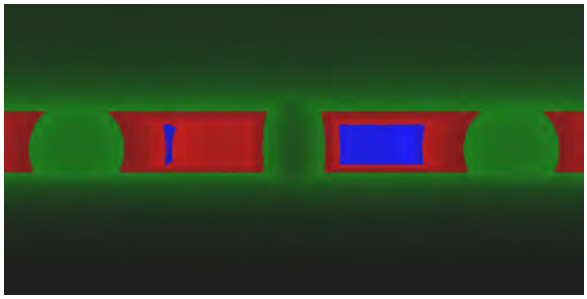
6.2.4.1 Classificação por profundidade da cena

Em certos tipos de panoramas pode ser apropriado separar as camadas segundo faixas de profundidade a partir ponto de origem do panorama. Desta forma, as camadas apresentam uma estrutura concêntrica formando fatias em certas faixas de profundidade. Uma classificação deste tipo pode ser útil quando for necessário fazer buscas em certas faixas de profundidade, pois podemos descartar facilmente camadas inteiras que estão fora da faixa considerada.

Na classificação por profundidade, uma camada i com carta de parametrização V_i deve satisfazer

$$\begin{aligned} &\mathcal{P}_i|_d : V_i \subset \mathbb{S}^2 \to \mathbb{R}, \qquad \text{is} \quad C^0, \\ &\mathcal{P}_i|_d(\omega) \in [d^i_{min}, d^i_{max}] \quad \forall \omega \in V_i, \end{aligned} \tag{6.18}$$

Como pode ser apreciado na equação (6.18), a camada deve manter a continuidade sobre o domínio de parametrização V_i e os valores de profundidade devem estar na faixa $[d^i_{min}, d^i_{max}]$ escolhida para a camada.



(a) Fundo segmentado em camadas por profundidade.

(b) Objetos segmentados em camadas por profundidade.

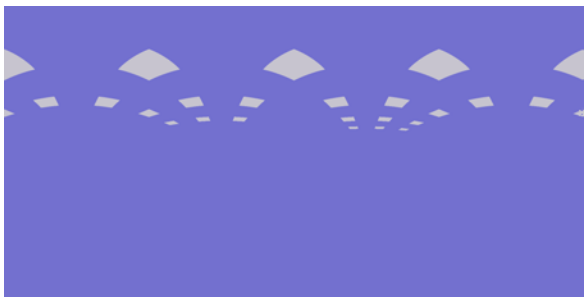
Figura 6-7: (a) A camada de fundo do panorama original da figura (6-5) deu lugar a 3 camadas obtidas por classificação de profundidade. (b) A camada de objetos da figura (6-5) foi separada em duas novas camadas pela profundidade dos objetos. As faixas de profundidade de perto para longe são: verde, vermelho e azul.

6.2.4.2 Classificação por estrutura geométrica

Esta forma de classificação está fortemente relacionada com a estrutura geométrica da cena e dos objetos que a compõem. Colocar objetos ou partes contínuas dos mesmos, desde o ponto de vista do panorama, na mesma camada pode facilitar tarefas que requerem manipulação de objetos da cena. Por exemplo, fazer aplicações de realidade aumentada, onde queremos mover um objeto, ou retirar ele da cena, pode ser simples ao ponto de movimentar uma camada ou exclui-la da renderização. Um exemplo deste tipo de classificação pode ser encontrado na figura (6-8) onde separamos o teto em uma camada, o piso em outra camada e as paredes em outra camada, mesmo que não exista descontinuidade entre todas elas.

O exemplo da figura (6-5) é uma classificação minimalista, pois conta da camada de fundo, e uma camada onde estão todos os objetos. Uma opção razoável é dispor de uma classificação primaria minimalista, que leve essencialmente em conta as descontinuidades.

(a) Camadas do teto, paredes e piso.



(b) Camada de luzes.

Figura 6-8: (a) Camada de fundo do panorama da figura (6-5), segmentada em 3 camadas por classificação geométrica. (b) Uma camada com as luzes da cena. Este tipo de camada é importante para cálculos de iluminação.

Logo, a partir deste grupo de camadas básicas, podemos idealizar novas camadas seguindo alguma estratégia em particular ou combinando elas.

A maioria dos panoramas são capturados em relação a uma base ou piso, exceto algum panorama aéreo. O piso, geralmente, uma superfície planar, também pode ser considerada separadamente. Em geral os movimentos de câmera não vão atravessar o piso, logo esta pode ser considerada naturalmente como uma camada de fundo. O piso é uma das camadas mais importantes, pois nele se apoiam muitas estruturas da cena, e a correta relação entre o posicionamento dos objetos em relação ao piso, inclusive as sombras, aportam o realismo necessário para a cena. Objeto apoiado sobre a camada do piso, mas sem sombras, parecem flutuar. Por outros cálculos errados ou simplificações inadequadas na profundidade das camadas podem fazer com que os objetos pareçam dançar sobre a superfície do piso quando mudamos o ponto de vista da câmera.

6.2.4.3 A camada do infinito

Em cenas exteriores, onde parte da cena é muito distante, é possível aplicar uma estratégia adicional. Por exemplo podemos delimitar uma camada para o céu e assumir sua profundidade como infinita. Desta forma a camada do céu se comporta como uma camada direcional para todos os pontos de vista. Esta estratégia pode incrementar o desempenho dos algoritmos de visibilidade, pois sabemos que o céu sempre será a camada de fundo, e é muito fácil renderizar esta camada, pois é tratada como um mapa direcional. Além do céu, outros objetos distantes podem ser colocados na camada do infinito, especialmente quando for determinada que sua paralaxe dentro do raio de reconstrução está abaixo de um dado limiar. Este tipo de estratégia de "achatar"o fundo é bastante utilizado em cinema 3D para aumentar o conforto visual da cena, e pode ser explorado na construção de camadas. Este tipo de camada merece um cuidado especial na sua construção. A camada do céu certamente pode ser considerada direcional, pois claramente no caso do céu toda a borda do domínio de parametrização V_i é descontínuo do resto da cena, ou seja o céu é completamente separado do resto da cena pela descontinuidade em profundidade. Não acontece o mesmo com objetos distantes que estão conectados a um meio de suporte, por exemplo árvores e montanhas apoiadas sobre o solo. Se separarmos estes elementos e os tratarmos numa camada direcional, corremos o risco de que ao gerar novas vistas da cena os objetos fiquem desalinhados com o solo. Parecera que estes objetos estão flutuando quando mudemos a posição de câmera. Uma estratégia acer-

tada neste caso é medir a distância mínima d_{dirmin} necessária para que qualquer vista dentro do raio de reconstrução da cena tenha uma paralaxe inferior a 1 pixel. Neste caso todos os elementos com profundidade maior que d_{dirmin} podem ser colocados na camada do infinito, pois sua paralaxe não se vê afetada dentro do raio de reconstrução. Outro ponto que pode ser levado em consideração é a composição da cena. Se a região de continuidade entre a camada que pretende ser colocada no infinito e as camadas com profundidade real não apresenta textura, é possível gerar a camada do infinito desde que os deslocamentos não sejam perceptíveis. Normalizar uma camada. Por exemplo demos uma camada na faixa $[d_{min}, d_{max}]$, e a é a última camada, ou somente tem a camada no infinito. Podemos compactar a camada e fazer $[d_{max}, d_{min}] \rightarrow d$. A camada não é direcional mas está simplificada a uma calota esférica de raio d. Isto permite eficiência em cálculos de ray tracing. A estratégia do achatamento de profundidade deve ser utilizada com cuidado. Achatamentos devem ser utilizados para distâncias tais que $\frac{d_{max}-d_{min}}{d_{min}} \ll 1$ e o *raio de reconstrução* $\ll d_{min}$ de modo que as distorções provocadas não sejam perceptíveis. As distorções resultam melhor camufladas em cenas naturais, onde não abundam linhas retas e construções regulares que podem ser usados para comparação.

6.2.4.4 Ordenação das camadas

Como pode ser observado nos exemplos anteriores, as estratégias de classificação de camadas não garantem por si só que as camadas geradas possam ser ordenadas mantendo algum critério de visibilidade. Dependendo da estratégia utilizada durante o processo de seleção de camadas, é possível garantir uma ordenação parcial, como a do algoritmo do pintor, de maneira tal que desde o ponto de vista da origem do panorama, a visualização da cena seja a correta renderizando sequencialmente as camadas numa ordem dada. Uma ordem deste tipo é parcial e pode mudar quando nos deslocarmos da origem do *POMC*.

6.3 Construção de panoramas em camadas

Uma pergunta pertinente é como construir ou capturar um panorama em camadas. Esta pergunta por si só abre uma nova série de problemas de pesquisa interessantes. Os panoramas em camadas utilizados nos próximos capítulos foram construídos num software de modelagem, porém acreditamos que a captura deste tipo de informação não será problema no futuro

próximo se olharmos para os novos dispositivos que vem sendo desenvolvidos ultimamente, tais como o *Project Tango*, *Structure Sensor*, *Google Jump*, *Lytro Immerge*, entre outros (figura (6-9)).

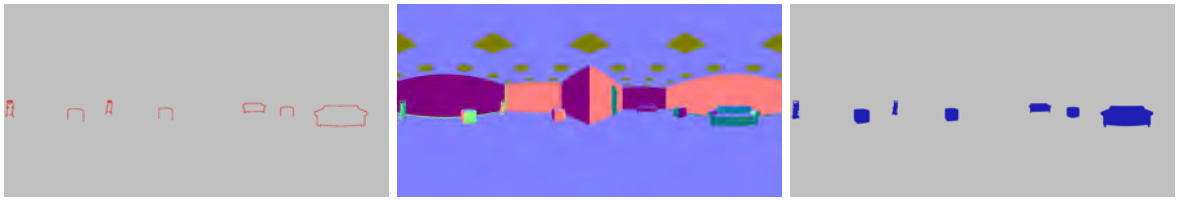
Figura 6-9: Equipamentos de captura disponíveis no mercado. De esquerda à direita: Google Jump, Ladybug (captura RGB), Project Tango, e Structure Sensor.

6.3.1 Construção por incremento de um panorama RGB-D

Nossa primeira proposta é construir um panorama em camadas a partir de um panorama *RGBD* capturado ou modelado por computador. Dado que as camadas são delimitadas parcial ou totalmente por regiões de descontinuidade na profundidade, propomos como estratégia para definir as camadas, detectar as curvas de descontinuidade de profundidade (figura (6-10a)), para logo unir os extremos das mesmas a fim de formar regiões fechadas. Esta união pode ser mediante uma linha reta ou guiada por uma análise de descontinuidades nas normais na camada de profundidade (figura (6-10b)), dado que geralmente a transição de um objeto para outro tem descontinuidades de profundidade ou de normais. Uma vez calculadas as camadas com esta estratégia, resta preencher as regiões de oclusão geradas pelas camadas da frente nas camadas mais profundas. Isto pode ser feito de maneira satisfatória utilizando síntese de textura, [22,41], ou inpaiting, [14]. Em muitos casos remover um objeto grande de uma imagem e fazer inpainting da região pode gerar resultados pouco convincentes, porém no nosso caso isto não é problemático pois estamos interessados somente em áreas próximas das bordas das regiões geradas pela separação de camadas, e nesse caso muitas técnicas de inpainting ou mesmo síntese de textura apresentam resultados aceitáveis e realistas. Finalmente, a profundidade nas regiões de desoclusão também pode ser estimada com inpainting.

6.3.2 Construção por renderização direta

Para panoramas em camadas gerados inteiramente por computador temos uma boa flexibilidade e as soluções iniciais podem ser bastante diretas. Escolhido o ponto de origem do



(a) **(b)** **(c)**

Figura 6-10: (a) Mapa das descontinuidades no atributo de profundidade. (b) Mapa de normais. (c) Máscara da camada C_1 calculada usando o mapa de descontinuidade de profundidade e gradiente do mapa de normais.

panorama na cena, o renderizador lança raios em todas as direções. Os cálculos de iluminação direta e contribuições difusas são realizados em todas as interseções dos raios ao longo da cena. Por exemplo se um raio ω_i intersecta a cena 3 vezes, os cálculos são efetuados 3 vezes e alocados em 3 camadas diferentes. Para saber em qual camada será colocada cada amostra renderizada, o processo de renderização acumula inicialmente um grande número de amostras num buffer de amostras. Posteriormente este buffer é processado e são geradas as camadas iniciais. Cada nova amostra a ser renderizada é comparada com as camadas que já foram geradas, e se estiver dentro de um nível de tolerância de erro para alguma das camadas será alocado nesta camada, caso contrário, é colocada no buffer e aguardará até acumular um número suficiente de amostras para determinar novas camadas ou dispor de informação para completar as antigas. O panorama da figura (6-5) foi renderizado nas duas camadas apresentadas. Todos os cálculos e usos posteriores são realizados diretamente sobre o panorama, sem levar em consideração o modelo 3D original.

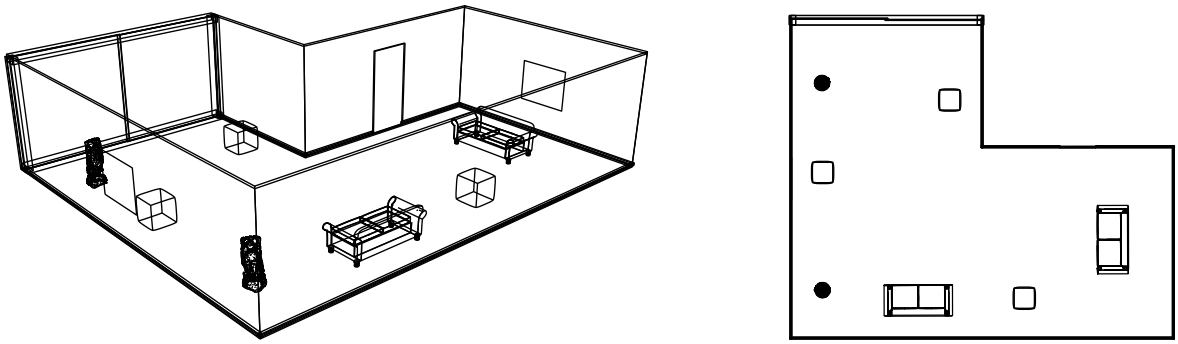
6.3.3 Construção por captura com dispositivos RGB-D

No mercado existe uma ampla variedade de dispositivos de captura *RGB-D* com sensores de localização integrados, que facilitam as tarefas de posicionamento e calibração. Suponhamos que utilizando algum destes dispositivos (*Project Tango*, *Structure Sensor*, *Kinect*), obtemos uma nuvem de pontos no mundo, que pode ser transformada para um par (ω, d), onde ω é a direção em que a amostra se encontra em relação à origem e orientação escolhida para o panorama, e d é a correspondente distância. Dispondo de uma nuvem densa de amostras, é possível para uma resolução adequada, reconstruir as camadas para o panorama, na resolução permitida pela densidade e amostras. O processo não é simples, mas existe uma vasta quantidade de trabalhos na área de escaneamento e reconstrução de malhas que podem ser

utilizados para fazer o alinhamento das amostras para garantir a continuidade das camadas.

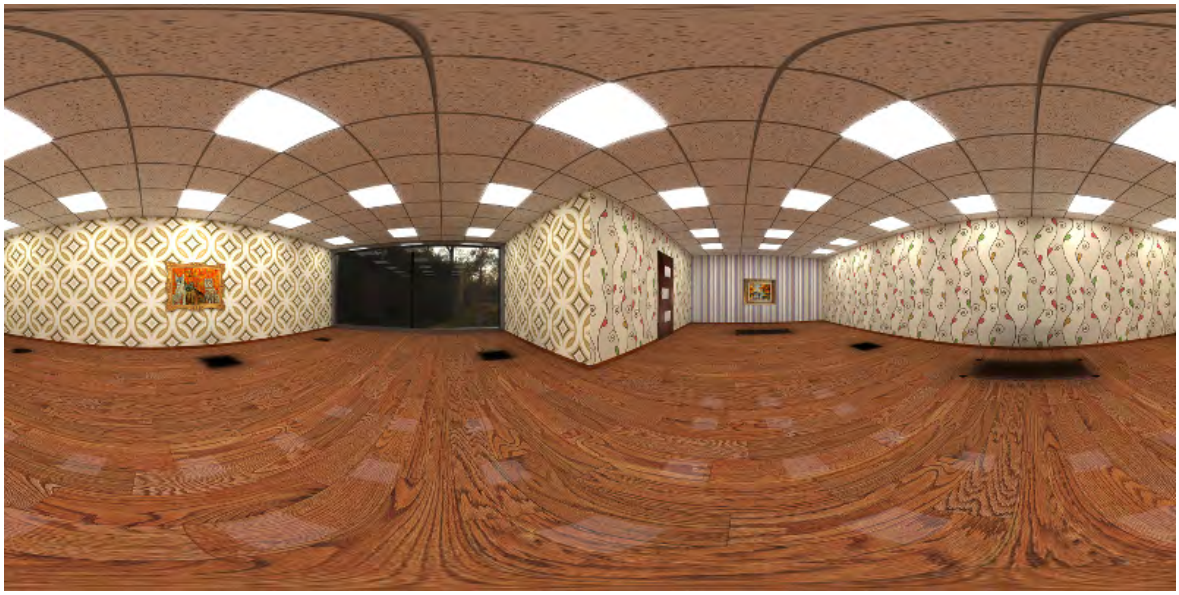
6.4 Considerações

Câmeras com capacidade de capturar o light field, como a *Lytro Immerge*, podem aportar dados altamente redundantes, mas estruturados de tal forma que sua análise e pré-processamento podem permitir a extração de panoramas em camadas com muitas propriedades de interesse. Por exemplo, dado que para cada posição visível do mundo, vários raios são capturados, uma análise da mudança de radiância nos mesmos pode fornecer informação sobre as propriedades de espalhamento da luz nas superfícies, ou seja, podemos aproximar as brdfs, e separar o mapa em albedo difuso e mapa de especularidade. Medindo a variação angular dos raios provenientes de cada amostra é possível extrair informação da estrutura 3D da cena. Câmeras como a *Lytro Immerge* estão sendo pensadas e desenvolvidas para prover soluções para o setor da indústria do cinema. As nossa proposta tem um alcance diferente, pensando em opções para os usuários e provedores de conteúdo panorâmico.

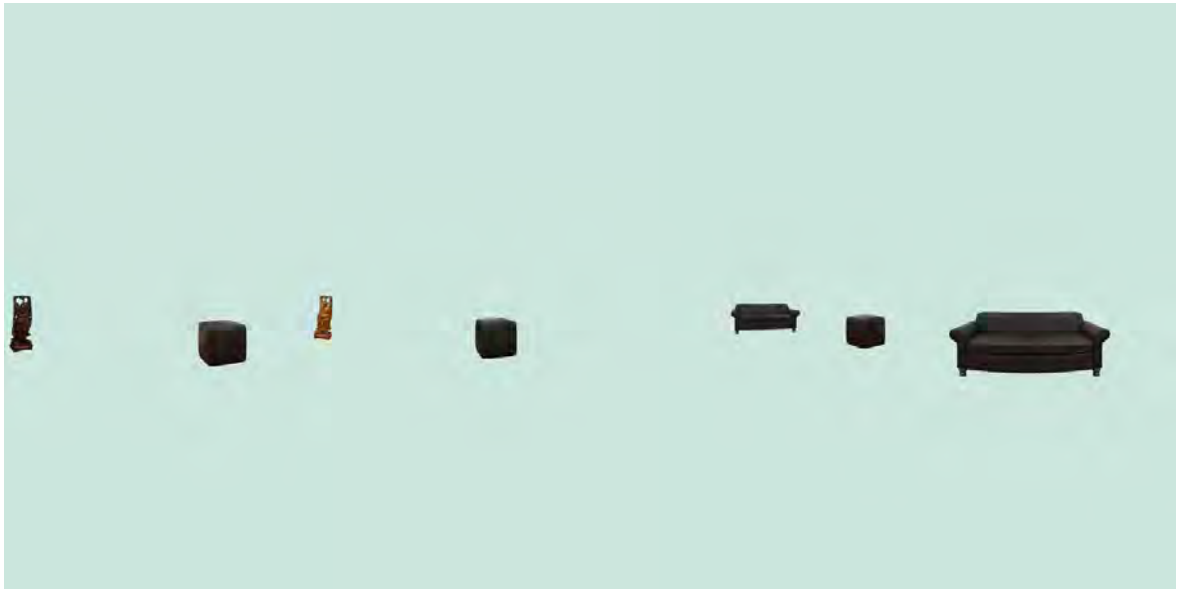


(a) Vista perspectiva. **(b)** Vista ortográfica superior.

Figura 6-11: Renderização esquemática do ambiente utilizado para construir o *POMC*.



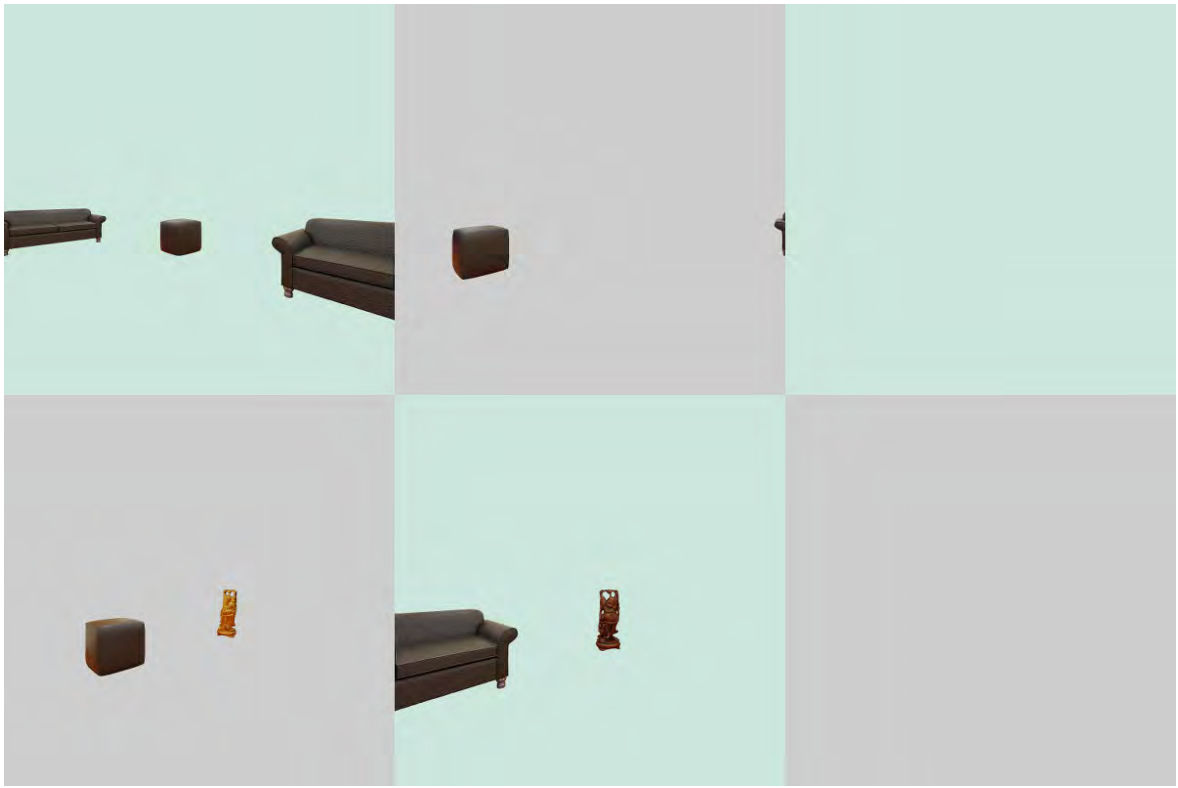
(a) Camada de fundo do *POMC*. Somente atributo de radiância.



(b) Camada de objetos do *POMC*. Somente atributo de radiância.

Figura 6-12: Parametrização equiretangular do *POMC*.

(a) Camada de fundo do *POMC*. Somente a informação de radiância.



(b) Camada de objetos do *POMC*. Somente a informação de radiância.

Figura 6-13: Parametrização em mapeamento de cubo do *POMC*. Ordem das faces do mapa de cubo: as 3 faces superiores correspondem aos eixos $+X$, $+Y$ e $+Z$. A 3 faces inferiores correspondem aos eixos $-X$, $-Y$ e $-Z$.

Capítulo 7

Cálculo de superfícies visíveis e visualização para *POMC*

Dada uma cena formada por um conjunto de objetos 3D e uma especificação de visualização, estamos interessados em determinar quais partes das superfícies dos objetos são visíveis desde o centro de projeção especificado. Este processo é conhecido com o nome de determinação de superfícies visíveis. Se bem a formulação do problema possa parecer simples, na prática, sua implementação requer um esforço computacional importante. Esforços estes, que motivaram o desenvolvimento de variados algoritmos de visibilidade e também de arquiteturas específicas para resolver este problema.

Neste capítulo estudaremos o problema de visibilidade no contexto de *Panoramas Omnidirecionais com Múltiplas Camadas*. Mesmo tendo um *POMC* ordenado e com visibilidade resolvida para o ponto de captura do mesmo, os movimentos de câmera necessários para gerar novas vistas podem gerar oclusões e desoclusões na cena que devem ser devidamente tratadas para produzir uma visualização correta. Antes de continuar com o estudo de algoritmos de visibilidade para *POMC*, apresentaremos brevemente o problema de determinação de superfícies visíveis tradicional e os algoritmos clássicos.

7.1 Cálculo de Superfícies Visíveis

Essencialmente existem duas abordagens no problema de determinação de superfícies visíveis. A primeira delas, algoritmo 5, determina qual dos objetos da cena é visível em cada pixel da imagem. É fácil ver que numa abordagem por força bruta, para uma imagem de p

pixels e uma cena com n objetos, a complexidade é proporcional a np.

Algorithm 5 Visibilidade baseada na imagem

```
1: for cada pixel na imagem do
2:     Determinar o objeto mais próximo ao observador que é visível através do pixel;
3:     Pintar o pixel com a cor do objeto visível;
4: end for
```

A segunda abordagem, algoritmo 6, consiste em comparar cada objeto com os restantes, eliminando objetos e partes de objetos não visíveis. Comparando cada objeto com todos os outros teremos uma complexidade proporcional a n^2. Parece bom para $n < p$, mas tem que levar em conta que a comparação entre objetos pode ser computacionalmente cara.

Algorithm 6 Visibilidade baseada na cena

```
1: for cada objeto na cena do
2:     Determinar quais partes do objeto não são obstruídas por outros objetos ou outras
       partes do próprio objeto;
3:     Pintar as partes na cor apropriada;
4: end for
```

Na literatura estas duas abordagens são conhecidas como *algoritmos na precisão de imagem* e *algoritmos na precisão de objetos*. Os algoritmos em precisão de objeto foram desenvolvidos originalmente para sistemas gráficos vetoriais. Eles trabalham com as representações dos objetos e calculam a solução exatamente. Devolvem como resultado uma lista ordenada das porções de superfícies projetadas na tela virtual. Em contraste os algoritmos em precisão de imagem foram desenvolvidos para sistemas de rasterização. Estes algoritmos se concentram em resolver o problema para um nível de resolução, não necessariamente igual ao da imagem final. Estes algoritmos tentam resolver o problema para cada pixel, analisando as profundidades relativas ao longo do raio de visão que passa pelo pixel.

Nesta tese vamos nos concentrar em algoritmos na precisão de imagem, e dentro deste grupo em dois algoritmos amplamente difundidos: Z-Buffer e Traçado de Raios.

7.1.1 Z-Buffer

O *Z-Buffer* é um algoritmo em precisão da imagem, desenvolvido por Catmull [10], e trata-se de um dos algoritmos de determinação de superfícies visíveis mais simples de serem implementados, tanto em software quanto em hardware. Este algoritmo está implementado nativamente no firmware de todas as placas gráficas da atualidade. O *Z-Buffer* armazena,

para cada pixel, a distância até a superfície mais próxima naquele ponto da imagem, que é a superfície visível. A estrutura de dados interna armazena a informação de profundidade para cada pixel. O algoritmo trata um objeto por vez e o opera sobre os pixels de sua projeção.

O algoritmo requer um *frame buffer* F para armazenar as cores e um *Z-buffer* para armazenar os valores de z de cada pixel. No sistema de câmara inicializamos o *Z-buffer* com valores MAX_FLOAT (infinito ou a distância do plano far) e o *F-buffer* com a cor de fundo. O menor valor z que pode ser armazenado é $z = n$ (plano near), ou n/f no volume normalizado. Os polígonos são processados em qualquer ordem no frame buffer. Durante o processo de rasterização (conversão para o frame buffer), ao processar o ponto (x,y), se o mesmo tem valor z menor que o valor atual armazenado na posição (x,y) do *Z-buffer*, os valores dos buffers Z e F são atualizados com os valores de cor e profundidade do ponto atual.

Não é necessário realizar pré-ordenação nem comparações objeto a objeto. O processo se resume a fazer uma comparação sobre $Z_i(x,y); F_i(x,y)$ para (x,y) fixos para encontrar o menor valor de Z_i. O Z buffer e o Frame buffer guardam a informação associada com o menor valor de z encontrado para cada (x,y) até o momento. Assim os polígonos aparecem na tela na ordem na qual estão sendo processados. Cada polígono é processado de a uma scanline por vez nos buffers.

Como o tamanho Δx e Δy dos pixels da imagem são constantes, é possível simplificar o cálculo de z explorando o fato de que os objetos são compostos por polígonos planares. Se o polígono é planar, para uma posição (x,y) calculamos z resolvendo uma equação linear do

Algorithm 7 Z-Buffer

```
1:  for y ← 0, YMAX do
2:      for x ← 0XMAX do
3:          I(x,y) ← BACKGROUND;
4:          Z(x,y) ← MAX_VAL;
5:      end for
6:  end for
7:  for cada polígono do
8:      for cada pixel na projeção do polígono do
9:          pz ← valor z do polígono no pixel (x,y);
10:         if pz < Z(x,y) then
11:             Z(x,y) ← pz;
12:             I(x,y) ← cor do polígono no pixel (x,y);
13:         end if
14:     end for
15: end for
```

tipo $Ax+Bx+Cx+D=0$ na variável z:

$$z = \frac{-D-Ax-By}{C}. \tag{7.1}$$

Se no ponto (x,y) temos $z=z_1$, no ponto $(x+\Delta x,y)$ o valor de z é dado por

$$z = \frac{-D-A(x+\Delta x)-By}{C} = \frac{-D-Ax-By}{C} - \frac{A\Delta x}{C} = z_1 - \frac{A\Delta x}{C}. \tag{7.2}$$

Da mesma forma em $(x,y+\Delta y)$ temos $z = z_1 - \frac{B\Delta y}{C}$.

O algoritmo faz uma comparação por pixel para cada objeto que contém esse pixel na sua projeção na imagem. O tempo do cálculo de visibilidade tende a ser independente do número de polígonos em um objeto, pois na media, o número de pixels cobertos por cada polígono decresce na magnitude com que cresce o numero de polígonos no objeto. Mas temos que levar em contas que se tratando de polígonos planos, quanto menor é a quantidade de polígonos da cena, mais rápido será o algoritmo, devido a que é possível utilizar os cálculos simples da equação (7.2), que acelera o tempo utilizado para determinar o valor de z.

Devido a que o algoritmo de Z-Buffer opera na precisão de imagem, ele apresenta os efeitos de aliasing. O algoritmo de A-Buffer introduzido por Carpenter [9], resolve este tipo de problemas utilizando uma aproximação discreta para amostragem de área não ponderada.

7.1.2 Ray tracing

O algoritmo de *ray tracing* determina a visibilidade de superfícies traçando um raio de luz imaginário desde a posição do observador em direção à cena. Trata-se de um algoritmo de precisão de imagem, devido a que os raios são traçados desde a posição da câmara, passando pelo centro dos pixels da imagem. São escolhidos um centro de projeção (câmera) e uma janela no plano de projeção (tela virtual). A janela é dividida numa grade regular, que corresponde aos pixels da resolução desejada. Logo, para cada pixel (célula da grade) na janela, é disparado um raio desde o centro de projeção O através do centro do pixel. É calculado o ponto de interseção deste raio com a cena, e se dita interseção existe, a cor do pixel é a do objeto da interseção mais próxima. O algoritmo de *ray tracing* foi desenvolvido inicialmente por Appel [4] e Goldstein e Nagel [28, 33]. Appel foi o primeiro a traçar raios para calcular sombras, enquanto que Goldstein e Nagel foram pioneiros no seu uso no cálculo

Algorithm 8 Ray tracing

```
1: Escolher o centro de projeção e a janela na tela virtual;
2: for para cada scanline na imagem do
3:     for para cada pixel do scanline do
4:         Determine o raio desde o centro de projeção através do pixel;
5:         for cada objeto na cena do
6:             if o objeto é intersectado e está mais pero que a interseção anterior then
7:                 Salvar os dados da interseção e do objeto;
8:             end if
9:         end for
10:        Atribuir ao pixel a cor do objeto que foi intersectado mais perto ;
11:    end for
12: end for
```

de operações booleanas. Em 1980, Whitted [57] e Kay [36] estenderam o algoritmo de *ray tracing* para permitir o calculo de reflexões especulares e refração. Estes efeitos adicionais serão explorados na seção de iluminação.

7.2 Visualização de Panoramas Omnidirecionais com Múltiplas Camadas

Dado que o *POMC* foi proposto para armazenar informação codificada da *função plenóptica*, uma das aplicações naturais de um *POMC* é a visualização de amostras da *função plenóptica* desde um ponto de vista próximo ao centro do *POMC*, isto é, gerar novas vistas da cena armazenada no panorama com camadas.

Pensamos em duas abordagens para realizar a visualização parcial de um *POMC*: renderização baseada em amostras de pontos e renderização baseada em malhas. A seguir vamos descrever as duas abordagens e comparar os resultados obtidos com elas.

7.3 Renderização baseada em amostras de pontos

Este processo de renderização consiste, fundamentalmente, em escolher amostras 2D nas imagens panorâmicas que compõem as camadas do *POMC*, calcular a correspondente posição 3D das mesmas, e reprojetá-las novamente no plano da imagem da câmera de visualização escolhida. Após obter as amostras resultantes no plano da imagem de saída, resolvemos o problema de visibilidade (oclusão) descartando as amostras que não são visíveis desde a

posição de visualização escolhida, e finalmente, filtramos as amostras resultantes para obter a imagem final.

O processo de renderização baseado em amostras de pontos pode ser dividido nas seguintes etapas:

- escolher amostras nas camadas do *POMC*;
- calcular a posição 3D (no mundo) das amostras e reprojetá-las no plano da imagem;
- resolver o problema de visibilidade para as amostras geradas;
- realizar uma filtragem das amostras visíveis resultantes;
- salvar a imagem de filtrada;

Segundo veremos nos algoritmos propostos a seguir, algumas das etapas podem ser realizadas de maneira recursiva mais de uma vez, como por exemplo uma seleção de amostras iniciais seguidas de um processo de refinamento adaptativo da amostragem.

Devido a que a informação contida no *POMC* representa parte de uma cena 3D, ao mudarmos a posição desde a qual visualizamos a cena, podem ocorrer vários problemas de oclusão/desoclusão. Um dos problemas comuns é a visualização de áreas que não estão armazenadas no *POMC*. Isto pode ser prevenido colocando um raio de deslocamento r, o *raio de reconstrução* do panorama, a fim de garantir que os movimentos de câmara dentro desse raio permitam uma reconstrução da *função plenóptica* e deste modo gerar uma visualização sem buracos. Quando nos movemos dentro da cena, vamos nos aproximar de alguns objetos e nos afastar de outros, também vai mudar o ângulo com que vamos visualizar os objetos, isto pode introduzir muitos problemas de visualização. Ao mudarmos o ponto de vista, na reprojeção da cena observaremos que algumas áreas estão se agrupando e outras se afastando (figura (7-1)). Ao trabalharmos com amostras pontuais isto significa que os pontos projetados podem se juntar ou afastar na imagem resultante (figura (7-4)). Pode acontecer que algumas áreas estejam muito populadas e outras tenham muita dispersão de amostras. Isto é problemático na hora de resolver uma possível oclusão e realizar a filtragem.

(a) Câmara na origem do panorama. **(b)** Câmara deslocada da origem do panorama.

Figura 7-1: Efeito do movimento de câmara. Como resultado da mudança do ponto de vista, algumas regiões se contraem e outras se expandem. Na imagem (b) a área da porta é maior porque a mesma está sendo visualizada desde um ângulo mais frontal. A parede com texturas circulares é vista desde um ângulo mais lateral, provocando uma contração na imagem.

7.3.1 Amostragem direta com refinamento nas descontinuidades de profundidade

Este algoritmo realiza uma amostragem pontual regular das camadas do *POMC*. Após a amostragem inicial, é realizado um processo de classificação de amostras visíveis no qual pixels que se encontram próximos às descontinuidades de profundidade são amostrados novamente. Assim, existe um refinamento localizado da amostragem, a fim de resolver o problema de visibilidade ao tempo que se obtém um efeito de antialiasing. O algoritmo pode ser descrito na seguinte sequencia de passos:

1. selecionar amostras em cada camada do *POMC*;
2. calcular a posição no mundo das amostras selecionadas;
3. projetar as amostras no plano da imagem da câmera;
4. agrupar amostras em blocos 3×3 pixels e resolver a oclusão;
5. descartar as amostras que não são visíveis em cada bloco;
6. refinar a amostragem nos blocos com problemas de oclusão;
7. resolver a oclusão pixel a pixel nos blocos refinados;
8. realizar uma filtragem das amostras visíveis resultantes;

O processo de classificação das amostras visíveis e não visíveis é feita por etapas. Ini-

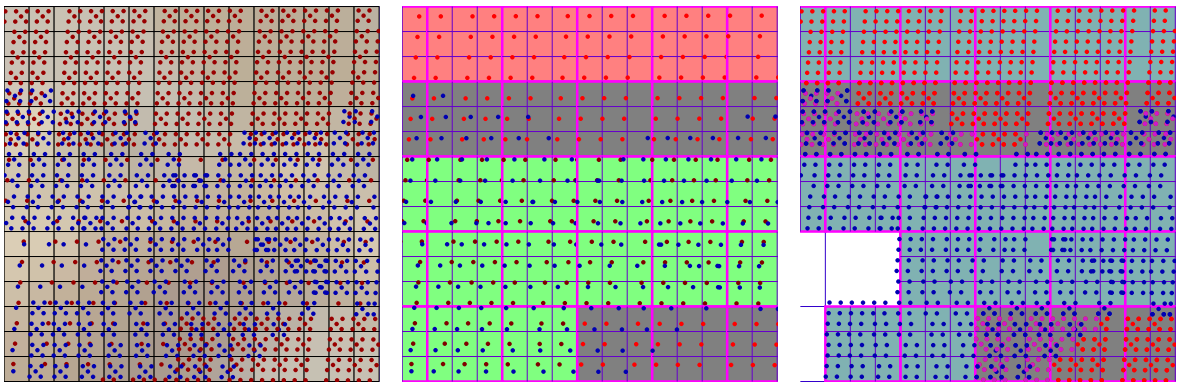
cialmente, o suporte da imagem é dividido em blocos de 3×3 pixels (passo 4). Para cada bloco I identificamos as amostras localizadas nele. Seguidamente ordenamos as amostras do bloco I segundo a camada e segundo a profundidade. Se todas as amostras do bloco I pertencem à mesma camada, são marcadas como visíveis. Se o bloco I tem amostras em mais de uma camada, fazemos uma comparação com os blocos da sua vizinhança 8-conectada. Para cada bloco vizinho consideramos a amostra com menos profundidade. Se as 8 (ou menos) amostras com menor profundidade dos blocos vizinhos pertencem à mesma camada que a amostra com menor profundidade do bloco I, significa que estamos na região interior da camada superior e não na borda da mesma (blocos verdes na figura (7-3)). Assim, marcamos as amostras da camada superior de I como visíveis e as amostras das demais camadas como não visíveis (passo 5).

Possivelmente teremos uma série de blocos que não podem ser classificados porque cada um deles tem amostras em varias camadas e suas vizinhanças 8-conectadas não têm suas amostras de menor profundidade na mesma camada que o bloco central (figura (7-3)). Estes blocos estão localizados nas bordas das camadas. Para resolver o problema de visibilidade refinamos a amostragem destes blocos e sua vizinhança 8-conectada e processamos novamente todos os blocos com amostragem refinada usando o critério do parágrafo anterior (passo 6). Durante o refinamento alguns blocos não classificados podem ter sido resolvidos e blocos que pareciam classificados podem mudar seu status para não resolvidos, devido ao surgimento das novas amostras. Como a densidade a amostras é maior, a seguir fazemos uma classificação das amostras pixel a pixel em cada bloco não resolvido para determinar as amostras visíveis e não visíveis (passo 7). Por fim, é feita uma filtragem com as amostras visíveis resultantes do processo de seleção anterior (passo 8).

7.3.1.1 Filtragem de pontos

Atualmente realizamos uma filtragem espacialmente invariante ao longo de todo o domínio da imagem, (algoritmo 9). Em certos casos isto não é recomendado, pois as amostras resultantes no processo de visualização se movem de maneira que existem regiões em expansão e outras em contração (figura (7-4)). Assim sendo, um filtro invariante pode não cobrir pixels da imagem a ser reconstruída em regiões com poucas amostras, ou pode suavizar a imagem reconstruída em regiões com alta concentração de amostras. O uso de filtros espacialmente variantes já foi estudado na área de *warping* de imagens por Heckbert [32], e é recomendável

Figura 7-2: Visualização de frustum limitado do panorama da figura (6-5), com posição de câmara fora do centro do panorama. A região azul de 15×15 pixels destacada na imagem é analisada na figura (7-3).



(a) Amostras reprojetadas. **(b)** Classificação de blocos. **(c)** Refinamento da amostragem.

Figura 7-3: Filtragem da figura (7-2). (a) Amostras da camada de fundo (vermelho), e da camada de objetos (azul). (b) Classificação dos blocos 3×3 (magenta). Blocos com todas as amostras na camada de fundo (rosa), e blocos com vizinhança 8-conectada na mesma camada superior (verde) são resolvidos. Em cinza blocos que precisam refinamento da amostragem. (c) Refinamento da amostragem nos blocos cinza e suas vizinhanças 8-conectada.

incorporar estas ideias no processo de renderização para lograr resultados superiores.

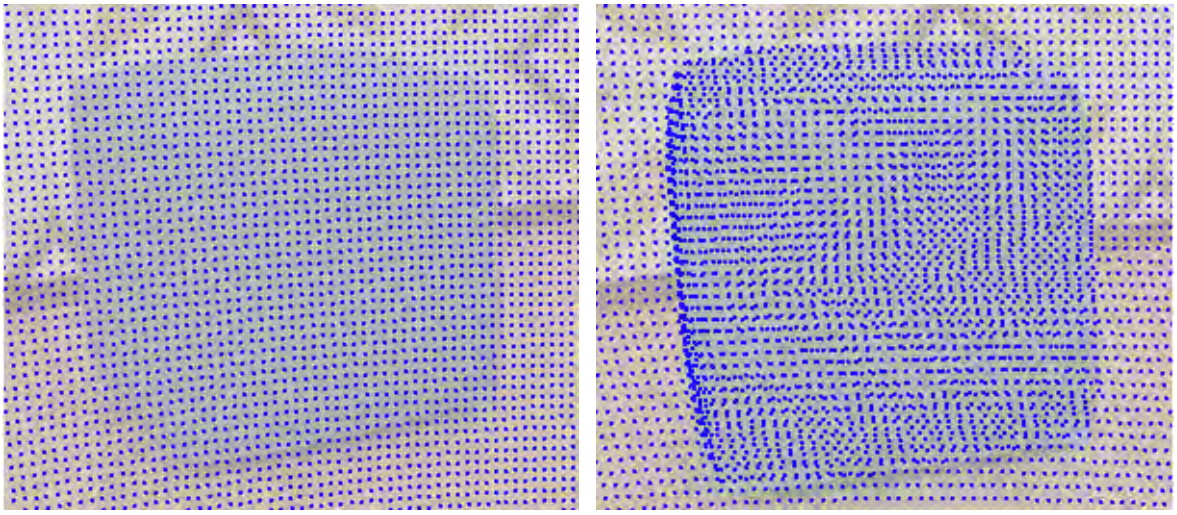
É necessário fazer um esclarecimento acerca deste algoritmo. Fixamos blocos de trabalho de 3×3 pixels para realizar a determinação de visibilidade das camadas. Levamos em consideração de que para que o algoritmo seja válido é necessário garantir que as amostras geradas pela projeção do panorama original no novo ponto de vista na resolução final garanta uma distribuição adequada de amostras para qualquer posição de câmera dentro do raio de

reconstrução do *POMC*. Este fato também é afetado pela resolução da imagem de saída e o fov da câmera. Por exemplo, se as camadas do *POMC* são representadas no formato equiretangular com uma resolução de $2N \times N$ pixels, é recomendável uma imagem de saída com resolução não maior a $fov/360 \cdot 2N$ pixels de resolução horizontal/vertical, onde fov é o ângulo de visão horizontal/vertical da câmera para evitar perdas de qualidade.

Como foi possível observar na figura (7-2), o algoritmo anterior apresenta bons resulta-

Algorithm 9 filtragem de amostras válidas

```
1:  Input: array L de amostras válidas, e raio r do filtro;
2:  Output: imagem filtrada destImg;
3:  for δ ∈ L do
4:      (u,v,c) ← posição (x,y) e camada k de δ;
5:      for x ← u − r to u + r do
6:          for y ← v − r to v + r do
7:              w ← EvalFilter(x − u, y − v) ;
8:              destImg(x,y) ← destImg(x,y) + w ∗ PanoramaColor(u,v,c) ;
9:              weights(x,y) ← weights(x,y) + w ;
10:         end for
11:     end for
12: end for
13: for x ← 0 to W do
14:     for y ← 0 to H do
15:         destImg(x,y) ← destImg(x,y)/weights(x,y);
16:     end for
17: end for
18: return destImg;
```



(a) **(b)**

Figura 7-4: Distribuição de amostras na imagem final em uma visualização com amostragem regular do panorama: (a) câmera na origem do panorama, (b) a câmera encontra-se deslocada do centro do panorama, fazendo com que algumas amostras se contraiam (lateral do Puff).

dos mas está ligado a fatores de resolução do panorama de das visualizações. Para contornar estes e outros problemas desenvolvemos outro algoritmo baseado em amostras que utiliza um processo adaptativo, e será apresentado a seguir.

7.3.2 Amostragem adaptada à vizinhança 8-conectada

Este algoritmo é do tipo adaptativo, pois faz uma primeira amostragem do *POMC*, analisa a projeção das amostras e com isto determina que regiões do *POMC* devem ter um refinamento na amostragem e quantas amostras devem ser selecionadas em cada pixel. Este algoritmo explora o fato de que as camadas do *PMC* são contínuas no atributo de profundidade, fato que permite explorar a conectividade das amostras de maneira implícita na hora de determinar o nível de refinamento necessário para conseguir uma boa distribuição de amostras no plano da imagem de saída.

Neste tipo de algoritmo a intenção é conseguir uma distribuição de amostras relativamente regular no plano de projeção da imagem de saída, o que pode implicar em realizar uma amostragem irregular nas camadas do *POMC*. A sequência de passos do algoritmo é a seguinte:

1. selecionar amostras em cada pixel de cada camada do *POMC*;
2. calcular a posição no mundo das amostras selecionadas;
3. descartar amostras que se encontram fora do frustum da câmera;
4. projetar as amostras no plano da imagem da câmera;
5. armazenar as posições das amostras resultantes;
6. para cada amostra resultante, medir o afastamento máximo em x e y das amostras da sua vizinhança 8-conectada;
7. subamostrar o pixel em x e y segundo o afastamento determinado;
8. descartar as amostras que não são visíveis;
9. realizar uma filtragem das amostras visíveis;

Como já foi comentado anteriormente, as camadas são consideradas localmente con-

tínuas. Portanto a superfície interpolada pelos pixels de uma vizinhança 8-conectada de amostras é considerada contínua. Utilizamos uma interpolação bilinear dos valores de profundidade armazenados nos pixels da camada para determinar a posição das amostras.

Podemos melhorar os resultados do método introduzindo um refinamento maior nos pixels cujas vizinhanças 8-conectadas apresentam descontinuidade na profundidade (a descontinuidade pode ser determinada através de um limiar de profundidade). Assim, resolvendo a visibilidade no nível de subpixel podemos diminuir o cisalhamento nas bordas das camadas.

A figura (7-5) ilustra os passos 1 a 6 no algoritmo, onde os pixels da camada são repro-

Algorithm 10 Amostragem adaptada à vizinhança 8-conectada

Input: panorama em camadas A;
Output: imagem I de resolução $w \times h$;
Criar uma lista vazia L;
Criar um buffer de amostras S;
for camada $c_i \in$ panorama A **do** ▷ amostragem inicial
    Adicionar um mapa b_i com a resolução de c_i na lista L;
    for pixel $(x,y) \in c_i$ **do**
        Calcular a posição de reprojeção $p = (x_p, y_p)$;
        $b_i(x,y) \leftarrow p$; ▷ guarda a posição de reprojeção de uma amostra $(x,y) \in c_i$
        Adicionar p ao buffer de amostras S;
    end for
end for
for $b_i \in L$ **do** ▷ cálculo de amostras extras
    for pixel $(x,y) \in b_i$ **do**
        $m_h \leftarrow$ maior distancia horizontal na vizinhança 8-conectada de (x,y);
        $m_v \leftarrow$ maior distancia vertical na vizinhança 8-conectada de (x,y);
        guardar um mapa $(m_h, m_v)_i$;
    end for
end for
for camada $c_i \in$ panorama A **do** ▷ amostragem adaptada
    for pixel $(x,y) \in c_i$ **do**
        reamostrar pixel (x,y) com $m_h \cdot m_v$ amostras;
        adicionar as $m_h \cdot m_h$ novas amostras (x_p, y_p) no buffer S;
    end for
end for ▷ Determinação da visibilidade pixel a pixel
Criar um buffer de visibilidade Z de resolução $w \times h$;
for amostra $k \in S$ **do**
    $p \leftarrow position(k)$; ▷ p é a posição de reprojeção de k
    if $depth(k) < depth(Z(p)$ **then**
        $Z(p) \leftarrow k$;
    end if
end for
Filtrar amostras válidas: $I \leftarrow filter(Z)$;
return I;

jetados na nova imagem. Para cada amostra (x,y) fazemos a medição do espalhamento da vizinhança 8-conectada. Determinamos o afastamento vertical dy e horizontal dx máximo das amostras vizinhas. Os valores dy e dx indicaram o refinamento de subamostragem que devemos realizar no pixel da amostra (x,y) para que ao reprojetar as novas subamostras, seja possível conseguir uma densidade de amostras adequada na imagem de destino. O passo de subamostragem e reprojeção das amostras adicionais (passo 7) é ilustrado na figura (7-6).

Como resultado do processo de amostragem e subamostragem, obtemos uma densidade adequada de amostras e a visibilidade pode ser resolvida para cada pixel. Se todas as amostras do pixel são da mesma camada, o pixel pertence à camada das amostras. Se o pixel contém amostras de camadas diferentes, selecionamos a amostra de menor profundidade e as amostras que pertencem à mesma camada desta amostra, descartando as amostras de outras camadas. Após este processo, obtemos um conjunto de amostras visíveis, que serão encaminhadas para o processo de filtragem (algoritmo 9) para obter a imagem final. Na fi-

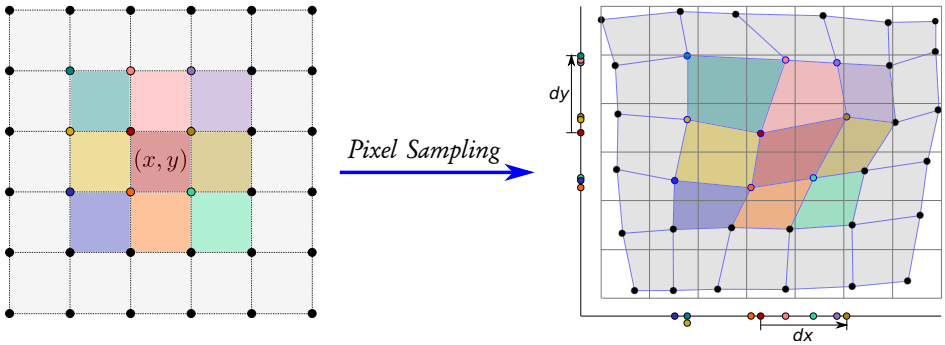


Figura 7-5: Esquerda: Pixels originais da camada do *POMC*. Os círculos representam a amostragem dos pixels. **Direita:** Os pixels da camada após sua transformação 3D e reprojeção no novo plano de imagem. Em linhas azuis ilustramos a relação de vizinhança das amostras originais.

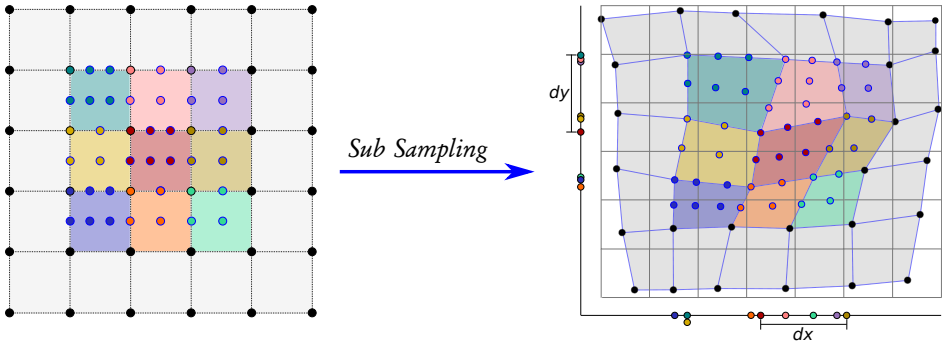


Figura 7-6: Esquerda: Amostragem adaptativa do POMC. As distâncias dx e dy de separação das amostras da vizinhança 8-conectada (direita), são utilizadas para determinar o número de amostras a ser coletadas em cada pixel. **Direita:** Posição das amostras adicionais geradas pela subamostragem na imagem da esquerda.

gura (7-7) mostramos algumas renderizações realizadas a partir de posições deslocadas da origem do *POMC*, onde a visibilidade foi tratada com o algoritmo 10.

(a) Renderizações de vistas a partir do *POMC*. **(b)** Detalhes ampliados. **(c)** Amostragem inicial.

Figura 7-7: (a) Renderização utilizando o algoritmo 10. Foi realizada uma filtragem bilinear com raio do filtro igual a 0.75 pixels. (b) detalhes aumentados das renderizações. (c) Pixels cobertos pelas amostras iniciais. Os espaços em branco não contêm amostras e são preenchidos no processo de subamostragem.

7.4 Renderização baseada em malhas

Os algoritmos de renderização e determinação de visibilidade propostos na seção anterior utilizam como primitivo básico o ponto. No *POMC*, pontos 3D são determinados a partir de uma direção $\omega \in \mathbb{S}^2$ e a correspondente profundidade segundo a camada considerada. Os métodos de determinação de superfícies visíveis tradicionais trabalham com primitivos bidimensionais, a diferença de um ponto que tem dimensão zero. Portanto, para trabalhar com algoritmos como *Z-Buffer* ou *ray tracing*, é preciso dispor de informação geométrica bidimensional da cena. Considere a nuvem de pontos 3D gerada a partir da informação

codificada na imagem de profundidade de uma camada. Estes pontos são amostras pontuais da superfície original da camada, logo uma interpolação adequada dos mesmos permite reconstruir uma aproximação da superfície original da camada.

Existem varias formas de aproximar a superfície da camada, entre elas: encontrar uma superfície implícita que aproxime a geometria da camada, ou bem gerar uma triangulação da superfície da camada a partir das amostras disponíveis. A renderização baseada em malhas de triângulos é a solução escolhida para propósitos de visualização interativa em tempo real, pela simplicidade e porque vários dos problemas (e.g. a visibilidade) são resolvidos pelo hardware gráfico.

As camadas foram construídas de maneira que sejam contínuas no atributo de profundidade no interior do seu domínio de parametrização. Este fato facilitará o trabalho de triangulação, pois não devemos nos preocupar com descontinuidades na superfície que podem atrapalhar a construção da triangulação. Por sua vez, as informações de cor da camada podem ser usadas como textura da malha obtida.

Como estamos usando parametrizações da esfera para codificar os domínios das camadas dos panoramas, podemos construir a triangulação em duas etapas. Primeiro construímos uma triangulação no domínio da parametrização U_i e logo aplicamos a transformação h_i aos pontos de dita triangulação para obter uma triangulação que aproxima a superfície 3D da camada em $\mathbb{R}^3$. Na figura (7-8) ilustramos o processo geral de triangulação. Se $N_i \subset U_i$ é um conjunto de pontos selecionados na parametrização 2D da camada i, e $\mathcal{T}(N_i)$ é uma triangulação de N_i, definiremos por $\mathcal{T}(h_i(N_i))$ a nova triangulação que representa a geometria 3D da camada i.

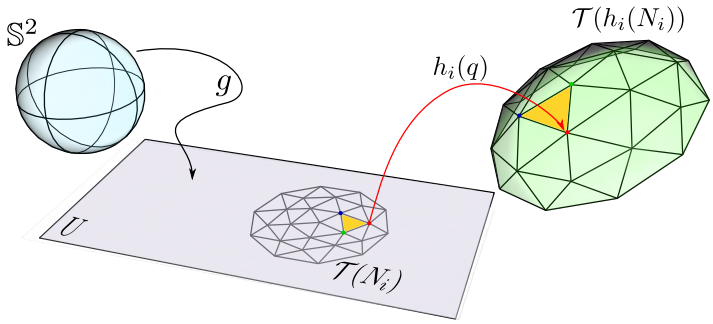


Figura 7-8: Processo de triangulação. Fazemos uma triangulação $\mathcal{T}(N_i)$ no domínio 2D, e aplicando a função h_i aos vértices 2D obtemos a triangulação $\mathcal{T}(h_i(N_i))$ em $\mathbb{R}^3$.

7.4.1 Malhas uniformes

Podemos construir uma grade regular de pontos $\mathcal{G} \subset \mathbb{R}^2$, e tomar $N_i = \{v \in \mathcal{G} \mid v \in U_i\}$, i.e., os pontos da grade $\mathcal{G}$ que pertencem a U_i. Finalmente, construímos uma triangulação $\mathcal{T}(N_i)$ do espaço de parametrização U_i e a correspondente triangulação $\mathcal{T}(h_i(N_i))$ da superfície da camada i. Como as camadas são independentes, podemos usar grades diferentes para cada camada e obter triangulações mais ou menos refinadas em cada camada.

Na prática as informações das camadas são armazenadas em imagens que têm uma determinada resolução em pixels. Assim, existe uma grade $\mathcal{G}_m$ que está correlacionada com a estrutura discreta induzida pela imagem de armazenamento da camada. Nesta grade $\mathcal{G}_m$ cada ponto $v \in U_i$ se corresponde com um pixel na imagem que contém os dados de profundidade da camada i.

Na malha uniforme de resolução máxima, um pixel da imagem de profundidade será representado por 2 triângulos retângulos. Desta forma, se U_i é uma região retangular representada por uma imagem I_i de $W \times H$ pixels, a malha de resolução máxima terá $2(W \cdot H)$ triângulos. Uma triangulação mais refinada não acrescentará detalhes pois a reconstrução é baseada nas informações armazenadas na imagem I_i que guarda as informações de profundidade da camada i. No contexto desta tese estaremos utilizando interpolação bilinear para amostrar posições dentro de uma imagem.

Para representar malhas uniformes, escolhemos uma estrutura de malha 4-8, para assim facilitar possíveis simplificações e poder trabalhar em resoluções diferentes se for necessário. Na figura (7-9) mostramos como os vértices dos pixels são usados para definir a triangulação no caso da malha de máxima resolução.

Não é recomendado usar uma malha uniforme de máxima resolução gerada a partir do *POMC*, especialmente para interação em tempo real. Uma alternativa intermediária é utilizar a textura da camada *RGB* do *POMC* em alta resolução e as camadas de profundidade em

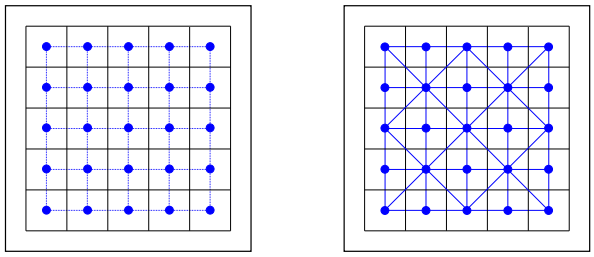
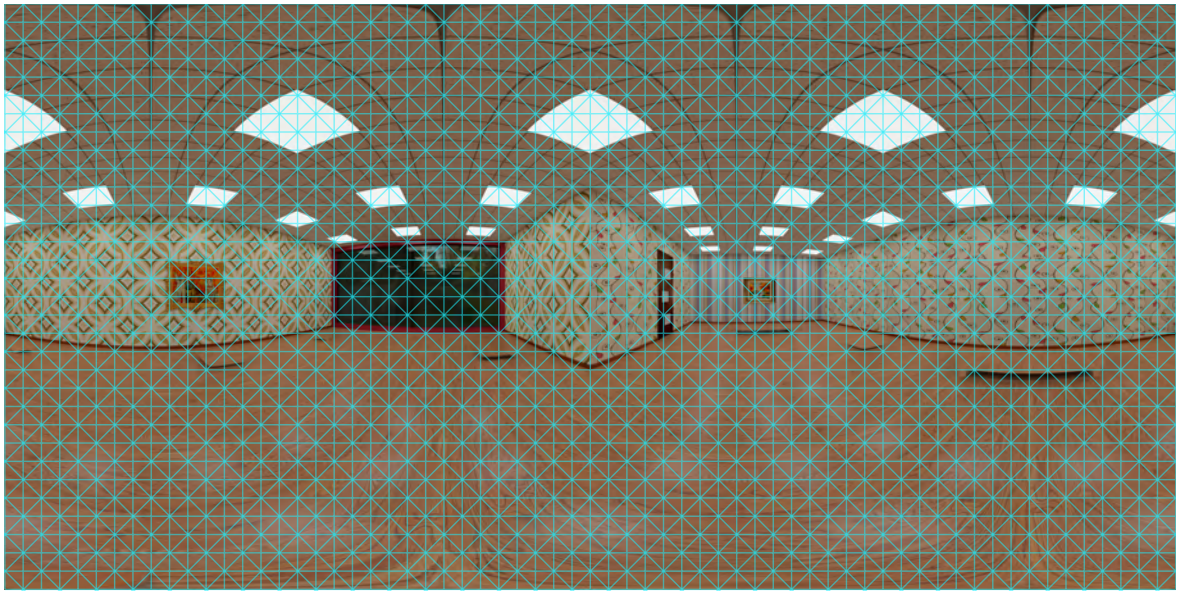
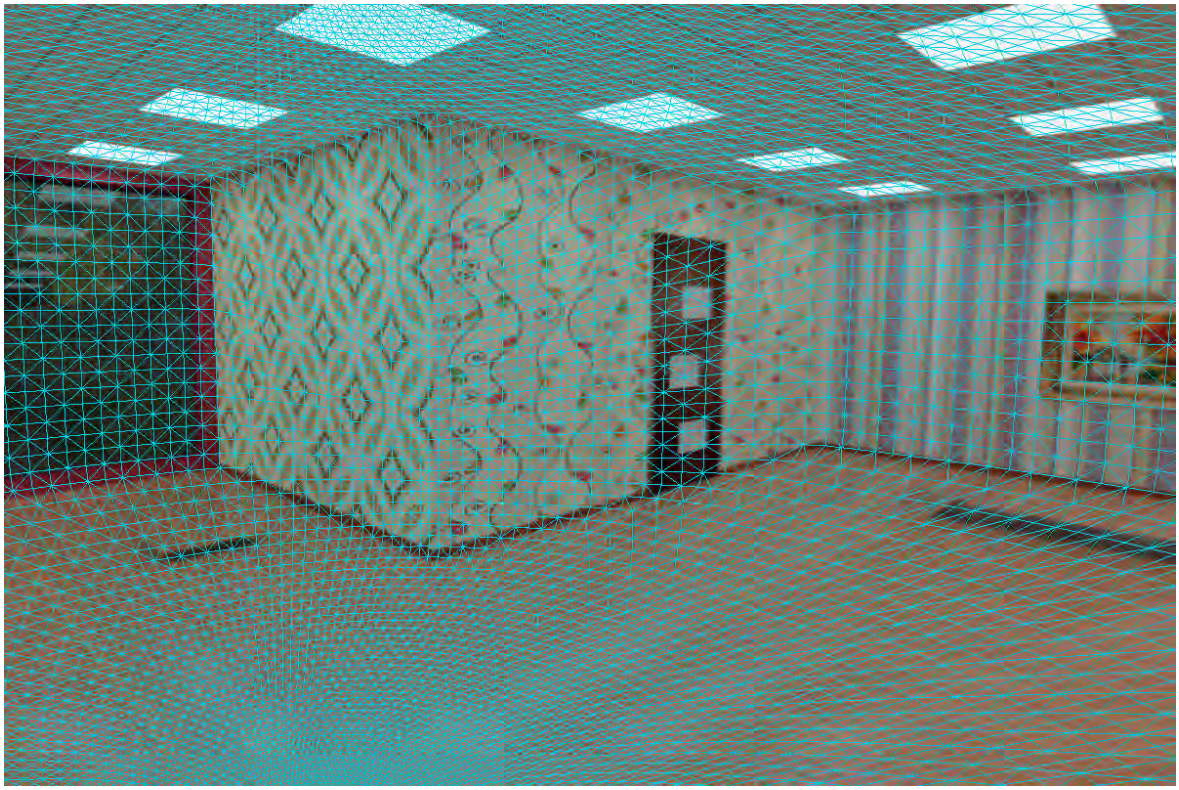


Figura 7-9: Esquema de montagem da malha de triângulos a partir dos pixels da imagem panorâmica. Cada pixel Subdivisão de pixels do panorama em triângulos com estrutura 4-8.



(a) Triangulação regular do domínio de parametrização do *POMC*.



(b) Visualização e uma vista 3D da malha regular (a).

Figura 7-10: Estrutura da malha base para uma camada do *POMC*. Para efeitos de ilustração diminuímos a resolução da malha. (a) é possível observar que o tamanho dos triângulos é regular. A parametrização representada é a do formato equiretangular. (b) é possível observar a distribuição dos triângulos da malha no espaço 3D. Os triângulos correspondentes às áreas próximas aos polos na representação equiretangular são menores e estão concentrados ao redor dos polos na malha 3D.

resolução menor, de forma que a triangulação é feita com a resolução da imagem de profundidade, más a textura *RGB* mantém a resolução original. Na figura (7-10) apresentamos um

exemplo de malha regular para a camada de fundo de um *POMC*.

Como as camadas são independentes e podem ser trabalhadas em resoluções diferentes, é aconselhável diminuir somente a resolução da camada de fundo, que geralmente contém o suporte da cena e não tem tantos detalhes geométricos quanto as camadas da frente. Assim, os objetos próximos devem ser mantidos na sua resolução original para não degradar seus atributos e forma, especialmente no contorno da malha.

Outra estratégia recomendável é mudar da representação equiretangular para uma representação de mapa de cubo, que apresenta menor distorção da cena e se adéqua melhor pois como sabemos um triangulo no mundo é projetado como um triângulo no cubo, salvo os triângulos projetados em mais de uma face do cubo. Isto não acontece no formato equiretangular, no qual a maior parte das linhas retas do mundo é mapeada em curvas, o que traz problemas na hora de fazer mapeamentos de textura, especialmente se a resolução da malha é muito diferente da resolução da textura.

7.4.2 Malhas não uniformes

As malhas uniformes podem ser suficientes em determinadas situações, mas em geral não são a melhor opção para representar uma superfície proveniente de uma cena real. Se a superfície não tem uma curvatura relativamente uniforme, uma malha uniforme não é a opção mais indicada. Para capturar bem as regiões com muita curvatura, são necessários triângulos pequenos, mas em regiões com curvatura próxima de 0 os triângulos podem ser maiores. O ideal é que a malha se adapte à geometria da camada para minimizar a quantidade de triângulos utilizados.

Nesta seção apresentaremos uma estratégia para construir uma triangulação adaptada. A partir de uma parametrização planar de uma camada i, faremos uma subdivisão do domínio de parametrização com uma quadtree adaptativa e construiremos uma triangulação $\mathcal{T}(N_i)$ com um conjunto de pontos N_i selecionados na quadtree. Assim, transformando os pontos, obteremos a triangulação $\mathcal{T}(h_i(N_i))$ da superfície da camada.

O método para gerar malhas adaptadas não uniformes que vamos apresentar é baseado no uso de quadtrees. A seguir descreveremos rapidamente o conceito de quadtree seguindo a abordagem de Mark de Berg et al. [16], fazendo as adaptações necessárias para o nosso problema de geração de malhas.

7.4.2.1 Quadtrees

As *quadtrees* são um tipo especial de árvore onde todos os nós têm quatro filhos ou são nós folha, e receberam este nome no trabalho de Finkel e Bentley em 1974 [25]. Cada nó em uma quadtree corresponde a um retângulo. Se um nó v tem filhos então estes correspondem aos quatro quadrantes do retângulo de v. Os nós filhos são indicados por NE, NW, SW, SE para indicar a que quadrante correspondem, ao 1°, 2°, 3° ou 4° quadrante respectivamente. Dois nós folha não se intersectam, e a união de todos os nós folhas forma o retângulo do nó raiz v^0.

Para construir uma quadtree, em geral é necessário seguir uma regra para decidir quais nós devem ser subdivididos e quando a subdivisão acaba. Se subdividirmos todos os nós até um nível k obteremos uma quadtree uniforme, onde teremos 4^k folhas, todas elas no nível k e com a área dos seus retângulos igual a área$(v^0)/4^k$. Podemos subdividir até um nível máximo k ou até satisfazer uma condição de parada. A figura (7-11) mostra dois exemplos de quadtrees e as respectivas árvores.

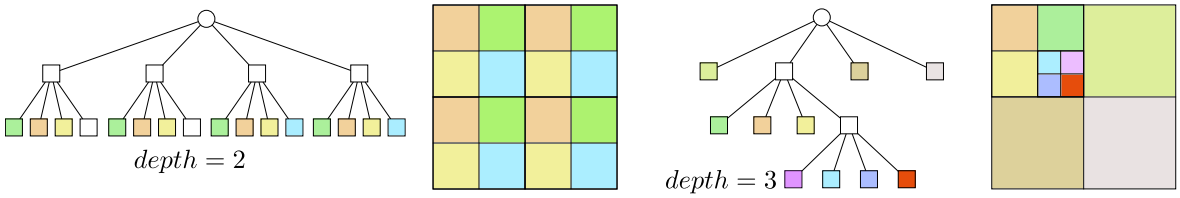


Figura 7-11: Exemplos de quadtrees. Do lado esquerdo uma quadtree uniforme e do lado direito uma quadtree não uniforme.

As quadtrees podem ser usadas para armazenar diferentes tipos de dados. Nossa descrição é genérica mas orientada ao propósito de geração de malhas. Começando pelo nó raiz v^0, faremos subdivisões recursivamente segundo o valor de uma função $\mathcal{S} : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}$ avaliada sobre a região retangular do nó.

Seja v um nó da quadtree, definimos o retângulo associado por $\delta(v) := [x_0, y_0] \times [x_1, y_1]$. As regras de subdivisão são as seguintes:

- Se $\mathcal{S}(\delta(v)) = 0$, então o nó v não será subdividido e será marcado como uma folha.
- Se $\mathcal{S}(\delta(v)) = 1$, então o nó v precisa ser subdividido em 4 quadrantes v_{NE}, v_{NW}, v_{SW} e v_{SE}. Se $x_m := (x_0 + x_1)/2$ e $y_m := (y_0 + y_1)/2$, os retângulos dos novos quadrantes são

definidos como:

$$\begin{aligned}\delta(v_{NE}) &:= [x_m, y_m] \times [x_1, y_1],\\ \delta(v_{NW}) &:= [x_0, y_m] \times [x_m, y_1],\\ \delta(v_{SW}) &:= [x_0, y_0] \times [x_m, y_m],\\ \delta(v_{SE}) &:= [x_m, y_0] \times [x_1, y_m],\end{aligned}$$

A definição recursiva da quadtree se translada imediatamente a um algoritmo recursivo para gerar a quadtree. Começando pelo nó raiz v^0 é avaliada a função $\mathcal{S}$ no nó e o valor da mesma decide se o nó é subdividido em 4 novos nós ou não. A recursão acaba em um nó acaba quando $\mathcal{S}$ devolve um valor que impede a subdivisão. O algoritmo 11 mostra o processo de recursão para criar a quadtree. A primeira chamada é feita com o nó raiz, $Quadtree(v^0)$.

Algorithm 11 $Quadtree(v)$

```
1:  Input: um nó v;
2:  if S(v) = 1 then
3:      split v into 4 nodes v_NE, v_NW, v_SW, e v_SE;
4:      Quadtree(v_NE);
5:      Quadtree(v_NW);
6:      Quadtree(v_SW);
7:      Quadtree(v_SE);
8:  else
9:      return;
10: end if
```

Uma operação que costuma ser necessária ao trabalhar com quadtrees é a busca de vizinhos: Dado um nó v e uma direção (Norte, Sul, Leste, Oeste), determinar o nó v' tal que $\delta(v')$ é adjacente a $\delta(v)$ nesta direção. Se existir, o nó retornado deve ter a mesma profundidade que v ou se for menor , v' deve ser uma folha. Por exemplo, suponha que dado um nó v, desejamos encontrar seu vizinho do Sul. Se v não é o nó raiz, possui um nó pai w. Se $v = w_{NE}$ o vizinho ao sul de v é w_{SE}. Se $v = w_{NW}$ o vizinho ao sul é w_{SW}. no entanto, se $v = w_{SW}$ ou $v = w_{SE}$, o vizinho Sul de v não é um filho de w. Buscamos o vizinho sul de w. Se este vizinho é uma folha, ele será o vizinho sul de v, caso contrario algum dos seus filhos dos quadrantes Norte (E/W) será o vizinho sul de v. Nos algoritmos 12 e 13 mostramos como é realizada a busca dos vizinhos do sul e oeste respectivamente.

O algoritmo de busca de vizinhos devolve para um nó v o nó adjacente a ele na direção indicada e que tem a mesma profundidade que v na quadtree. No caso que não seja possível

determinar um nó com igual profundidade, o algoritmo retorna o nó vizinho v' com maior profundidade $d' < d$. Se v' existe, ele é único, pois como $d' < d$ isto significa que o retângulo $\delta(v')$ tem lado maior que $\delta(v)$. Um nó v que tem um dos lados do seu retângulo $\delta(v)$ sobre a borda do retângulo $\delta(v^0)$ do nó raiz v^0, não terá vizinho na direção que aponta para a aquela borda. A figura (7-12) mostra alguns exemplos do vizinho do sul para alguns nós de uma quadtree.

Outro conceito importante para no contexto de geração de malhas a partir de quadtrees é o de quadtree balanceada. Dizemos que uma quadtree está balanceada se para qualquer par de nós vizinhos v e v', a razão de área entre seus retângulos é no máximo 4. Assim, numa quadtree balanceada duas folhas vizinhas têm uma diferença de nível de no máximo 1. A figura (7-13) mostra uma quadtree Q não balanceada e sua versão balanceada.

Algorithm 12 *SouthNeighbor*(v, Q)

Input: um nó v da quadtree Q;
Output: o nó v' mais profundo cuja profundidade é no máximo a profundidade de v;
if $v = root(Q)$ **then return** NULL; **end if**
if $v =$ NW-child of $parent(v)$ **then return** SW-child of $parent(v)$; **end if**
if $v =$ NE-child of $parent(v)$ **then return** SE-child of $parent(v)$; **end if**
$m \leftarrow SouthNeighbor(parent(v),\ Q)$;
if $m =$ NULL **or** m is leaf **then**
    return m;
else if $v =$ SW-child of $parent(v)$ **then**
    return NW-child of m;
else
    return NE-child of m;
end if

Algorithm 13 *WestNeighbor*(v, Q)

Input: um nó v da quadtree Q;
Output: o nó v' mais profundo cuja profundidade é no máximo a profundidade de v;
if $v = root(Q)$ **then return** NULL; **end if**
if $v =$ NE-child of $parent(v)$ **then return** NW-child of $parent(v)$; **end if**
if $v =$ SE-child of $parent(v)$ **then return** SW-child of $parent(v)$; **end if**
$m \leftarrow WestNeighbor(parent(v),\ Q)$;
if $m =$ NULL **or** m is leaf **then**
    return m;
else if $v =$ NW-child of $parent(v)$ **then**
    return NE-child of m;
else
    return SE-child of m;
end if

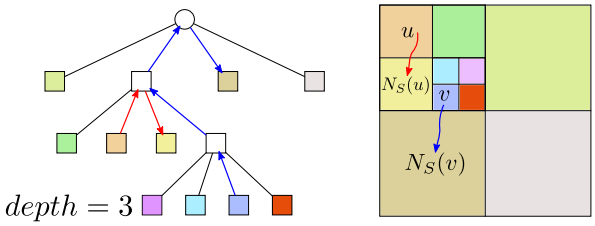


Figura 7-12: Busca de vizinhos na quadtree. Na árvore do lado esquerdo desenhamos o caminho a seguir na árvore para encontrar o vizinho do sul paro os nós u e v, em vermelho e azul respectivamente. O vizinho sul do nó v é uma folha de profundidade menor que v.

O algoritmo 14 mostra como é feito o processo de balanceamento de uma quadtree. Para balancear a quadtree é preciso percorrer a árvore e decidir se algumas folhas devem ou não ser subdivididas. Em primeiro lugar, é preciso decidir se uma folha v deve ou não ser subdividida. Para isto temos que olhar os 4 vizinhos da folha v. Se alguma folha que comparte parte de uma aresta com $\delta(v)$ tem área menor que $1/4$ da área de $\delta(v)$ (isto significa uma diferença de profundidade maior a 1), é necessário subdividir v. Para fazer isto usamos as funções de busca de vizinhos. Por exemplo, dado o nó v procuramos seu vizinho do sul $w = SouthNeighbor(v)$. O lado do retângulo $\delta(w)$ é maior ou igual ao lado de $\delta(v)$, logo se w é uma folha, não é preciso subdividir v. Se w não é folha, devemos olhar de w_{NE} e w_{NW} são folhas. Se ambos são folhas v não precisa ser subdividido, caso contrario v será subdividido. Este teste deve ser realizado par as 4 vizinhanças.

Em segundo lugar, se v foi subdividido, seus filhos tem um nível $d+1$. Agora é necessário verificar se algum dos antigos vizinhos de v devem ser subdivididos por causa dos novos filhos. Isto também é analisado usando as funções de busca de vizinhos, como foi feito com v.

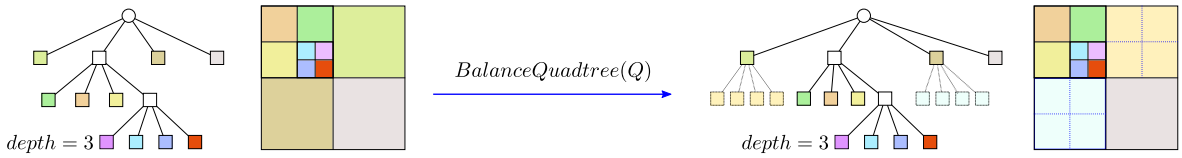


Figura 7-13: Balanceamento da quadtree. Depois do processo de balanceamento, os nós vizinhos não podem ter uma diferença maior que um nível de profundidade.

Como os cálculos que pretendemos fazer são sobre representações discretas das camadas, utilizaremos uma versão discreta de quadtree que discutiremos a seguir.

Algorithm 14 *BalanceQuadtree*(Q)

Input: uma quadtree Q;
Output: uma versão balanceada de Q;
Colocar todas as folhas de Q numa lista $\mathcal{L}$;
while $\mathcal{L} \neq \emptyset$ **do**
    remover a folha v de $\mathcal{L}$;
    if $\delta(v)$ deve ser subdividido **then**
        marque v como nó interno com 4 filhos v_{NE}, v_{NW}, v_{SW}, e v_{SE}.
        Inserte v_{NE}, v_{NW}, v_{SW}, e v_{SE} em $\mathcal{L}$.
        Verificar se $\delta(v)$ tem vizinhos que precisam ser subdivididos. Se sim, insira-los em $\mathcal{L}$.
    end if
end while

7.4.2.2 Quadtree sobre domínio discreto

Após ter visto como é definida e funciona uma quadtree sobre um domínio contínuo, passaremos agora a ver como adaptamos esta ideia para o caso discreto. As imagens onde estão armazenadas as informações do *POMC* são discretas e tem uma resolução dada. Portanto buscamos construir uma árvore quadtree usando como nó raiz a uma imagem e subdividindo ao longo dos pixels, de forma tal que os pixels sempre estejam contidos completamente em algum quadrante e não aconteça que um pixel fique dividido em duas ou mais partes ocupando mais de um quadrante. Desta forma os quadrantes sempre terão uma área inteira em pixels. Também vamos interromper a subdivisão quando a área de um retângulo for igual a 1 pixel, assim um nó folha terá área maior ou igual a 1 pixel.

Para poder operar de maneira discreta é necessário levar em conta algumas pequenas mudanças na quadtree contínua, que não são difíceis de entender conceitualmente, mas são aumentam as dificuldades de implementação.

Suponha que temos um nó v cujo retângulo $\delta(v) := [x_0, y_0] \times [x_1, y_1]$ é formado pelos pixels $I(x,y)$ tais que $x_0 \leq x \leq x_1$ e $y_0 \leq y \leq y_1$. Definimos o pixel médio do retângulo $\delta(v)$ como $x_m := \lfloor (x_0 + x_1)/2 \rfloor$ e $y_m := \lfloor (y_0 + y_1)/2 \rfloor$. O processo de subdivisão é feito da seguinte forma:

- Se $(x_1 - x_0 > 1)$ e $(y_1 - y_0 > 1)$, subdividimos v em quatro nós v_{NE}, v_{NW}, v_{SW}, e v_{SE}

cujos retângulos são

$$\begin{aligned}\delta(v_{NE}) &:= [x_m, y_m] \times [x_1, y_1],\\ \delta(v_{NW}) &:= [x_0, y_m] \times [x_m, y_1],\\ \delta(v_{SW}) &:= [x_0, y_0] \times [x_m, y_m],\\ \delta(v_{SE}) &:= [x_m, y_0] \times [x_1, y_m],\end{aligned}$$

- Se $(x_1 - x_0 > 1)$ e $(y_1 - y_0 = 1)$, subdividimos v em dois nós horizontais (split vertical) v_{SW}, e v_{SE} cujos retângulos são

$$\begin{aligned}\delta(v_{SW}) &:= [x_0, y_0] \times [x_m, y_1],\\ \delta(v_{SE}) &:= [x_m, y_0] \times [x_1, y_1],\end{aligned}$$

- Se $(x_1 - x_0 = 1)$ e $(y_1 - y_0 > 1)$, subdividimos v em dois nós verticais (split horizontal) v_{SW}, e v_{NW} cujos retângulos são

$$\begin{aligned}\delta(v_{SW}) &:= [x_0, y_0] \times [x_1, y_m],\\ \delta(v_{NW}) &:= [x_0, y_m] \times [x_1, y_1],\end{aligned}$$

- Se $(x_1 - x_0 = 1)$ e $(y_1 - y_0 = 1)$, v não será subdividido.

No caso discreto, como as subdivisões são inteiras, em varias oportunidades teremos que as áreas dos 4 quadrantes da subdivisão podem ser diferentes, como é ilustrado na figura (7-14). A diferença é dada pelo arredondamento do ponto médio de subdivisão. Isto não traz problemas, e de modo geral a estratégia adotada na definição do ponto médio é a de truncamento de coordenadas reais para coordenadas inteiras. Outra mudança que notamos é que nem sempre será possível fazer subdivisões obtendo 4 filhos, quando um dos lados do nó a ser subdividido for igual a 1 pixel. As parametrizações que utilizaremos, equiretangular e

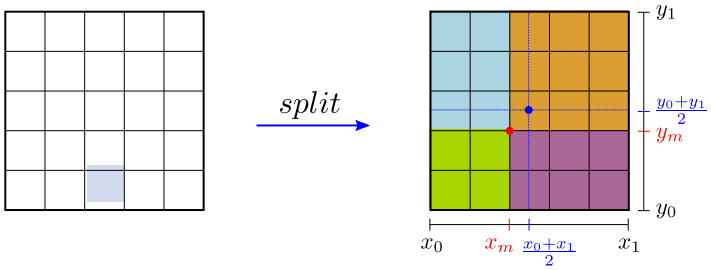


Figura 7-14: Subdivisão da quadtree discreta. Se o centro do nó é $(x,y) = \left(\frac{x_0+x_1}{2}, \frac{y_0+y_1}{2}\right)$, o ponto de subdivisão discreto é dado pelas coordenadas inteiras $(x_m, y_m) = (\lfloor x \rfloor, \lfloor y \rfloor)$.

Algorithm 15 *SouthNeighbor*(v, Q)

```
1: Input: um nó v da quadtree Q;
2: Output: o nó v' mais profundo cuja profundidade é no máximo a profundidade de v;
3: if v = root(Q) then return NULL; end if
4: m ← SouthNeighbor(parent(v), Q);
5: if δ(v) tem lados maiores que 1 pixel then
6:     if v = NW-child of parent(v) then return SW-child of parent(v); end if
7:     if v = NE-child of parent(v) then return SE-child of parent(v); end if
8:     if m = NULL or m is leaf then return m;
9:     else if v = SW-child of parent(v) then
10:        return NW-child of m;
11:    else
12:        return NE-child of m;
13:    end if
14: else if δ(v) tem o lado horizontal maior que 1 pixel then
15:    if m = NULL or m is leaf then return m; end if
16:    if δ(m) tem lado vertical maior que 1 pixel then
17:        if v = SW-child of parent(v) then
18:            return NW-child of m;
19:        else
20:            return NE-child of m;
21:        end if
22:    else
23:        if v = SW-child of parent(v) then
24:            return SW-child of m;
25:        else
26:            return SE-child of m;
27:        end if
28:    end if
29: else if δ(v) tem o lado vertical maior que 1 pixel then
30:    if m = NULL or m is leaf then
31:        return m;
32:    else
33:        return NW-child of m;
34:    end if
35: end if
```

mapa de cubo, têm domínios retangulares de razão 2:1 e 3:2 respectivamente. A pior relação 2:1 pode gerar na pior das hipóteses retângulos de razão 3:1 na horizontal, figura (7-14). Isto significa que estes retângulos podem ser de 3×1 pixels e podem sofrer duas subdivisões mais. No entanto, estas subdivisões degeneradas acontecem no último e penúltimo nível de subdivisão e não trazem grandes inconvenientes para a estrutura da quadtree.

A busca de vizinhos também deve levar em conta os tipos de subdivisões, pois perto das folhas onde as subdivisões são degeneradas, a vizinhança não pode ser estimada com o

algoritmo tradicional. Apresentamos no algoritmo 15 a estratégia correta para determinar os vizinhos no caso da quadtree discreta.

Tanto no processo de busca de vizinho quanto de balanceamento da quadtree é necessário contemplar os casos de partições degeneradas.

7.4.2.3 Imagem integral - *Summed Area Table*

O conceito de imagem integral ou *Summed Area Table* (SAT), foi introduzido em computação gráfica por Crow em 1984 [15], para ser usado com *mipmaps* na filtragem de texturas.

Para uma imagem $I(x,y)$, a imagem integral $SAT_I(x,y)$ é dada por

$$SAT_I(x,y) = \sum_{y' \leq y} \sum_{x' \leq x} I(x',y'). \tag{7.3}$$

A imagem integral SAT_I pode ser calculada percorrendo uma única vez a imagem I dado que

$$SAT_I(x,y) = I(x,y) + SAT_I(x-1,y) + SAT_I(x,y-1) - SAT_I(x-1,y-1). \tag{7.4}$$

Uma vez que temos calculada a imagem integral SAT_I, é possível calcular a soma de intensidades dos pixels para um retângulo $R = [x_0,x_1] \times [y_0,y_1]$ na imagem I através da imagem integral SAT_I realizando o cálculo em tempo constante para qualquer retângulo

$$\sum_{\substack{x_0 \leq x \leq x_0 \\ y_0 \leq y \leq y_1}} I(x,y) = SAT_I(x_1,y_1) - SAT_I(x_0-1,y_1) - SAT_I(x_1,y_0-1) + SAT_I(x_0-1,y_0-1). \tag{7.5}$$

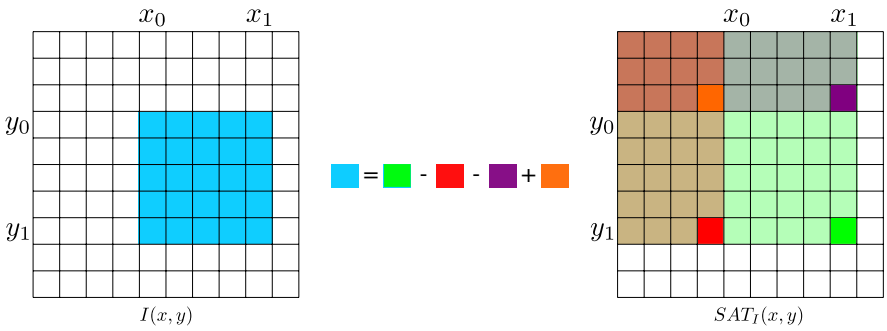


Figura 7-15: Relação entre a imagem $I(x,y)$ e sua imagem integral $SAT_I(x,y)$. Na figura está ilustrada a relação entre $I(x,y)$ e $SAT_I(x,y)$ no cálculo da intensidade acumulada nos pixels de um retângulo da imagem I. A equação (7.5) mostra como o cálculo é efetuado.

Nas próximas seções, veremos que as imagens integrais serão uma ferramenta fundamental para avaliar uma função sobre a área de um retângulo de uma quadtree.

7.4.2.4 Malhas baseadas em quadtree

Dada uma quadtree Q que foi subdividida com uma função de subdivisão $\mathcal{S}$, queremos construir uma malha de triângulos usando os vértices dos retângulos das folhas de Q. Se subdividirmos todos os retângulos que correspondem a um nó folha de Q em dois triângulos, obteremos uma triangulação $\mathcal{T}$. O problema é que se Q não é uniforme, a malha obtida desta forma não será conforme, i.e., alguns vértices de triângulos estarão no meio da aresta de outros triângulos. Ao transformar esta malha do plano 2D para uma representação 3D teremos buracos (cracks) visíveis na malha. Para evitar estas malhas e assegurar a conformidade da malha final, necessário ter alguns cuidados. O primeiro passo é balancear a quadtree Q. Com isto garantimos que a diferença de nível entre um nó e seu vizinho é no máximo 1, o que significa que no interior de cada lado de um retângulo de um nó existe no máximo um vértice correspondente a outro retângulo, figura (7-13).

Uma vez que temos a quadtree balanceada Q, procedemos a realizar sua triangulação da seguinte forma. Retângulos que não têm um vértice no interior de um dos seus lados são triangulados dividindo o retângulo em 2 triângulos por uma das diagonais. Como Q está balanceada, os retângulos restantes tem no máximo um vértice no interior de cada um dos seus lados (posicionados no meio de cada lado, exceto no caso discreto). Por isso, adicionamos um novo ponto no centro do retângulo e o conectamos com todos os vértices que estão sobre os lados do retângulo (no pior caso são 8 vértices - 4 vértices do retângulo e 4 vértices no meio dos lados), figura (7-16).

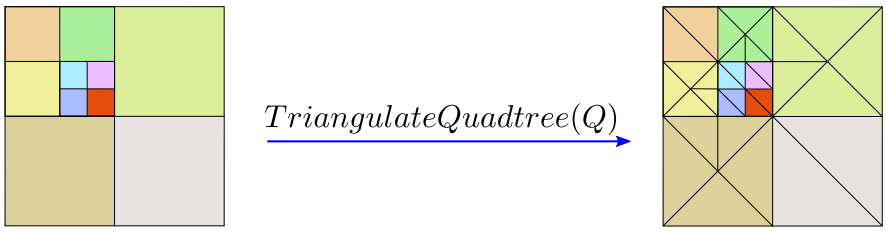


Figura 7-16: Triangulação de uma quadtree balanceada. Nós que tem profundidade maior ou igual que todos seus vizinhos são subdivididos em 2 triângulos. Nós que tem algum vizinho com maior profundidade recebem um ponto adicional no centro do nó. Este ponto é unido com os 4 vértices de nó e com os vértices existentes no meio das arestas.

7.4.2.5 Malha não uniforme para uma camada

Para uma camada i, usaremos como nó raiz v^0 a imagem de profundidade $I(x,y)$ da camada. Realizamos a construção da quadtree discreta Q usando uma função de subdivisão

$\mathcal{S} : \mathbb{N}^2 \times \mathbb{N}^2 \to \mathbb{R}$. Uma vez que temos a quadtree Q, balanceamos a mesma para proceder à triangulação da mesma. A triangulação de Q é realizada segundo a especificação dada na seção 7.4.2.4. Obteremos um conjunto de pontos 2D $K_i \subset \mathbb{N}^2$ e a triangulação $\mathcal{T}(K_i)$ da quadtree discreta Q. Transformando os pontos do conjunto K_i para seus correspondentes no espaço 3D, obteremos a malha não uniforme $\mathcal{T}(h_i(K_i))$ que representa a superfície da camada.

Como os domínios U_i das camadas podem ser irregulares e podem não coincidir com o suporte da imagem $I(x,y)$ onde está armazenada a informação, é necessário usar uma imagem D de máscara para delimitar o domínio discreto da camada na imagem I. Esta máscara D é usada como informação pela função de subdivisão $\mathcal{S}$ para ajustar o refinamento da quadtree sobre o domínio da camada. À medida que são identificados nós que estão completamente fora do domínio ou completamente dentro dele, são devidamente marcados. Assim, no momento de realizar a triangulação da camada, somente são selecionados vértices que se encontram em nós que pertencem ao domino para formar o conjunto K_i a ser triangulado.

Na próxima seção mostraremos como é definida a função de subdivisão $\mathcal{S}$ para nossos propósitos, e como a mesma é implementada algoritmicamente.

7.4.2.6 Função de subdivisão $\mathcal{S}$

Como a função de subdivisão tem que realizar diversos cálculos sobre a área do nó, que é retangular, são utilizadas *Summed Area Tables* para realizar estes cálculos de forma eficiente. O processo de avaliação da função $\mathcal{S}(v)$ envolve uma série de passos e testes para determinar se o nó v deve ou não ser subdividido. Estes testes fazem uso de funções intermediarias calculadas a partir dos dados da camada. No algoritmo 16 apresentamos o processo que decide a subdivisão no algoritmo 11 de criação da quadtree.

Para facilitar o entendimento do processo de construção da função de subdivisão $\mathcal{S}$, dividiremos o processo em etapas. Os passos a seguir basicamente são:

- SAT_D : *Summed Area Table* do domínio discreto D da camada;
- SAT_B : *Summed Area Table* da borda B do domínio D da camada;
- SAT_F : *Summed Area Table* de features F obtidas pela variação de normais na camada;
- SAT_{V_R} : *Summed Area Table* valores residuais V_R da variação de normais na camada;

Algorithm 16 Função de subdivisão $\mathcal{S}(v)$

Input: um nó v da quadtree Q;
Output: 1 - continua a subdivisão, 0 - interrompe a subdivisão;

if $SAT_D(v) = 0$ **then** ▷ v está completamente fora do domínio D
    marcar v como nó externo;
    return 0;
else
    if $SAT_D(v) =$ área(v) **then** marcar v como nó interno; **end if**
    if área$(v) = 1$ **then** ▷ v tem área mínima = 1 pixel
        return 0;
    end if
    if $SAT_B(v) \geq 1$ **then** ▷ v intersecta a borda B do domínio
        return 1;
    end if
    if $normals\_threshold \leq \frac{SAT_{V_R}(v)}{\text{área}(v)}$ **then** ▷ variação residual media de $v >$ limiar
        return 1;
    end if
    if $SAT_F(v) \geq 1$ **then** ▷ v intersecta alguma feature F
        return 1;
    end if
    return 0;
end if

Tanto a imagem de features F, quanto a imagem de valores residuais de variação de normais V_R, estão correlacionadas e são obtidas a partir de operações sobre uma imagem de normais $N(x,y)$ da camada.

A seguir vamos descrever como obter cada uma das imagens usadas para construir as *Summed Area Tables* que são avaliadas na função $\mathcal{S}(v)$.

Domínio discreto D e borda B da camada

O domínio discreto ou máscara do domínio para a imagem de atributos I da camada é definida como

$$D(x,y) := \begin{cases} 1 & \text{se} \quad (x,y) \text{ pertence à camada ,} \\ 0 & \text{caso contrario} \end{cases} \tag{7.6}$$

onde D pode ser obtido analisando um canal especial de I (e.g. canal alpha) ou bem é determinada por um valor especial da imagem $I(x,y)$ nos pixels que não pertencem ao domínio (e.g. um valor negativo no atributo de profundidade). Seja $\mathcal{V}_8$ a vizinhança 8-conectada, definida por

$$\mathcal{V}_8(x,y) = \{(u,v) \in [0,W] \times [0,H] : \qquad \|(x-u, y-v)\|_{max} = 1\}. \tag{7.7}$$

Dada uma máscara D do domínio, a correspondente máscara B da borda é definida como

$$B(x,y) := \begin{cases} 1 & \text{se} \qquad D(x,y) = 1 \quad \text{e} \quad \exists (u,v) \in \mathcal{V}_8 : \quad D(u,v) = 0, \\ 0 & \text{caso contrario.} \end{cases} \tag{7.8}$$

Alguns exemplos de domínios e fronteiras de uma camada podem ser visualizados nas figuras (7-18b) e (7-19b).

Calculo de normais

Se não dispomos das normais correspondentes às amostras armazenadas nas camadas, devemos realizar uma estimativa das mesmas a partir da informação disponível. A seguir mostramos como inferir uma normal para uma amostra p armazenada no pixel (x,y) da imagem de profundidade I de uma camada.

Dado um triângulo $T = (x_i, x_j, x_k)$, a normal associada ao triângulo T é calculada como

$$\mathbf{n}(T) = \frac{(x_j - x_i) \times (x_k - x_i)}{\|(x_j - x_i) \times (x_k - x_i)\|}. \tag{7.9}$$

Em uma malha de triângulos, a normal para um vértice v da malha é calculada como uma média ponderada das normais dos triângulos formados pela vizinhança *one-ring* $\mathcal{N}_1(v)$ do vértice:

$$\mathbf{n}(v) = \frac{\sum_{T \in \mathcal{N}_1(v)} \alpha_T \mathbf{n}(T)}{\|\sum_{T \in \mathcal{N}_1(v)} \alpha_T \mathbf{n}(T)\|}, \tag{7.10}$$

onde o parâmetro de ponderação α_T pode ser atribuído de varias maneiras:

- Pesos constantes $\alpha_T = 1$. Simples e eficientes para os cálculos, mas não levam em conta o tamanho dos triângulos da vizinhança *one-ring*.
- Pesos baseados na área do triângulo, $\alpha_T = \text{área}(T)$. Método eficiente, produz resultados melhores que o peso constante, pois leva em conta o tamanho dos triângulos.
- Peso baseado no ângulo de incidência $\alpha_T = \theta_T$. Os cálculos de funções trigonométricas envolvidas fazem deste método menos eficiente, porém apresenta resultados superiores.

Para uma camada i, com imagem de profundidade $I(x,y)$, consideramos a triangulação uniforme máxima $\mathcal{T}(I)$ e sua correspondente triangulação $\mathcal{T}(h_i(I))$ no espaço 3D. Para cada pixel $(x,y) \in I$, podemos estimar a normal do ponto 3D $p = h_i(x,y)$ correspondente, usando a fórmula (7.10)

$$\mathbf{n}(p) = \frac{\sum_{T \in \mathcal{N}_1(p)} \alpha_T \mathbf{n}(T)}{\| \sum_{T \in \mathcal{N}_1(p)} \alpha_T \mathbf{n}(T) \|}, \tag{7.11}$$

onde $\mathcal{N}_1(p)$ é o *one-ring* de p em $\mathcal{T}(h_i(I))$, e α_T é uma das ponderações apresentadas. Finalmente, armazenamos todas as normais calculadas em uma imagem $N(x,y)$. Nas figuras (7-17a), (7-18a) e (7-19a) é possível observar as normais calculadas para as diferentes camadas do *POMC*.

Funções de variação de normais

A partir da imagem N das normais correspondentes aos pontos 3D $h_i(I)$, calculamos uma função de variação angular de normais $\text{grad}(N) = (\partial_x N, \partial_y N)$ da seguinte maneira

$$\partial_x N(x,y) = 1 - \| \langle N(x,y),\ N(x-1,y) \rangle \| \tag{7.12}$$

$$\partial_y N(x,y) = 1 - \| \langle N(x,y),\ N(x,y-1) \rangle \| \tag{7.13}$$

Observe que $\text{grad}(N)(x,y) \in [0,1] \times [0,1]$. Construímos agora uma imagem $V_N(x,y)$ tal que

$$V_N(x,y) = \| \text{grad}(N)(x,y) \|, \tag{7.14}$$

isto é,

$$V_N(x,y) = \sqrt{(\partial_x N(x,y))^2 + (\partial_y N(x,y))^2}. \tag{7.15}$$

A imagem V_N codifica a função que mede a variação angular das normais. Exemplos da imagem V_N para diferentes camadas do *POMC* podem ser vistas nas figuras (7-17b), (7-18c) e (7-19c).

Dado um limiar angular *angular_threshold*, vamos extrair a partir de V_N duas novas imagens complementares:

$$F(x,y) := \begin{cases} 1 & \text{se} \quad V_N(x,y) \geq \mathit{angular\_threshold}, \\ 0 & \text{se} \quad V_N(x,y) < \mathit{angular\_threshold}, \end{cases} \tag{7.16}$$

$$V_R(x,y) := \begin{cases} V_N(x,y) & \text{se} \quad V_N(x,y) < angular\_threshold, \\ 0 & \text{se} \quad V_N(x,y) \geq angular\_threshold, \end{cases} \tag{7.17}$$

O limiar angular *angular_threshold* é determinado a partir de um ângulo de variação θ como

$$angular\_threshold = 1 - \cos(\theta). \tag{7.18}$$

A forma de *angular_threshold* está diretamente relacionada com a forma de V_N. Como pode ser observado nas equações (7.16) e (7.17), o valor *angular_threshold* é usado para separar V_N em uma máscara de features, F, que contém as áreas de maior variação de normais, enquanto que a imagem V_R é um conjunto residual da variação de normais na região complementar de F. Nas figuras (7-17c), (7-18d) e (7-19d) apresentamos alguns exemplos das imagens F e V_R para as camadas do *POMC*. Nas figuras (7-20) e (7-21) mostramos alguns resultados de malhas 2D e 3D das camadas do *POMC*, obtidas pelo processo de uma subdivisão baseado em quadtrees discretas que acabamos de descrever. Em cada caso a função de subdivisão $\mathcal{S}$ foi implementada com os resultados expostos nas figuras (7-17), (7-18) e (7-19).

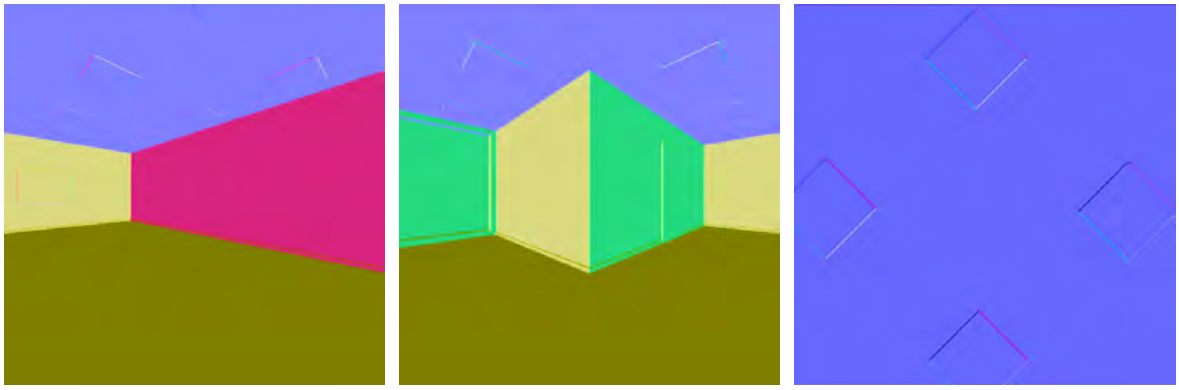
7.4.2.7 Codificação de triangulações

As triangulações baseadas em quadtree podem ser codificadas e armazenadas no panorama. Podemos fazer isto de varias maneiras podemos guardar um número inteiro que indique a posição do elemento do triangulo na imagem.

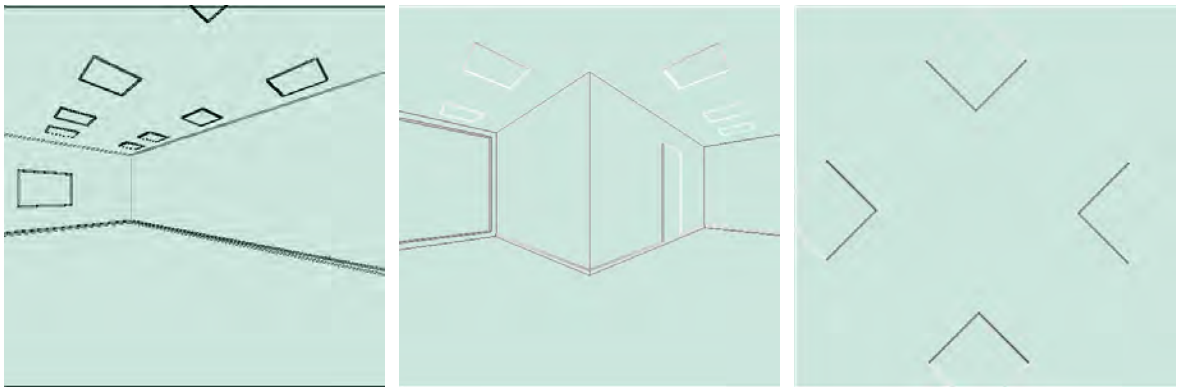
Triangulações podem ser armazenadas de maneira eficiente em *POMC*. Se considerarmos o panorama completo, com sua informação de profundidade, somente precisamos guardar 2 listas para determinar a triangulação. A primeira lista contém a posição do vértice no panorama e a segunda lista contém índices da triangulação. Na primeira lista não é necessário guardar valores de posição (x,y,z). Em lugar disso podemos guardar o offset do pixel que determina o vértice no panorama. Conhecendo a resolução do panorama, recuperamos a posição (i,j) do pixel e a partir dele , usando a informação de profundidade podemos reconstituir o ponto (x,y,z).

Uma alternativa, para malhas baseadas em quadtrees, é guardar uma lista de bits que armazena de maneira linear a estrutura da quadtree. O primeiro bit representa o nó raiz, os

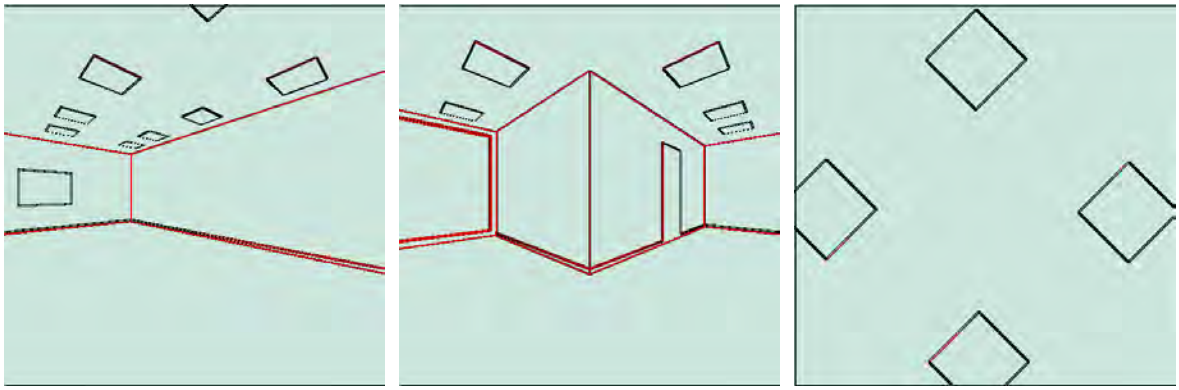
4 seguintes os seus filhos, e assim por diante. Um bit 0 indica que o nó é uma folha e não terá filhos. A quantidade de bits usados é menor que o número de triângulos da triangulação (1 bit por quad, e pelo menos 2 triângulos por quad nó). Assim, uma malha de 1 milhão de triângulos pode ser codificada com menos de 122Kbytes.



(a) Mapa de normais para camada de fundo do *POMC*.

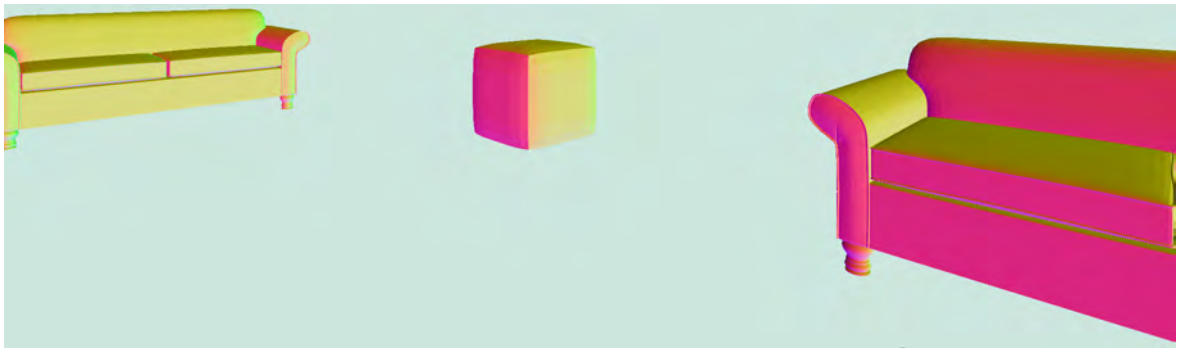


(b) Mapa das diferenças de normais correspondente as face de (a).

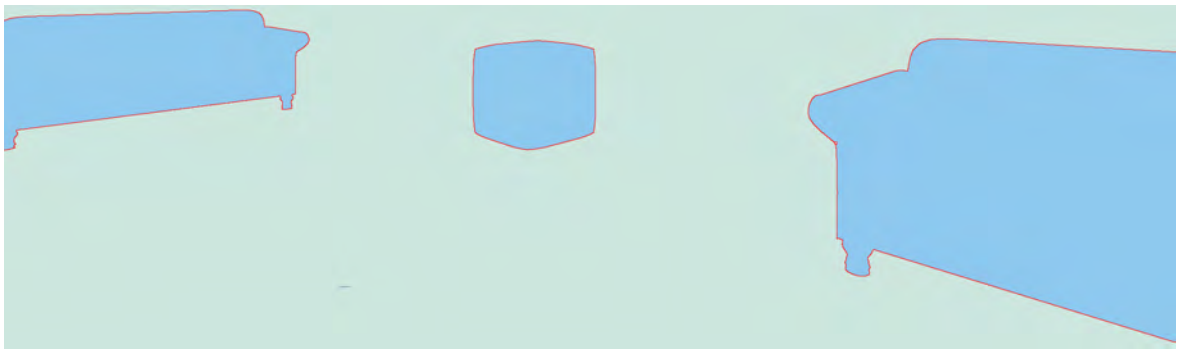


(c) Mapa de saliências. Em vermelho as saliências que compõem F. Em cinza a imagem residual V_R.

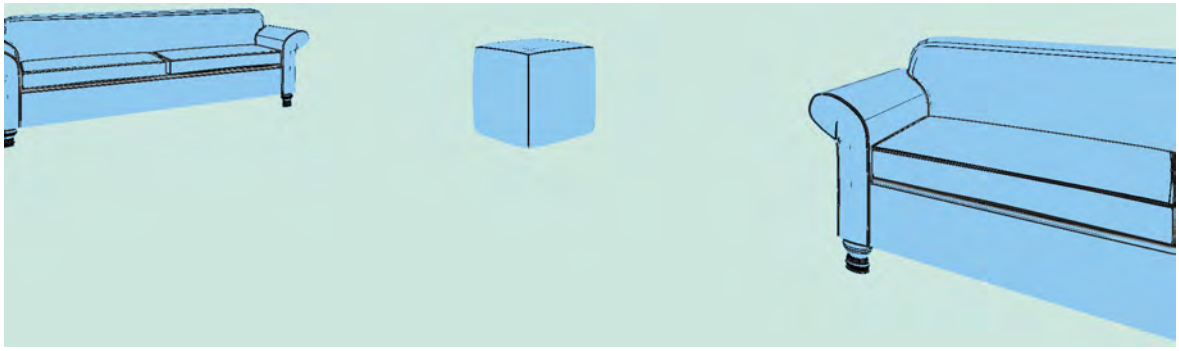
Figura 7-17: O mapa de normais (a) é usado para calcular os mapas de diferenças de normais (b). A partir dos mapas de diferenças (b) são criadas as imagens F e V_R (em (c)) com as quais construímos SAT_F e SAT_{V_R}. Os valores que superam o limiar angular formam a imagem de saliências F (traços vermelhos). Os valores restantes formam a imagem residual V_R. Nas imagens (b) e (c) alteramos a cor do fundo de preto para turquesa para dar contraste.



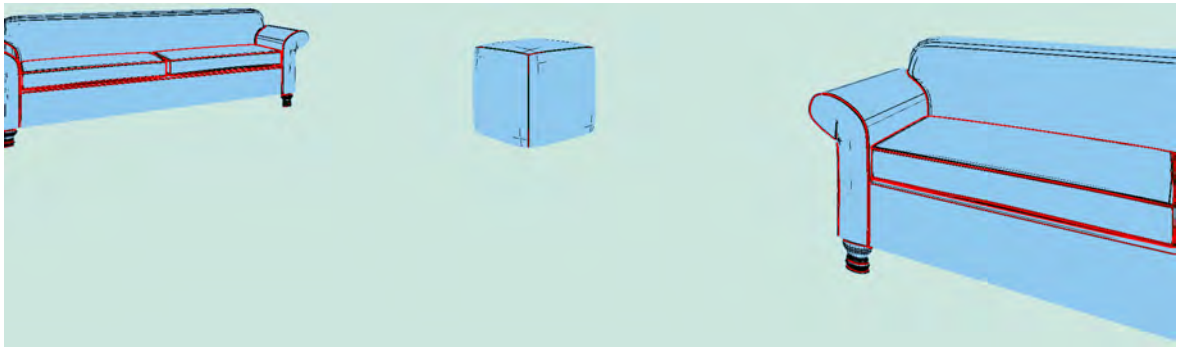
(a) Mapa parcial das normais da camada de objetos do *POMC*.



(b) Mapa parcial do domínio da camada de objetos do *POMC*. Em turquesa mais escuro o interior do domínio D, e em vermelho a fronteira do domínio B. D e B são usados para obter SAT_D e SAT_B.

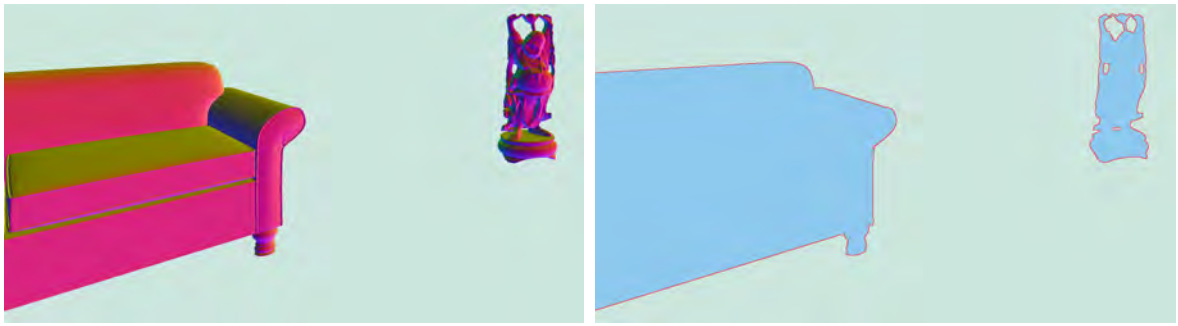


(c) Mapa parcial das diferenças de normais correspondente à figura (a).



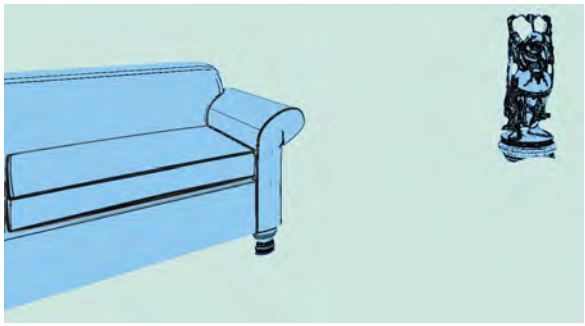
(d) Mapa parcial de Saliências. Vermelho: saliências que compõem F. Cinza: imagem residual V_R.

Figura 7-18: (c) mapa de diferenças de normais, calculado a partir das normais em (a). A partir de (c) são criadas as imagens F e V_R (em (d)) com as quais construímos SAT_F e SAT_{V_R}. Os valores que superam o limiar angular formam a imagem de saliências F (traços vermelhos). Os valores restantes formam a imagem residual V_R. Em cor turquesa claro regiões fora do domínio. A cor turquesa mais escura foi colocada no lugar da cor preta para dar contraste às figuras.

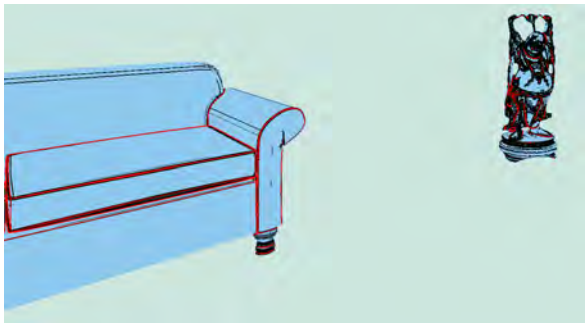


(a) Mapa de normais da camada de objeto.

(b) Mapa parcial do domínio da camada.



(c) Mapa parcial de diferenças de normais.



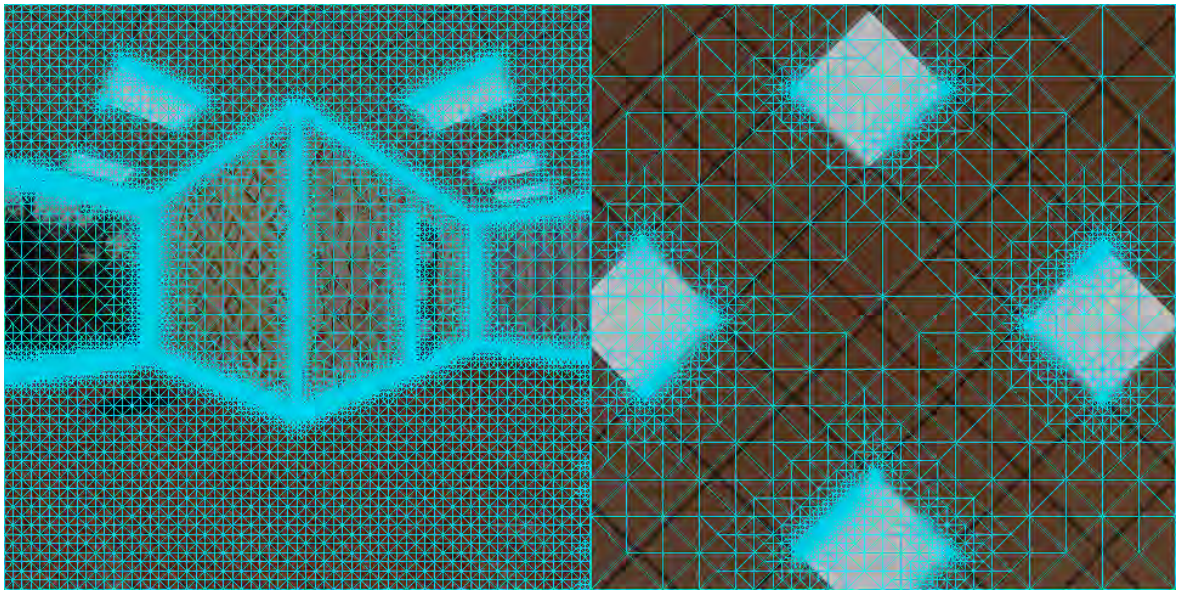
(d) Mapa de Saliências.

Figura 7-19: (c) mapa de diferenças de normais, calculado a partir das normais em (a). A partir de (c) são criadas as imagens F e V_R (em (d)) com as quais construímos SAT_F e SAT_{V_R}. Os valores que superam o limiar angular formam a imagem de saliências F (traços vermelhos). Os valores restantes formam a imagem residual V_R. Em cor turquesa claro marcamos regiões fora do domínio da camada. A cor turquesa mais escura foi colocada no lugar da cor preta para dar contraste às figuras.

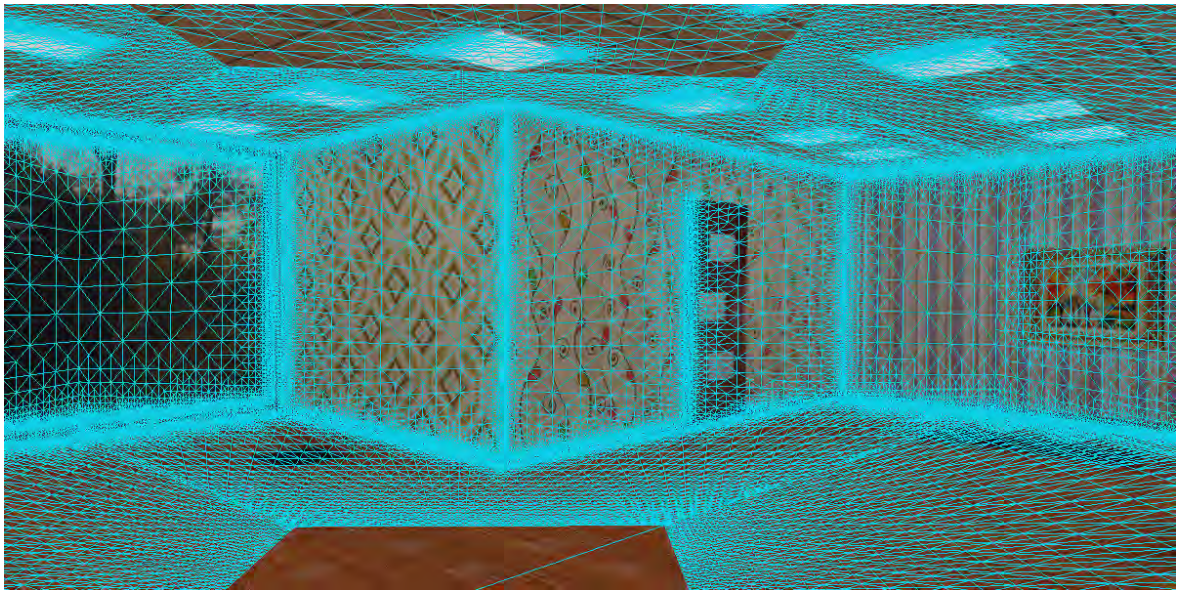
7.4.3 Resultados e comentários

Na última seção deste capítulo vimos uma forma de determinar uma malha a partir de uma análise da informação de profundidade do panorama. Esta estratégia pode ser utilizada também para gerar as malhas na reconstrução de cenas estudada no capítulo 5. No capítulo 5 construímos uma malha de ambiente por modelagem baseada na informação *RGB* do panorama e a calibração do mesmo, para finalmente gerar informação de profundidade via renderização. Nesta seção fizemos o caminho inverso, a partir de informação de profundidade recuperamos uma malha que representa a informação geométrica da cena.

Nas imagens das figuras (7-22), (7-23) e (7-24) é possível apreciar renderizações realizadas com os métodos de visibilidade propostos neste capítulo. Renderizamos a cena da sala desde duas posições: $C_0(0,0,1.15)$ e $C_1 = (-0.6,-0.6,1.15)$. A posição C_0 corresponde ao ponto de captura do *POMC*. Na figura (6-11) é possível apreciar vistas esquemáticas do ambiente capturado.

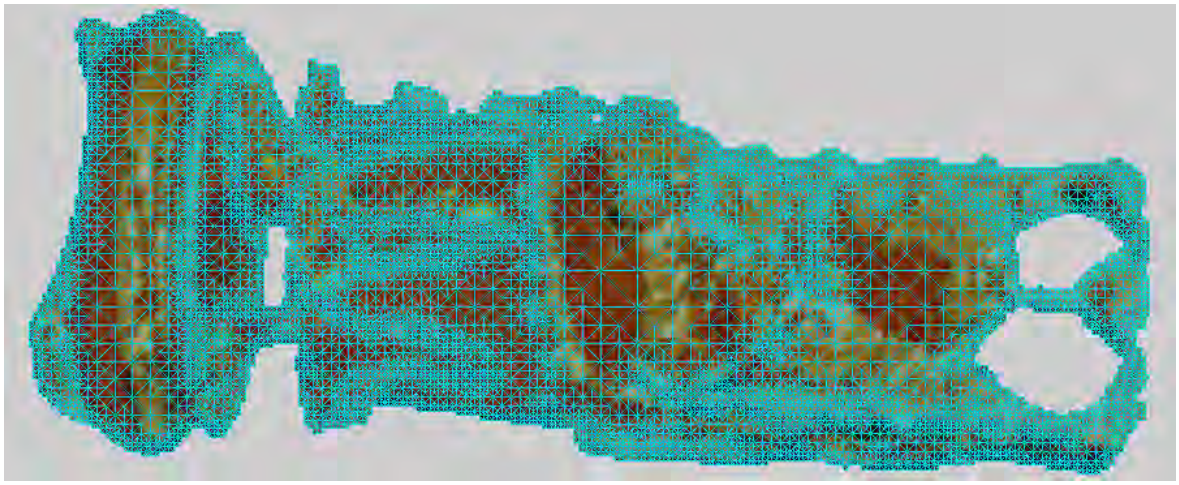


(a) Malha 2D de parte da camada do fundo. Ilustramos 2 faces ($+Y$ e $+Z$) do mapeamento de cubo que parametriza a camada do *POMC*.

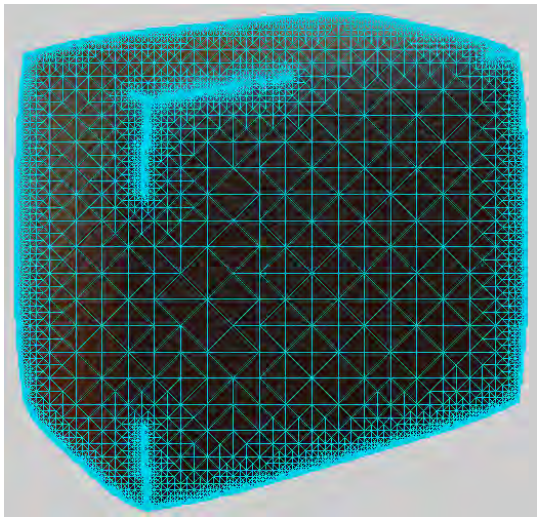


(b) Malha de triângulos 3D, $\mathcal{T}(h_i(N_i))$, obtida após transformar os pontos do domínio 2D para o mundo pela equação (6.11).

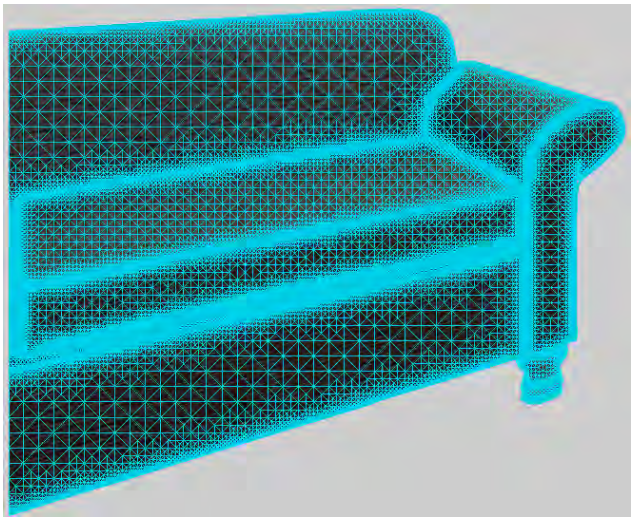
Figura 7-20: Processo de triangulação não uniforme da camada do fundo do *POMC*. (a) Obtemos uma malha 2D aplicando o método de subdivisão baseado em quadtree discreta. A função de subdivisão $\mathcal{S}$ foi implementada com os resultados fornecidos pela figura (7-17). (b) A malha 3D resultante de transformar os vértices da malha 2D (a) para sua posição no mundo.



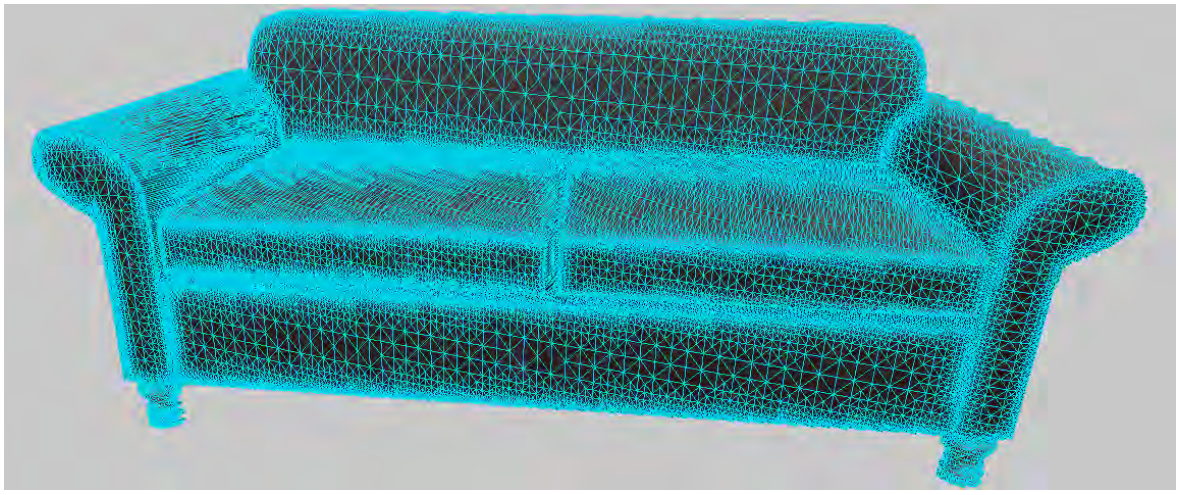
(a) Malha 2D de um dos Budas da camada da frente do *POMC*.



(b) Malha 2D de um Puff.



(c) Malha 2D de parte de um sofá.



(d) Triangulação 3D obtida para um dos sofás da camada de objetos do *POMC*.

Figura 7-21: Malhas obtidas pelo processamento da camada de objetos do *POMC* com uma quadtree. (a), (b) e (c) malhas no espaço de parametrização 2D da camada. Por razões de espaço mostramos um recorte das malhas de alguns objetos da camada. (d) malha 3D de um dos objetos da camada (sofá). Visualização gerada em MeshLab.

A câmera de visualização tem *FOV* horizontal de 60° e a resolução da imagem é de 960 × 540 pixels para todas as imagens geradas. Os algoritmos baseados em pontos utilizaram como dado de entrada as informações em uma resolução de 8192 × 4096 pixels, em todas as camadas e para todos os atributos. Nas visualizações de malhas regulares, utilizamos o mapeamento equiretangular, com as seguintes resoluções para os mapas:

- **camada de fundo:** *textura:* 8192 × 4096 pixels; *depth:* 2048 × 1024 pixels.
- **camada de objetos:** *textura:* 8192 × 4096 pixels; *depth:* 8192 × 4096 pixels.

Assim a malha da camada de fundo, com 2 triângulos por pixel gerados a partir da imagem de profundidade, tem uma resolução de 4190208 triângulos (na primeira e última linha da imagem geramos 1 triângulo por pixel, devido aos polos), enquanto que a camada de objetos tem uma malha com 1563887 triângulos. No total, temos pouco mais de 5.76milhões de triângulos.

Na visualização de malhas não uniformes, utilizamos o mapeamento de cubo, para realizar a subdivisão via quadtree discreta e gerar as malhas. Os mapas utilizados têm a seguinte resolução:

- **camada de fundo:** *textura:* 6144 × 4096 pixels; *depth:* 1536 × 1024 pixels.
- **camada de objetos:** *textura:* 6144 × 4096 pixels; *depth:* 6144 × 4096 pixels.

As malhas resultantes têm a seguinte resolução: camada de fundo com 190mil triângulos, e a camada de objetos com 274mil triângulos. No total, pouco mais de 464mil triângulos, i.e., 8% dos triângulos utilizados na malha uniforme.

Observamos que uma malha uniforme da camada de fundo par um mapa 8k equiretangular, tem pouco mais de 67milhões de triângulos. Fazendo uma malha não uniforme baseada em quadtree para um mapa de cubo com 6 faces de resolução 2048 × 2048 cada uma, obtemos uma malha de aproximadamente 1milhão de triângulos, i.e., 64 vezes menor que a malha uniforme.

Os detalhes aumentados da figura (7-24) permitem observar a qualidade dos resultados e os problemas de aliasing nas bordas das camadas. Nos exemplos com malhas nenhum tratamento de antialiasing foi realizado. Nos exemplos com renderização baseada em pontos, o antialiasing decorre do processo de filtragem onde foi aplicado um filtro bilinear de raio 1.

Nas malhas não uniformes, em geral temos que evitar o uso de triângulos muito grandes

em áreas onde a face do mapa de cubo não tem a normal alinhada com a face do triângulo. Fazemos isto para evitar problemas de distorção no mapeamento de textura quando movimentamos a câmera para uma posição diferente de C_0. De modo geral existem diversas soluções para este problema, entre elas:

- fazer um warping da textura para corrigir a deformação introduzida pela parametrização do mapa de cubo, e gerar uma nova textura;
- usar textura projetiva para realizar o warping na *GPU* [32];
- subdividir a triangulação para diminuir o efeito da distorção;
- usar um esquema de mapeamento bilinear [32];

Optamos por aumentar o numero de triângulos pois a quantidade de triângulos adicionais não foi substancial nos casos analisados. Outro problema de trabalhar com malhas de triângulos é a conformidade. No caso de mapa de cubo, geramos uma quadtree para cada face e logo uma malha por cada face do cubo. Assim torna-se necessário fazer um passo final de costura das 6 malhas para ter conformidade global. Outra opção, que será abordada no futuro, é a utilização de malhas simplificadas. Podemos usar a malha não uniforme da quadtree e fazer um processo de simplificação para juntar triângulos pequenos que estão sobre regiões planas mas foram subdivididos por causa das saliências (features) detectadas. Em construções feitas pelo homem, muitas vezes as saliências são linhas retas, de modo que é muito conveniente juntar os triângulos alinhados com a saliência e fazer um triangulo que tem um dos seus lados alinhado com a saliência. No caso de malhas simplificadas será importante um estudo mais detalhado da deformação do mapeamento de textura e será necessário explorar outra solução que não seja refinar a triangulação.

Como conclusão, podemos destacar que as soluções baseada em renderização de pontos e malhas triangulares mostraram resultados iniciais promissores. Para efeitos de visualização direta, sem efeitos adicionais de iluminação, implementações otimizadas de renderização baseada em pontos pode ser uma solução interessante, pela sua simplicidade. Por outra parte, as malhas oferecem um suporte geométrico bidimensional ao qual estamos acostumados e que praticamente todos renderizadores suportam.

(a) Amostragem direta com refinamento nas descontinuidades de profundidade.

(b) Amostragem adaptada à vizinhança 8-conectada, algoritmo 10.

(c) Malha uniforme gerada a partir do mapa equiretangular. 2 triângulos por pixel da imagem de *depth*.

(d) Algoritmo baseado em malhas não uniformes. Gerado com mapa de cubo, com resolução de 2048×2048 por face para texturas de cor em cada camada. Imagem de depth do fundo com 512×512 pixels e depth da camada de objetos com 2048×2048 pixels.

Figura 7-22: Comparação dos 4 métodos de visualização implementados. (a) e (b) Algoritmos baseados em pontos, com filtragem bilinear de raio 1. (c) e (d) Algoritmos baseados em malhas de triângulos. Em (c) as malhas são uniformes. Em (d) as malhas são não uniformes, geradas por subdivisão com quadtrees. As imagens da esquerda têm a câmara posicionada no centro do *POMC*, no ponto $(0, 0, 1.15)$. As imagens da direita tem a câmera deslocada, na posição $(-0.6, 0.6, 1.15)$.

(a) Amostragem direta com refinamento nas descontinuidades de profundidade.

(b) Amostragem adaptada à vizinhança 8-conectada, algoritmo 10.

(c) Malha uniforme gerada a partir do mapa equiretangular. 2 triângulos por pixel da imagem de *depth*.

(d) Algoritmo baseado em malhas não uniformes. Gerado com mapa de cubo, com resolução de 2048×2048 por face para texturas de cor em cada camada. Imagem de depth do fundo com 512×512 pixels e depth da camada de objetos com 2048×2048 pixels.

Figura 7-23: Comparação dos 4 métodos de visualização implementados. (a) e (b) Algoritmos baseados em pontos, com filtragem bilinear de raio 1. (c) e (d) Algoritmos baseados em malhas de triângulos. Em (c) as malhas são uniformes. Em (d) as malhas são não uniformes, geradas por subdivisão com quadtrees. As imagens da esquerda têm a câmara posicionada no centro do *POMC*, no ponto $(0, 0, 1.15)$. As imagens da direita têm a câmera deslocada, na posição $(-0.6, 0.6, 1.15)$.

(a) Amostragem direta com refinamento nas descontinuidades de profundidade.

(b) Amostragem adaptada à vizinhança 8-conectada, algoritmo 10.

(c) Algoritmo baseado em malhas uniformes.

(d) Algoritmo baseado em malhas não uniformes.

Figura 7-24: Comparação entre os métodos de visibilidade. As imagens correspondem à coluna direita da figura (7-22). A posição da câmera das mesmas é $C_1 = (-0.6, 0.6, 1.15)$. Nas imagens (a) e (b) a filtragem de amostras foi realizada com um filtro bilinear de raio 1. Do lado direito mostramos detalhes de regiões das imagens da coluna esquerda, que permitem observar melhor o resultado dos algoritmos nas regiões de descontinuidade.

Capítulo 8

Efeitos de iluminação *view-dependent* para *POMC*'s

Em este capítulo estudaremos o problema de incluir efeitos de iluminação *view-dependent* em renderizações e visualizações feitas utilizando um *POMC*. Na figura (8-1) é possível observar o que acontece se tentarmos aplicar um método de visualização desenvolvido no capítulo 7 diretamente sobre um *POMC* com todas as informações pré-calculadas. Como é de se esperar, a informação *view-dependent* da cena é mapeada de maneira incorreta se fizermos unicamente um remapeamento da geometria. Para minimizar estes problemas, neste capítulo discutiremos opções para implementar corretamente estes efeitos de iluminação dependentes da posição do observador.

Consideraremos em este capítulo um *POMC* com iluminação estática e com algumas informações pré-calculadas. Casos com variação temporal da iluminação também podem ser processados de maneira semelhante ao caso de iluminação estática. Vamos supor que as camadas contém além da informação de profundidade, informação da iluminação difusa e ambiente pré-calculada na camada. Cada camada também possui um mapa com as propriedades de reflexão da camada. A partir desta informação podemos implementar métodos para renderizar os efeitos de iluminação *view-dependent*. Mesmo considerando que a iluminação difusa já foi pré-calculada, os métodos propostos podem ser estendidos para resolver a iluminação difusa. Entretanto, em aplicações de visualização em tempo real, é necessário pré-calcular toda a informação possível, para direcionar o esforço computacional onde realmente for necessário.

(a) **(b)**

Figura 8-1: Renderização de duas vistas utilizando reflexão pré-calculada na origem do *POMC*. (a) Os reflexos são corretos pois a câmera está posicionada na origem do *POMC*. (b) A câmera deslocada da origem do *POMC* causa distorções visíveis quando é realizado um mapeamento com reflexões pré-calculadas (e.g., as paredes refletidas no piso da sala).

8.1 Renderização

Efeitos de iluminação *view-dependent* para visualização de *POMC* podem ser implementados através de diversos algoritmos de renderização. Cada algoritmo tem peculiaridades na forma de proceder para conseguir implementar os diferentes efeitos. O tipo de algoritmo usado também terá impacto no desempenho e na qualidade de efeitos que podem ser logrados. Nesta seção analisaremos alguns algoritmos de renderização e comentaremos como podemos implementar os efeitos *view-dependent* em cada um deles. Falaremos com maiores detalhes do algoritmo renderização diferida (*deferred rendering*) e uma versão híbrida com traçado de raios em *GPU*. Este algoritmo mostrou-se como a melhor opção para conseguir velocidade, efeitos realistas e simplicidade de uso com panoramas em camadas.

8.2 Deferred rendering

r*Deferred rendering* é uma técnica de renderização bastante explorada na última década, especialmente na área de jogos onde ganhou espaço entre os desenvolvedores (por exemplo Starcraft 2 e Crysis 2). Apesar de ter sido proposta em 1988 por Deering et al. [20], somente com o hardware gráfico atual é possível utilizá-la em tempo real em computadores pessoais.

A técnica de *deferred rendering* consiste em efetuar a renderização por etapas. Numa primeira etapa é realizada a extração de dados a partir da cena, como posição, normais, albedo difuso, índices de reflexão e outros. Estes dados são salvos em buffers (*G-Buffer*). Numa segunda etapa, para cada luz da cena é realizado o cálculo de iluminação utilizando as

informações do *G-Buffer*, e finalmente as informações são combinadas para obter a imagem final. A figura (8-2) ilustra o pipeline básico de um sistema de *deferred rendering*.

Uma descrição detalhada de como implementar *deferred rendering* em OpenGL pode ser encontrada no trabalho de Policarpo e Fonseca, [52]. Nos últimos anos surgiram muitos trabalhos que procuram melhorar o desempenho para *deferred rendering*. Uma boa fonte sobre o assunto são as notas do curso ministrado por Andrew Lauritzen [39] no *SIGGRAPH'2010*.

A seguir descreveremos o *G-Buffer* e a etapa geométrica do processo de *deferred rendering*, que serão usadas no sistema de renderização híbrido construído para implementar efeitos de iluminação view-dependent em *POMC*.

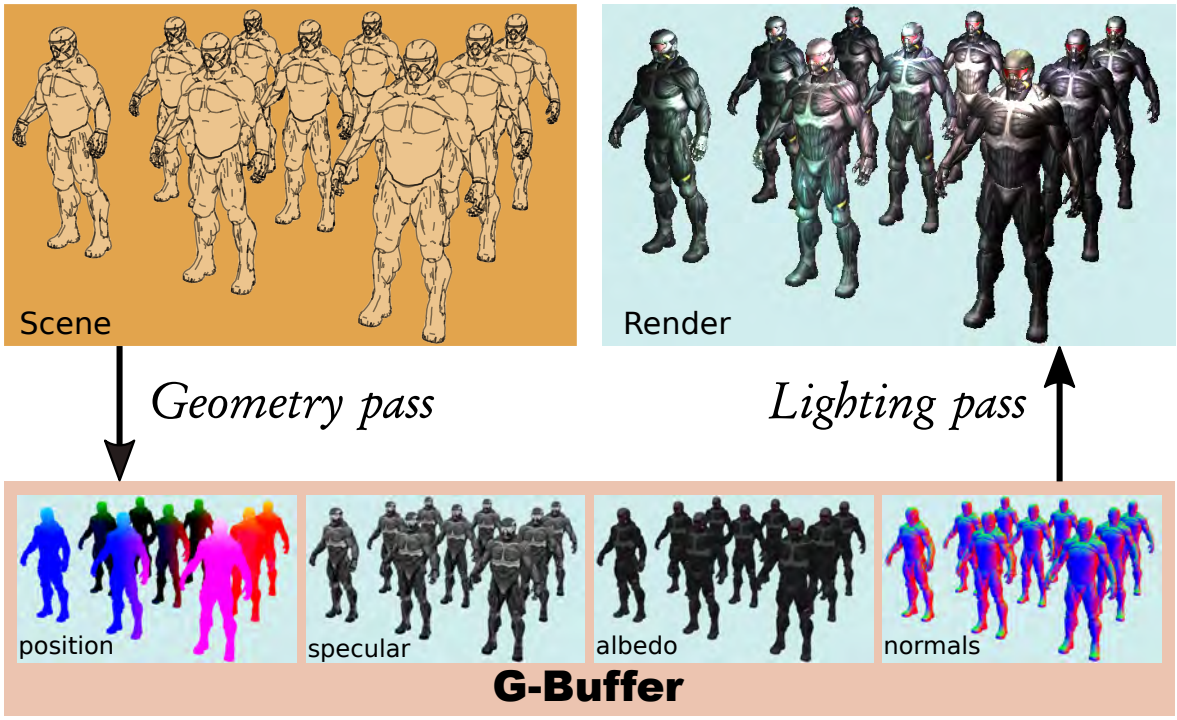


Figura 8-2: Pipeline básico do *deferred rendering*. A cena é processada e as informações resultantes armazenadas em texturas no *G-Buffer*. Num estágio posterior são efetuados os cálculos de iluminação para compor a renderização final.

8.2.1 G-Buffer

O *G-Buffer* (Geometric Buffer) é um conceito utilizado em *deferred rendering*. Consiste em um grupo de texturas 2D que guardam informações (posições, normais, cor e outros dados) para cada pixel de uma cena renderizada. O *G-Buffer* é gerado e preenchido por um shader especializado, que recebe como input as geometrias dos objetos (vértices, texturas, mapeamentos uvs, . . .), e ao invés de renderizar uma imagem final, são renderizadas diversas

imagens, cada uma contendo alguma das informações citadas anteriormente. Quando disponível, o uso de *MRT* (*Multiple Render Targets*) permite gerar o *G-Buffer* em único passo, enviando uma única vez as geometrias para a *GPU*.

As informações armazenadas no *G-Buffer* serão utilizadas posteriormente para o cálculo da iluminação. O tipo de informação que devemos armazenar dependerá tanto do modelo quanto dos efeitos de iluminação a ser implementados. Uma estrutura de *G-Buffer* típica contém buffers com a seguinte:

- **Buffer de posição:** armazena as posições dos fragmentos gerados;
- **Buffer de normais:** armazena a normal de cada fragmento gerado;
- **Buffer de albedo difuso:** armazena a cor do albedo de cada fragmento;
- **Buffer de índices de reflexão:** armazena valores para realizar o cálculo de reflexão;

• **Posição:** este componente que exige muita precisão. Uma opção é guardar uma terna (x, y, z) da posição no mundo (ou no espaço de câmera). Usar 3 valores de ponto flutuante de 32bits impacta consideravelmente na memória requerida para armazenar este buffer. Uma alternativa é comprimir a informação, armazenando apenas o elemento Z/W da posição no espaço de projeção em ponto flutuante de 32bits. Como a posição é representada em coordenadas homogêneas (x, y, z, w), é possível recuperar a posição 3D no espaço do mundo (ou de câmera) a partir do valor Z/W e sua posição na textura.

• **Normal:** a normal de um fragmento é utilizada na maioria dos algoritmos de iluminação. O valor da normal pode ser codificado no espaço de câmera ou no espaço do mundo, dependendo das necessidades. Tendo normalizado o vetor da normal, podemos armazenar x e y e calcular z em tempo de execução. Em algumas situações podemos codificar as normais com 8bits por canal, ou mais se considerarmos necessário. Na prática são necessários alguns cuidados adicionais. Por exemplo, se usarmos em OpenGL uma textura de 8bits por canal, é necessário converter o intervalo $[-1, 1]$ (valores das componentes da normal) para o intervalo $[0, 255]$ e ter cuidados na hora de recuperar os valores para obter a normal correta.

• **Albedo:** representa a cor difusa de cada pixel visto na tela, normalmente é armazenado em um buffer *RGBA*. A precisão dependerá do tipo de dados que estivermos utilizando. Em imagens *LDR*, 8 bits para cada canal são suficientes, porém quando trabalharmos com imagens *HDR* é necessário usar floats de 16bits por canal (ou mais).

Para o caso que pretendemos apresentar neste capítulo, a iluminação difusa foi pré-calculada, logo este buffer contém a iluminação difusa da imagem final em lugar do albedo.

• **Reflexão:** os parâmetros de reflexão dependem do algoritmo de iluminação e podem variar desde fatores como *Roughness/Specular* para o caso de modelos de iluminação como o Cook/Torrance [13], ou parâmetros como *Specular Intensity* e *Specular Power* para o clássico Blinn-Phong [5, 51].

O algoritmo 8.1 mostra um exemplo de como o *G-Buffer* é criado dentro de um contexto de OpenGL. São criadas todas as texturas necessárias nos formatos adequados com a resolução $(W \times H)$ da imagem final pretendida. Nas seções seguintes veremos como os shaders recebem e enviam os dados para o *G-Buffer*.

Algoritmo 8.1: Criação do G-Buffer em OpenGL.

```
void Context::GenTexture( GLuint *texId, GLint internalFormat, GLenum format,
          GLenum type, GLenum attachment )
{
  glGenTextures(1, texId );
  glBindTexture(GL_TEXTURE_2D, *texId);
  glTexImage2D(GL_TEXTURE_2D, 0, internalFormat, WIDTH, HEIGHT, 0, format, type, NULL);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
  glFramebufferTexture2D(GL_FRAMEBUFFER, attachment, GL_TEXTURE_2D, *texId, 0);
}
void Context::initGBuffer() {
  glGenFramebuffers(1, &gBuffer);
  glBindFramebuffer(GL_FRAMEBUFFER, gBuffer);
  // Position color buffer
  GenTexture(&gPosition, GL_RGB16F, GL_RGB, GL_FLOAT, GL_COLOR_ATTACHMENT0);
  // Normal color buffer
  GenTexture(&gNormal, GL_RGB16F, GL_RGB, GL_FLOAT, GL_COLOR_ATTACHMENT1);
  // Color + Specular color buffer
  GenTexture(&gAlbedoSpec, GL_RGBA16F, GL_RGBA, GL_FLOAT, GL_COLOR_ATTACHMENT2);
  // Tell OpenGL which color attachments we'll use (of this framebuffer) for rendering
  GLuint attachments[3]={GL_COLOR_ATTACHMENT0,GL_COLOR_ATTACHMENT1,GL_COLOR_ATTACHMENT2};
  glDrawBuffers(3, attachments);
  // Create and attach depth buffer (renderbuffer)
  GLuint rboDepth;
  glGenRenderbuffers(1, &rboDepth);
  glBindRenderbuffer(GL_RENDERBUFFER, rboDepth);
  glRenderbufferStorage(GL_RENDERBUFFER, GL_DEPTH_COMPONENT, WIDTH, HEIGHT);
  glFramebufferRenderbuffer(GL_FRAMEBUFFER,GL_DEPTH_ATTACHMENT,GL_RENDERBUFFER,rboDepth);
  glBindFramebuffer(GL_FRAMEBUFFER, 0);
}
```

8.2.2 Etapa Geométrica

A etapa geométrica usa as informações dos modelos da cena, tais como vértices, índices e texturas. A informação de entrada pode ser bem variada, dependendo da composição e complexidade da cena. Assim, cada objeto da cena é processado sucessivamente pelo Vertex e Fragment Shader e são determinados os fragmentos do objeto visíveis na tela. Para cada fragmento visível, no Fragment Shader são geradas as informações que devem ser armazenadas no *G-Buffer*.

O algoritmo 17 mostra qual é o processo geral da etapa geométrica. Nos algoritmos 8.2 e 8.3 apresentamos um exemplo de como o processamento é realizado através dos shaders.

O *Vertex Shader* recebe as informações dos vértices dos objetos da cena e os processa. A posição e as normais são transformadas ao espaço e formato que queremos usar no *G-Buffer*. No código de exemplo do algoritmo 8.2 a posição e a normal estão no espaço do mundo tanto nos dados de entrada quanto na saída do shader (`vsPosition`, `vsNormal`). Também são enviadas para a saída as coordenadas de textura (`vsTextCoords`).

Os dados de saída do Vertex Shader (`vsPosition`, `vsNormal`, `vsTexCoords`) são usados como entrada do *Fragment Shader*, que pode efetuar alguns cálculos adicionais que sejam considerados necessários. Durante o processamento o teste de profundidade (Depth test do OpenGL) está ativado, portanto no estágio do *Fragment Shader* somente são processados os fragmentos visíveis. A textura `gAlbedoSpec` tem 4 canais. Nos 3 primeiros canais é armazenada a cor difusa do fragmento, obtida a partir da textura `textureDiffuse` do objeto através das coordenadas de textura `TextCoords`. De igual forma, é calculada a partir da textura `textureSpecular` o índice de especularidade do fragmento, e o valor é armazenado na quarta componente da textura `gAlbedoSpec`.

Algorithm 17 Etapa geométrica

1: inicializa o Depth Buffer $Z \leftarrow MAX\_VAL$;
2: **for** $objeto \in Scene$ **do**
3: **for** fragmento f de $objeto$ **do**
4: $pz \leftarrow$ depth do fragmento $f(x,y)$;
5: $(x,y) \leftarrow$ posição do fragmento f na tela;
6: **if** $pz < Z(x,y)$ **then**
7: $GBuffer(x,y) \leftarrow (position, normal, albedo, specular, etc.)$ do fragmento f;
8: **end if**
9: **end for**
10: **end for**

Algoritmo 8.2: Vertex Shader para popular o G-Buffer

```
#version 330 core
layout (location = 0) in vec3 position;
layout (location = 1) in vec3 normal;
layout (location = 2) in vec2 texCoords;
out vec3 vsPosition;
out vec3 vsNormal;
out vec2 vsTexCoords;
uniform   mat4 MVPMatrix;

void main() {
    gl_Position = MVPMatrix * vec4(position, 1.0f);
    vsPosition = position;     //in world coordinates
    vsNormal = normal;         //normal in world space
    vsTexCoords =  texCoords;  //texture coordinates
}
```

Algoritmo 8.3: Fragment Shader para popular o G-Buffer

```
#version 330 core
layout (location = 0) out vec3 gPosition;    //GBuffer position
layout (location = 1) out vec3 gNormal;      //GBuffer normal
layout (location = 2) out vec4 gAlbedoSpec;  //GBuffer diffuse color + specular parameter

in vec3 vsPosition;
in vec3 vsNormal;
in vec2 vsTexCoords;

uniform vec3 viewPos;
uniform sampler2D textureDiffuse;
uniform sampler2D textureSpecular;

void main() {
    gPosition = vsPosition;      // fragment position in the first Gbuffer  texture
    gNormal = normalize(Normal); // store the per-fragment normals into the Gbuffer
    gAlbedoSpec = texture(textureDiffuse, TexCoords); // diffuse per-fragment color
    // Store specular intensity in gAlbedoSpec's alpha component
    gAlbedoSpec.a = texture(textureSpecular, TexCoords).r;
}
```

Em aplicações que envolvem o uso de panoramas em camadas (*POMC*) podemos obter alguma vantagem adicional nos processamentos. De modo geral, os movimentos de câmera são próximos ao ponto de origem do panorama, portanto ter as camadas organizadas pela ordem de visibilidade pode minimizar os cálculos necessários no *Fragment Shader*, desde que os objetos (camadas) sejam processados na ordem das mais próximas para as mais distantes.

8.3 Ray tracing

O método de Ray tracing é uma das técnicas mais poderosas de rendering, pois além de ser um algoritmo de visualização, pode resolver de forma efetiva e relativamente simples vários fenômenos luminosos como sombras, reflexão e refração.

A ideia de utilizar o algoritmo de *ray tracing* para o cálculo da iluminação foi sugerida por Whitted [57] como uma extensão ao algoritmo de visualização de *ray-casting* utilizado para determinar as superfícies visíveis da cena, discutido no capítulo 7. Quando geramos uma imagem por *ray tracing* a intensidade de cada ponto da imagem é calculada lançando um raio em direção ao ponto, desde a posição da câmera. Na primeira interseção deste raio com os objetos da cena calculamos a intensidade luminosa utilizando algum modelo de iluminação local. A contribuição luminosa de cada fonte é computada se o ponto do objeto for visível (i.e., não estiver na sombra). Isto é determinado lançando um raio desde o ponto em questão em direção a cada fonte, e verificando se ele não é interceptado por outros objetos. Se a superfície for especular o raio refletido é seguido de modo de capturar a radiância vinda de outras superfícies. Se a superfície for translúcida procedemos de forma similar com o raio refratado.

O *ray tracing* é a base da maioria dos sistemas de renderização baseados em física. Nos capítulos 4 e 5 já utilizamos o algoritmo de path tracing para resolver o problema de iluminação global em cenas de realidade mista. Os métodos de *ray tracing* foto-realistas não são indicados para propósitos de renderização em tempo real, mas podem ser úteis para gerar resultados offline, como cálculo de mapas de reflexão e de iluminação difusa, que podem ser usados em renderizações em tempo real. Panoramas omnidirecionais com múltiplas camadas (*POMC*) que contenham informação de radiância em *HDR* podem ser utilizados como mapas de iluminação em algoritmos de *ray tracing* foto-realistas. Por exemplo, podemos usar um *POMC* no lugar de um mapa de iluminação *RGB-D* no algoritmo apresentado no capítulo 5.

Para obter maiores detalhes sobre sistemas de renderização baseados em ray tracing, o leitor pode consultar o livro *"Physically Based Rendering: from theory to implementation"* de M. Pharr e G. Humphreys [48, 50].

8.4 Rendering híbrido: deferred rendering + ray tracing

Sabemos que os algoritmos baseados em rasterização têm dificuldades e limitações para produzir efeitos de iluminação global. Em geral, estes efeitos são melhores e mais fáceis de conseguir com algoritmos baseados em ray tracing, com a desvantagem de um custo computacional mais elevado. Por estes motivos decidimos utilizar uma solução intermediaria, que utilize o melhor dos dois mundos.

O renderizador híbrido proposto funciona da seguinte forma: o método de *Deferred rendering* será utilizado para gerar as informações dos fragmentos visíveis e armazena-los no *G-Buffer*. Seguidamente aplicamos um método de *ray tracing* utilizando as informações contidas no *G-Buffer*. O método de *ray tracing* lança um raio $r(o,d)$ para cada fragmento f codificado no *G-Buffer*. O raio $r(o,d)$ tem origem o igual à posição do fragmento no mundo e direção d igual à direção de reflexão ideal ou de refração correspondente ao fragmento f.

O *G-Buffer* calculado pela primeira etapa do *deferred rendering* contêm as informações que correspondem à primeira interseção dos raios que partem da câmera com a cena quando usamos um ray tracing. Assim, ao aplicar o ray tracing, não é necessário fazer o seguimento dos raios desde a câmera, pois já conhecemos a primeira interseção dos mesmos. Portanto continuamos traçando os raios a partir da posição correspondente à primeira interseção, para procurar a segunda interseção com a cena. Esta continuação do raio corresponde ao processo de reflexão ou refração de um raio após a primeira interseção. Para encontrar a segunda interseção de cada novo raio, é necessário que a cena seja conhecida pelo sistema de ray tracing, a fim de poder testar as interseções do raio com os objetos.

Para validar esta ideia antes de pesquisar implementações mais eficientes, construímos uma implementação de *ray tracing* que utiliza uma estrutura de aceleração *BVH* (Bounding Volume Hierarchies) e é implementada em *Fragment Shader* no *OpenGL*.

Como já mencionamos previamente, a primeira interseção de raios do *ray tracing* não é calculada com ray tracing, obtemos esses valores através do *G-Buffer* da implementação diferida. A partir destes pontos traçamos os raios de reflexão e refração. À diferença de um traçado de raios estocástico, onde a direção de reflexão é escolhida de maneira probabilística, escolhemos unicamente a direção de reflexão ideal r. Procuramos a interseção do raio r com a cena. Uma vez determinada a interseção, buscamos no mapa de reflexão correspondente ao ponto intersectado o valor a ser usado para o cálculo de iluminação. Da mesma forma que no *ray tracing* recursivo, é possível calcular reflexões sucessivas se continuarmos a rebater o

raio na cena. Em geral, quando o raio não intersecta um objeto com índice de especularidade alto, não é necessário realizar mais de uma interseção para ter resultados realistas.

8.4.1 Estrutura de aceleração: BVH

Um dos aspectos centrais ao construir um algoritmo de *ray tracing* e a escolha de um bom método de aceleração, pois sabemos que o maior tempo de cômputo é gasto realizando testes de interseção, logo uma estrutura de dado que permita navegar de maneira eficiente é um ponto chave para conseguir eficiência no método.

Fizemos alguns testes em CPU das estruturas Kdtree e BVH com os dados de cena extraídos de panoramas. Os resultados de desempenho forma praticamente os mesmos para ambas as estruturas. Assim, devido à maior facilidade de implementação, escolhemos implementar em Shader a estrutura BVH.

Usamos uma estrutura de dados BVH como estrutura de aceleração para o algoritmo de *ray tracing* que resolverá o problema de reflexão e refração da cena. O processamento e construção da estrutura BVH usada é inspirada na apresentada no livro de M. Pharr e G. Humphreys, [50].

A classe BVHAccel que implementa a estrutura de dados BVH contém as seguintes informações:

$$BVHAccel = \begin{cases} L_T & : \text{lista dos primitivos da cena ordenados de acordo à BVH;} \\ BVH_L & : \text{lista de nós da estrutura onde está a informação da BVH;} \end{cases}$$

A estrutura de dados BVH é montada na CPU através da classe BVHAccel. A função que implementa a estrutura de dados BVH funciona assim:

- Recebe como input a cena;
- A partir de todos os primitivos da cena, é gerada uma estrutura BVH da cena;
- São preenchidas as listas L_T e BVH_L;

A lista L_T é um vetor que contém todos os triângulos da cena, assim o valor $L[i]$ indica o i-ésimo triângulo da lista. Cada triângulo T conhece as informações inerentes, tais como posição, normais e as respectivas coordenadas de textura de cada um dos seus 3 vértices. Com estas informações podemos acessar outros parâmetros em texturas associadas com os

objetos da cena. Na prática L_T contém ponteiros para os respectivos triângulos na malha.

A estrutura BVH_L é um array de nós N, assim $BVH_L[i]$ indica o i-ésimo nó. Um nó representa uma caixa no espaço (bounding box), alinhada com os eixos. Um nó N contém as seguintes informações:

$$BVHNode = \begin{cases} p_{min} & \text{: vértice inferior da caixa que delimita o nó;} \\ p_{max} & \text{: vértice superior da caixa que delimita o nó;} \\ axis & \text{: eixo de subdivisão do nó } (x=0,\ y=1,\ z=2); \\ np & \text{: número de primitivos dentro da caixa (total ou parcialmente),} \\ & \quad \bullet\ np > 0 \Leftrightarrow \text{o nó é uma folha da árvore.} \\ offset & \text{: índice de posição. Tem dois usos:} \\ & \quad \bullet\ \text{Se } np = 0 : \text{posição do segundo nó filho do nó;} \\ & \quad \bullet\ \text{Se } np > 0 : \text{posição do } 1^{\circ} \text{ primitivo do nó na lista } L_T. \end{cases}$$

É necessário mencionar que um nó com $np = 0$ sempre será um nó interno, i.e., que não é folha. Quando $np = 0$ automaticamente o valor da variável $offset$ representa a posição de um dos filhos do nó na lista BVH_L. Por outra parte quando $np > 0$ o nó é uma folha e os np primitivos associados podem ser determinados procurando na lista L_T, $L_T[offset]$ até $L_T[offset + np - 1]$.

A árvore original da estrutura BVH é achatada e linearizada para formar o array BVH_L. Os nós da árvore são achatados seguindo uma ordem de profundidade, de modo que o primeiro filho de cada nó sempre está na posição seguinte do array. Já o segundo filho pode estar em outra posição mais distante. Devido a esta organização, somente é necessário armazenar o offset do segundo filho, pois sempre sabemos que o primeiro filho encontra-se na seguinte posição (k+1) do array de nós. A figura (8-3) ilustra o esquema de achatamento e como é a organização dos nós no array.

Veremos agora como codificar a informação para envia-la ao Fragment Shader, onde queremos usá-la. A forma de fazer isto é codificar a informação da BVH e da geometria da cena em texturas que serão passadas ao Fragment Shader. tratar de passar arrays de dados como normalmente é feito quando enviamos as malhas para o Vertex Shader não funciona, pois o FS não recebe estes dados que já são processados no estágio do VS.

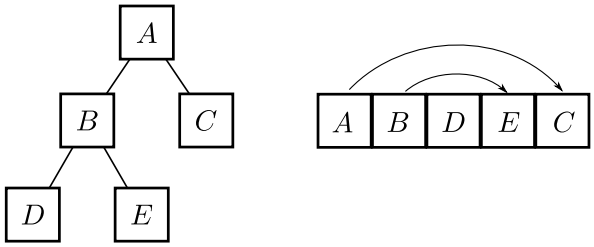


Figura 8-3: Layout da estrutura linear de uma *BVH* na memória. Os nós da *BVH* são armazenados na memória numa ordem que prioriza a profundidade (esquerda). Assim, para qualquer nó interior da árvore (*A* e *B* no exemplo), o primeiro nó filho está armazenado na memória imediatamente depois do pai. O segundo nó filho é encontrado através de um valor de offset, que indica o deslocamento a ser realizado na memória para encontrá-lo. Os nós folha não têm filhos (*D*, *E* e *C*).

Usaremos os dados das listas BVH_L e L_T e as vamos organizar em texturas que depois serão passadas ao FS onde será realizado o processo de ray tracing.

Criaremos 4 texturas bidimensionais:

- M_{vnt} : contêm todos os vértices $v = (x, y, z)$, normais $n_v = (n_x, n_y, n_z)$ e coordenadas de textura $t_v = (u, v)$;
- M_e : contêm todos os triângulos $e = (v_1, v_2, v_3)$, os v_i's indicam a posição dos vértices do triângulo na textura M_{vnt};
- M_b : contêm as caixas dos nós da BVH. Cada caixa é determinada por dois vértices opostos p_{min} e p_{max};
- M_d : contêm informações dos nós da BVH. Os valores inteiros $offset$, np e $axis$ estão nesta textura.

Fixamos um valor de largura horizontal W para todas as texturas. Este valor pode ser um valor fixo $W = 1000$ por exemplo, ou podemos fazer algum cálculo sobre a quantidade de dados para determinar um valor que julguemos apropriado.

Na textura M_e, cada elemento e tem 3 coordenadas inteiras, portanto esta textura é do tipo RGB32UI (3 canais de unsigned int de 32bits). Assim, o pixel na posição (i, j) de M_e guarda 3 valores inteiros v_1, v_2 e v_3 nos canais R, G e B do pixel respectivamente.

Na textura M_{vnt} temos 2 elementos com 3 componentes e 1 elemento com 2 componentes. No total são 8 componentes. Como as texturas têm no máximo 4 canais (RGBA), usaremos uma textura RGB32FLOAT (3 canais float de 32bits) e operamos da seguinte forma: colocamos primeiro todos os vértices v de forma que suas 3 componentes x, y e z ocupem os canais RGB de cada pixel. Assim, um vértice estará associado com um pixel na textura. A

seguir são alocadas as normais n (com 3 componentes) da mesma forma que fizemos com os vértices. As normais serão alocadas a partir a primeira linha horizontal livre após o último vértice. Suponhamos que o último vértice v foi alocado na posição (i, j), então as normais serão alocadas a partir da posição $(0, j+1)$. Por último alocaremos as coordenadas de textura, começando na posição $(0, k+1)$, onde k foi a linha onde alocamos a última normal. No caso das coordenadas de textura, somente usaremos os canais R e G do pixel.

Vemos que dada a dimensão W de largura da textura, a altura de M_{vnt} será $H = 3 \cdot \lceil \frac{\#vertices}{W} \rceil$, e o tamanho do offset vertical para cada bloco de dados é $\lceil \frac{\#vertices}{W} \rceil$.

A textura M_b é do tipo RGB32FLOAT, pois precisa alocar informação de posição espacial. Como temos que alocar os 2 vértices da caixa que determina cada nó, e cada vértice têm 3 componentes, fazemos uma distribuição como a usada na textura M_{vnt}. Alocaremos primeiro todos os vértices p_{min}, e na sequência alocaremos todos os vértices p_{max}. Desta forma a textura terá um tamanho horizontal W e vertical $H = 2 \cdot \lceil \frac{\#nodes}{W} \rceil$, e o offset vertical para acessar os dados p_{max} será de $\lceil \frac{\#nodes}{W} \rceil$.

A textura M_d tem por cada pixel 3 valores inteiros, $offset$, np e $axis$, alocados nos canais R, G e B do pixel. Assim esta textura tem um tamanho horizontal W e vertical $H = \lceil \frac{\#nodes}{W} \rceil$. A textura é do tipo RGB32UI como a textura M_e.

A figura (8-4) ilustra graficamente a estrutura das texturas e como as informações estão interligadas. A partir da ilustração é possível ter uma ideia de como será o processo de buscas para realizar interseção de raios.

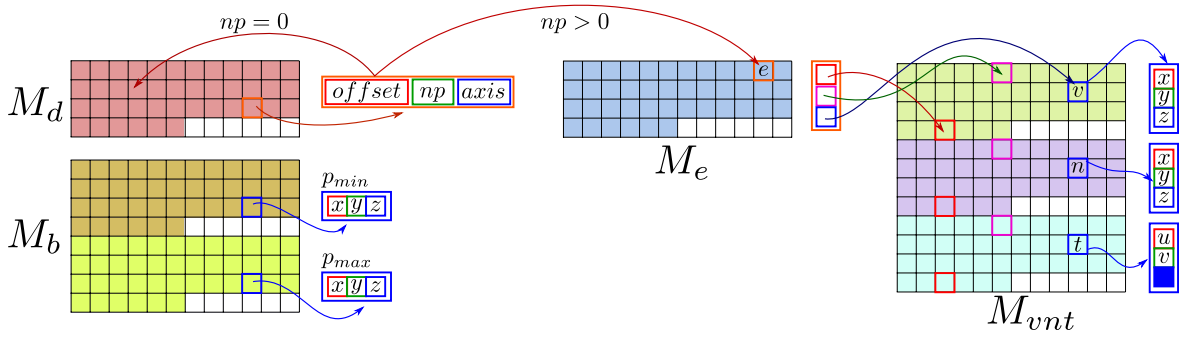


Figura 8-4: Texturas M_d, M_b, M_e e M_{vnt} utilizadas para carregar a cena e a estrutura *BVH* no *Fragment Shader*. Dado um nó (em laranja), o $offset$ permite procurar o segundo filho do nó (se $np = 0$), ou encontrar o primeiro triângulo que pertence ao nó (se $np > 0$). p_{min} e p_{max} delimitam a caixa do nó. A partir do elemento e na textura M_e, procuramos pela posição os 3 vértices do triângulo em M_{vnt}. As informações v (vértice), n (normal) e t (textura) estão distribuídas com um offset fixo.

8.4.2 Ray tracing em Fragment Shader

Até o momento temos os dados da primeira interseção dos raios partindo da câmera, calculados pelo método de *deferred rendering* e armazenados no *G-Buffer*. Também criamos as texturas necessárias para realizar um ray tracing. Agora é hora de enviar os dados para a GPU e realizar só cálculos necessários para adicionar os reflexos da cena à iluminação difusa já armazenada no *G-Buffer*.

O processo será realizado com um Quad rendering, isto é, passaremos para o Vertex Shader uma câmera genérica e um quad $[-1,-1] \times [1,1]$ com coordenadas de textura $[0,0] \times [1,1]$. Como o *quad* cobre a tela o Vertex Shader vai rasterizar e gerar os fragmentos. Não estamos interessados na sua posição , mas sim nas coordenadas de textura dos fragmentos, que são passadas do VS para o FS. No FS as coordenadas de textura são usadas para acessar o *G-Buffer* e extrais as informações de posição, normal, e cor difusa do fragmento correspondente à posição na textura. Para evitar problemas de interpolação, utilizamos o *G-Buffer* no modo GL_NEAREST. Os algoritmos 8.4 e 8.5 ilustram como o processo é realizado no VS e FS respectivamente.

A implementação das funções de interseção *raio/triângulo* e *raio/bounding-box* são semelhantes às encontradas nos livros especializados em sistemas de ray tracing.

Cada fragmento gerado pela rasterização do *quad* está identificado com um dos fragmentos que foi armazenado previamente no *G-Buffer*. Obtemos a posição p e a normal n armazenados no *G-Buffer* e construímos um raio $R(p,d)$ a partir do ponto p na direção de reflexão ideal

$$d = \frac{v - 2 \cdot \langle v,n \rangle \cdot n}{\|v - 2 \cdot \langle v,n \rangle \cdot n\|}, \tag{8.1}$$

onde $v = p - eye$.

A função $intersect\_ray\_bvh(R, isect)$ usa o raio $R(p,d)$ e procura sua a primeira interseção do mesmo com a cena e guarda as informações necessárias na classe *isect*. A classe *isect* armazena o parâmetros da interseção mais próxima. A partir dos parâmetros devolvidos pela variável *isect* é possível determinar a posição q da interseção, a camada do *POMC* que foi intersetada, e as coordenadas de textura correspondentes ao ponto de interseção na camada.

Algoritmo 8.4: Vertex Shader para *ray tracing* a partir dos dados do G-Buffer

```
#version 330 core
layout (location = 0) in vec3 position;
layout (location = 1) in vec2 texCoords;
out vec2 TexCoords;
void main() {
    gl_Position = vec4(position, 1.0f);
    TexCoords = texCoords;
}
```

Algoritmo 8.5: Fragment Shader para *ray tracing* a partir dos dados do G-Buffer

```
#version 330 core
in vec2 TexCoords;                  // quad texture coord from vertex shader
out vec4 FragColor;                 // output color
// ---------- G-BUFFER textures ----------------------------------------------
uniform sampler2D gPosition;        // 0- fragment position in world space
uniform sampler2D gNormal;          // 1- fragment normal in world space
uniform sampler2D gAlbedoSpec;      // 2- fragment diffuse illumination
// ---------- Mesh Structure textures ----------------------------------------
uniform sampler2DRect vertices;     // 3- (float) vertex, normal, uv  -> Mvnt
uniform sampler2DRect triangles;    // 4- (uint) triangles indices    -> Me
// ---------- BVH Structure textures -----------------------------------------
uniform sampler2DRect bvhBBounds; // 5- (float) nodes bounding boxes -> Mb
uniform sampler2DRect bvhTree;      // 6- (uint) offset, np, axis data -> Md
// ---------- POMC textures --------------------------------------------------
uniform sampler2D Panorama;         // 7- (float) POMC diffuse texture

\\ ... ALL INTERSECTION AND DATA NAVIGATION METHODS (NOT INCLUDED)...
void main()
{
  FragColor = vec4( texture2D(gAlbedoSpec, TexCoords).rgb, 1.0 );
  float SpecularFactor = texture2D(gAlbedoSpec, TexCoords).a;

  if( SpecularFactor > 0.1 )
  {
    Ray R = ray( texture2D(gPosition, TexCoords).rgb, texture(gNormal, TexCoords).rgb);
    Intersection isect;

    if( intersect_ray_bvh( R, isect ) )
    {
      if( SpecularFactor < 0.95 )
          SpecularFactor /= (1.0 + R.maxt*R.maxt);
      FragColor += vec4( SpecularFactor * texture2D(Panorama, isect.texcoord).rgb , 1.0);
    }
  }
}
```

Quando usamos um fator de atenuação baseado em distância para mapear a reflexão, podemos obter a distância $\|q-p\|$ pois temos parâmetros necessários. Se desejarmos podemos incluir atenuações e outros efeitos baseados no ângulo da reflexão, que podem ser obtidas usando o parâmetro $\langle v/\|v\|, n\rangle$.

Se tivermos acesso a mapas onde foram pré-calculados efeitos de rugosidade podemos simular os efeitos de reflexões sobre objetos metálicos por exemplo. Na seção 8.5.3 mostramos como é realizada a filtragem de mapas omnidirecionais para simular reflexões difusas.

Como foi mencionado, é possível calcular reflexões sucessivas. A cada cálculo de reflexões sucessivas podemos incluir termos de atenuação semelhantes aos utilizados no algoritmo de *path tracing* para simular o espalhamento da luz. Os resultados que obtivemos pelo método de renderização híbrido são apresentados na seção 8.6.

8.4.3 Considerações adicionais

Algumas observações podem ser feitas sobre o processo realizado. Como as estruturas de aceleração são utilizadas para realizar buscas na cena, i.e., procurar os possíveis primitivos com os quais um raio intersecta, é necessário fazer isto de maneira eficiente. Em geral, estruturas de busca binária consomem um tempo de $O(log(n))$ para encontrar uma folha da estrutura, isto supondo que temos uma árvore balanceada. Uma forma em que podemos reduzir este tempo é logrando diminuir a complexidade da cena. Como estamos usando o *ray tracing*somente com o propósito de determinar reflexões sobre os objetos, notamos que de maneira geral não é necessário ter uma estrutura geométrica tão detalhada da cena para obter um comportamento visualmente correto das reflexões. Isto porque em geral boa parte da cena acaba sendo refletida sobre um objeto relativamente pequeno, de forma que áreas de tamanho considerável na cena podem acabar sendo mapeadas em poucos pixels no reflexo (ou refração). Assim, nestas situações a utilização de malhas simplificadas pode reduzir o custo computacional sem comprometer q qualidade final. A simplificação traz 2 benefícios: uma redução no uso de memória da placa gráfica para alocar e fazer buscas nas texturas que contêm os dados da cena e uma aceleração no percorrido da estrutura devido à redução de tamanho.

8.5 Efeitos de reflexão para *POMC* usando rasterização

Efeitos de reflexão, refração e sombras, são exemplos de efeitos de iluminação global, nos quais um objeto da cena interfere na iluminação de outro objeto. No caso de uma cena estática, os efeitos de sombras são estáticos e independem da posição do observador. No entanto os efeitos de reflexões e refrações variam de acordo com a posição do observador. Os efeitos de reflexões e refrações são importantes porque dão à renderização um maior grau de realismo e permitem que o observador possa entender os objetos da cena, suas relações e distribuições espaciais. A seguir veremos como podemos a partir de um *POMC* com iluminação difusa pré-calculada, e com suas propriedades de reflexão estabelecidas, calcular mapas de reflexões para produzir reflexões *view-dependent*.

8.5.1 Mapeamento de reflexão

A técnica de mapeamento de reflexão (*reflection mapping*), também conhecida como *environment mapping*, foi introduzida em 1976 por Blinn e Newell [6]. Trata-se de uma técnica simples e poderosa para gerar reflexões aproximadas sobre superfícies curvas.

A ideia do mapeamento de reflexão consiste em utilizar uma imagem omnidirecional e mapeá-la sobre um objeto para simular a reflexão do ambiente sobre o mesmo. Assim, o mapeamento de reflexão produz uma aproximação de primeira ordem do método de traçado de raios.

O primeiro passo é obter uma amostra completa da função plenóptica ρ no ponto p, codificada em um mapa omnidirecional $\mathbb{M}$. Tendo o mapa omnidirecional $\mathbb{M}$, para calcular o valor de mapeamento num ponto $p \in O$, consideramos o vetor normal n à superfície O no ponto p, e o vetor de observação $v = \frac{p-o}{\|p-o\|}$ desde a câmera o até o ponto p. Tomamos o vetor de reflexão ideal r do vetor v com respeito à normal n. O vetor r é calculado como

$$r = v - 2 \cdot \langle v, n \rangle \cdot n. \tag{8.2}$$

O valor do mapeamento no ponto p é dado pelo valor armazenado na direção ω no mapa $\mathbb{M}$, sendo $\omega = WTL(r)$ a transformação do vetor r do sistema de coordenadas do mundo para o sistema de coordenadas local do mapa $\mathbb{M}$.

Uma vez obtido o valor $\mathbb{M}(\omega)$ correspondente ao mapeamento associado com o ponto p para a direção r, devemos utilizar este valor para calcular o valor da componente especular

na equação de iluminação utilizada.

O mapeamento de reflexão é simples de ser implementado, computacionalmente eficiente e muito efetivo para diversas situações. Os passos para implementar um mapeamento de reflexão numa cena baseada em *POMC* para aplicar reflexão em um objeto O são:

- Colocar uma câmera omnidirecional no centro p do objeto O e renderizar uma vista omnidirecional do *POMC* no ponto p. A cena deve ser renderizada sem incluir o objeto O, no seu lugar estará a câmera omnidirecional com a qual será feita a renderização para gerar o mapa $\mathbb{M}$.
- Associar o mapa $\mathbb{M}$ com o objeto O. Na renderização de novas vistas do *POMC*, por cada fragmento reflexivo do objeto O, calculamos a normal n do fragmento.
- Calculamos o vetor de reflexão r a partir do vetor do observador v e a normal do fragmento n.
- Calculamos $\omega = WTL(r)$ a direção correspondente a r no mapa de reflexão $\mathbb{M}$.
- O valor $\mathbb{M}(\omega)$ é utilizado para realizar o cálculo de a componente especular do fragmento.

O mapeamento de reflexão com um mapa omnidirecional de reflexão tem um ponto fraco. Não pode ser utilizado para calcular reflexões sobre objetos planares. Como vimos, o que faz que o mapeamento de reflexão funcione é a variação da normal r sobre a superfície do objeto sobre o qual será aplicado o efeito de reflexão. Num objeto planar, a normal r é constante sobre sua superfície, de modo que sempre obteremos a mesma direção ω em $\mathbb{M}$ para qualquer ponto da superfície.

A técnica de Blinn e Newell, [6], utiliza um mapeamento equiretangular da esfera para produzir os mapas de reflexão. Este tipo de mapas pode introduzir algumas distorções visíveis nas áreas próximas aos polos devido à grande deformação do mapeamento. Este tipo de problemas foi resolvido com a aparição do mapeamento de cubo, introduzido por Greene em 1986, [30]. Na figura (8-5) apresentamos dois exemplos de mapeamento de reflexão realizados sobre objetos com superfície curva. Cada caso foi simulado para reflexões especulares e reflexão especular rugosa. A reflexão especular rugosa foi obtida fazendo uma filtragem do mapa de reflexão com o procedimento apresentado na seção 8.5.3.

(a) Esfera com mapeamento de reflexão.

(b) Esfera com mapeamento de reflexão.

(c) Toro com mapeamento de reflexão.

(d) Toro com mapeamento de reflexão.

Figura 8-5: (a) e (c): O mapeamento de reflexão foi realizado com o *POMC* filtrado para simular uma reflexão mais difusa. (b) e (d): O mapeamento de reflexão foi realizado com o *POMC* original para simular uma reflexão especular ideal. No caso da esfera a câmera está posicionada na origem do *POMC* e a esfera em uma posição arbitrária da cena. No caso do toro, a câmera está deslocada, enquanto que o toro está posicionado na origem do *POMC*.

8.5.2 Mapeamento de reflexões planares

Como foi mencionado na seção anterior, a técnica de *environment mapping* não é útil para aplicar reflexos sobre superfícies planas. Sabemos pela lei de reflexão, que em uma reflexão especular ideal, o ângulo de incidência é igual ao ângulo de reflexão com respeito à normal da superfície, figura (8-6). No caso de um observador olhando um reflexo através de um espelho, o mesmo tem a sensação de que o objeto refletido encontra-se detrás do espelho, como se a cena estivesse duplicada. Assim, se retirarmos a superfície refletora e fizermos uma reflexão da cena no plano de reflexão, o observador teria a mesma sensação de estar olhando através do espelho. Esta ideia pode ser explorada para obter uma técnica de mapeamento para reflexões especulares. Em lugar de seguir o raio refletido, podemos prolongar o raio incidente através da superfície refletora e intersectar os mesmos pontos que o raio refletido, somente que sobre um objeto refletido, figura (8-6).

Portanto a técnica para realizar a reflexão sobre uma superfície plana é a seguinte:

Sejam n a normal e p um ponto da superfície planar refletora Ω. Seja M a matriz de

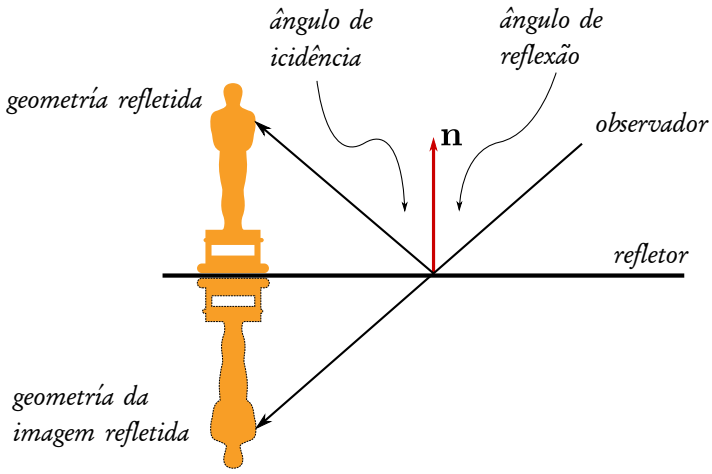


Figura 8-6: Cálculo de reflexões planares.

reflexão da cena sobre o plano $\Omega(p,n)$. Deixamos câmera na posição original. Aplicamos a matriz de reflexão M a todas as luzes da cena. A seguir aplicamos a matriz M a todos os elementos da cena restantes que se encontram na frente do plano de reflexão Ω, i.e., um ponto q é transformado se e somente se $\langle n, q-p\rangle > 0$. Os pontos que não foram transformados são descartados. A nova cena obtida, C_r, deve ser renderizada numa imagem I_r. Na verdade, em geral o objeto refletor Ω não ocupa toda a área da tela. Neste caso é possível guardar uma máscara da posição do objeto refletor na tela, por exemplo usando um *stencil buffer*, e renderizar unicamente os pixels da imagem I_r que correspondem à região do objeto refletor. Isto vai garantir que somente será transferido o reflexo para a imagem final nas áreas ocupadas pelo objeto refletor.

Se o sistema de renderização permite o uso de planos de clipping, é possível resolver o problema sem necessidade de refletir a cena, [31]. Neste caso refletimos a posição e orientação do observador, e colocamos um plano de clipping na posição da superfície refletora para fazer o clipping de todos os objetos que estão entre a nova posição do observador e o plano de clipping (plano de reflexão). Assim renderizando a cena desde a nova posição e com o clipping resultará na imagem de reflexão procurada I_r.

O método de reflexões planares pode ser aplicado para outro tipo de superfícies, desde que sejam quase planares. Por exemplo se a superfície é uma área que representa a superfície da agua numa piscina. A superfície não é totalmente plana, tem ondulações que distorcem o reflexo. Neste caso podemos proceder fazendo o cálculo do mapa de reflexão para uma superfície plana e na hora do mapeamento da reflexão na superfície original, podemos aplicar uma distorção via *warping* de textura. Também é possível realizar mapeamentos que

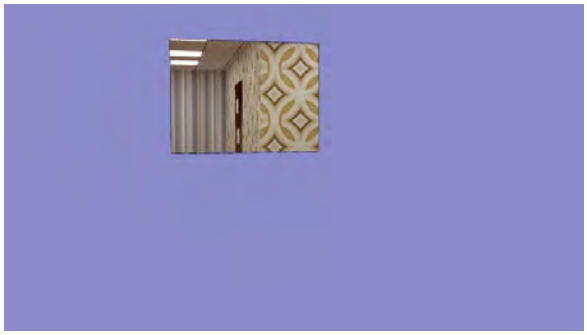
não representem uma reflexão ideal, por exemplo incluir uma atenuação por profundidade durante o cálculo da textura de reflexão fará com que na hora de aplica-la os objetos mais

(a) Vista $S3$. Reflexão no plano do espelho.

(b) Vista $S3$. Reflexão no plano do piso.

(c) Vista $S3$. Iluminação difusa.

(d) Combinação de iluminação difusa e reflexões.



(e) Vista $S13$. Reflexão no plano do espelho.

(f) Vista $S13$. Reflexão no plano do piso.

(g) Vista $S13$. Iluminação difusa.

(h) Combinação de iluminação difusa e reflexões.

Figura 8-7: Exemplos de reflexões planares calculada pelo método de reflexão da cena em torno do plano refletor. A variação das tonalidades na cor é devida ao processo de tone mapping aplicado em cada resultado parcial por separado.

distantes se desvaneçam na reflexão. Também é possível aplicar efeitos de profundidade de campo e outros.

Na figura (8-7) mostramos vistas das cenas $S3$ e $S13$ (todas as cenas são apresentadas na tabela (8.1)) calculadas utilizando o método de reflexões planares. Usando a máscara da área do espelho na tela, renderizamos uma vista da cena refletida, sem incluir o plano refletor. Isto é feito para o espelho na parede (figuras (8-7a) e (8-7e)), e para o piso (figuras (8-7b) e (8-7f)). Depois é renderizada a cena original, com iluminação difusa no piso e sem iluminação no espelho (só queremos reflexo). Na passada final, somamos os resultados parciais para incluir o reflexo do espelho e adicionar a reflexão parcial do piso.

8.5.2.1 Reflexões recursivas

Reflexões recursivas para objetos do *POMC* ou objetos sintéticos adicionados, podem ser logrados usando a técnica de mapeamento de reflexão e a técnica para reflexões planares, [7, 21]. Por exemplo, imagine dois objetos reflexivos O_1 e O_2. Suponha que para O_1 calculamos um mapa de reflexão $\mathbb{M}$ e que o objeto O_2 é plano e calculamos uma textura T da reflexão planar.

Escolhemos uma ordem de renderização, por exemplo $\mathbb{M}$, T. A seguir, tomamos a cena C_1 sem o objeto O_1, e calculamos um mapa $\mathbb{M}_0$. A seguir introduzimos na cena o objeto O_1 e quitamos O_2, e ajustamos a transformação para conseguir a reflexão no plano que contem O_2. Chamamos esta cena de C_2. Renderizamos a cena C_2 numa textura T_0, usando durante a renderização o mapa $\mathbb{M}_0$ para aplicar reflexão no objeto O_1.

Renderizamos novamente a cena C_1, somente que desta vez usamos a textura T_0 para aplicar reflexão no objeto O_2. O resultado é guardado no mapa de reflexão $\mathbb{M}_1$.

No próximo passo renderizamos C_2, usando o mapa $\mathbb{M}_1$ para aplicar reflexão no objeto O_1.

Desta forma renderizamos sucessivamente as cenas C_1 e C_2 alternadamente, salvando os mapas de reflexão $\mathbb{M}_k$ e T_k que serão usados na próxima iteração de renderização. O processo pode ser resumido no seguinte fluxo entrada-saída:

$$T_{k-1} \rightarrow Render_k(C_1) \rightarrow \mathbb{M}_k \tag{8.3}$$

$$\mathbb{M}_k \rightarrow Render_k(C_2) \rightarrow T_k \tag{8.4}$$

Numa aplicação interativa, é difícil realizar este tipo de cálculo recursivo frame a frame e obter uma boa taxa. Nesses casos uma estratégia é calcular um mapa por frame e usar os mapas acumulados dos frames anteriores para obter reflexões recursivas aproximadas, [27].

Os métodos de mapeamento de reflexão para superfícies curvas e o método para superfícies planas, podem ser utilizados com esquemas de *forward rendering* ou *deferred rendering*. Para o esquema de renderização híbrido, a proposta é explorar a busca de interseções dos raios gerados pelo algoritmo de *ray tracing* acoplado no rasterizador.

8.5.3 Filtragem de POMC

Até aqui explicamos quais são as técnicas que podemos utilizar para aplicar reflexões de tipo especular na visualização do de um *POMC*. Entretanto, no mundo real existem superfícies que têm brilho e refletem a cena, mas com certo grau de borramento. Estas superfícies têm uma *BRDF* (Bidirectional Reflectance Distribution Function) que espalha a luz de uma forma mais complexa que uma simples função delta. Para simular este tipo de materiais, é necessário adotar novas estratégias. O algoritmo de *ray tracing* consegue resolver o problema lançando raios de reflexão adicionais que são selecionados probabilisticamente, de forma que a estimativa de Monte Carlo dos mesmos converge para o valor da integral de espalhamento (3.1) que calcula a iluminação total dos raios incidentes sobre o hemisfério visível num ponto da superfície. Este processo é caro e somente é recomendado para renderizar visualizações offline do *POMC*. Uma alternativa é filtrar adequadamente os mapas de iluminação das camadas do *POMC* com uma função de convolução que simula a BRDF desejada. Depois usamos este mapa filtrado usando apenas o raio de reflexão ideal como fizemos até agora. Esta estratégia pode ser usada tanto com métodos de rasterização quanto com o método híbrido.

Como o processo de convolução elimina as altas frequências nos mapas processados, portanto não é necessário trabalhar com mapas de alta resolução. Podemos reduzir drasticamente o tamanho dos mapas: por exemplo os mapas que tem camadas de iluminação difusa cujas faces no mapa de cubo têm resolução de 2048×2048 podem ser reduzidos para versões de 256×256 segundo a necessidade e o tamanho do filtro de convolução, que dependerá do tipo de BRDF a ser simulada.

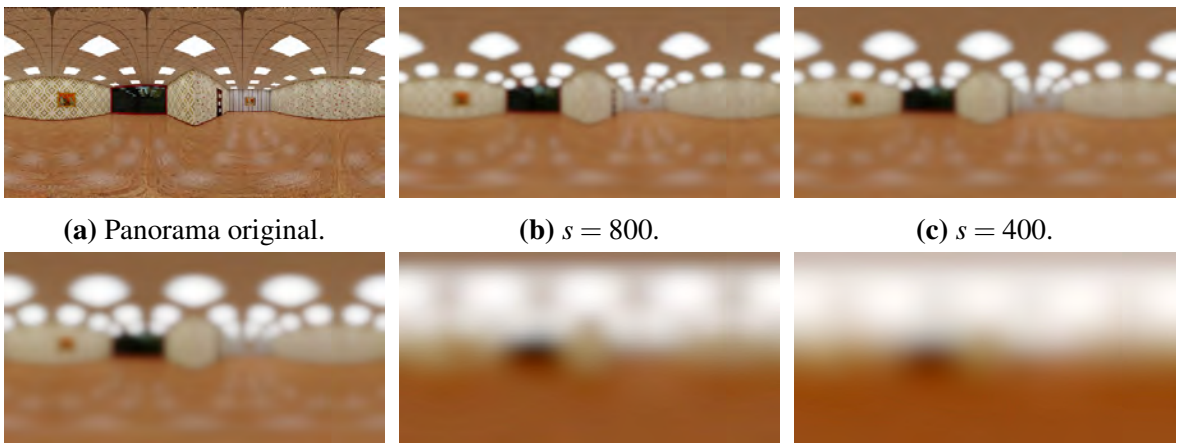
O algoritmo 18 mostra o processo básico para realizar uma filtragem guiada por uma função de convolução.

A função $brdf(\omega, d)$, é a função que determina o fator com que a iluminação entrante pela direção ω

construindo vários mapas para as camadas, podemos interpolar entre eles para consegui efeitos relativos à distância entre o objeto e recalcular um lóbulo mais ou menos aberto para simular o borramento maior causado pela distância da cena.

Algorithm 18 Filtragem de um mapa de iluminação

Input: mapa difuso M da camada;
criar um mapa M_c vazio;
for cada pixel j em M_c **do**
    $\omega \leftarrow$ direção esférica associada ao pixel j;
    $color \leftarrow (0,0,0)$;
    $weight \leftarrow 0$;
    for cada pixel i em M **do**
        $\delta \leftarrow$ direção esférica associada ao pixel i;
        $k \leftarrow$ deformação de área do mapeamento (equiretangular ou mapa de cubo);
        $f \leftarrow BRDF(\omega, \delta)$
        if $f > 0$ **then**
            $weight \leftarrow weight + k \cdot f$;
            $color \leftarrow color + k \cdot f \cdot M(i)$;
        end if
    end for
    $M_c(j) \leftarrow color/weight$;
end for



(a) Panorama original. **(b)** $s = 800$. **(c)** $s = 400$.

(d) $s = 200$. **(e)** $s = 20$. **(f)** $S = 10$.

Figura 8-8: Exemplos de convoluções realizadas para simular a *brdf* de Phong. A função utilizada é $f(\omega, \delta) = \langle \omega, \delta \rangle^s$. Usamos vários valores de s para ilustrar os efeitos na camada de fundo do *POMC*.

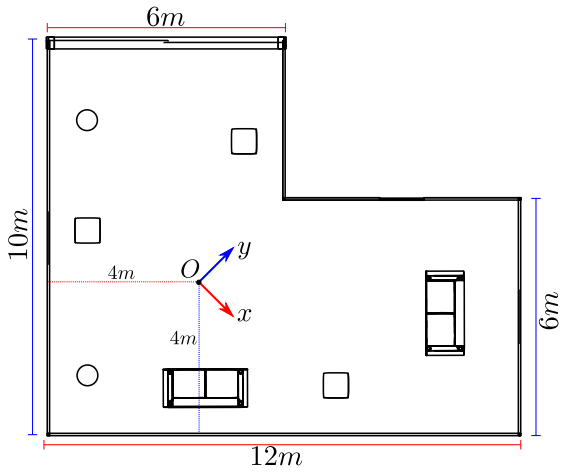


Figura 8-9: Vista esquemática da cena do *POMC*. O ponto *O* é a origem do *POMC*.

Vista	**Posição de câmera** $C = (p_x, p_y, p_z)$	**Direção de visada** $d = (d_x, d_y, d_z)$
S1	$(0.0, 0.0, 1.15)$	$(0.93, 0.34, -0.10)$
S2	$(0.0, 0.0, 1.15)$	$(0.34, -0.93, -0.10)$
S3	$(0.0, 0.0, 1.15)$	$(-0.95, -0.29, -0.10)$
S4	$(0.0, 0.0, 1.15)$	$(-0.05, 0.99, -0.10)$
S5	$(0.327, -0.503, 1.15)$	$(0.83, 0.54, -0.10)$
S6	$(0.686, -0.515, 1.119)$	$(0.84, 0.53, -0.10)$
S7	$(-0.053, -0.645, 1.119)$	$(0.02, 0.99, -0.10)$
S8	$(-0.041, -0.744, 1.119)$	$(-0.98, -0.14, -0.10)$
S9	$(0.158, -0.720, 1.119)$	$(0.30, -0.93, -0.19)$
S10	$(-0.023, -0.850, 1.138)$	$(0.95, -0.24, -0.21)$
S11	$(-0.063, -0.859, 1.333)$	$(-0.96, -0.20, -0.21)$
S12	$(-0.387, 0.9125, 1.089)$	$(-0.86, -0.50, -0.14)$
S13	$(0.023, 0.622, 1.052)$	$(-0.86, -0.51, -0.03)$

Tabela 8.1: Vistas do *POMC* utilizadas nos testes. As vistas *S*1, *S*2, *S*3 e *S*4 têm a câmera posicionada na origem do *POMC*.

8.6 Resultados

Os testes realizados foram feitos com OpenGL 3.3 em uma placa gráfica *HD4400* da *Intel*. Recentemente foram anunciados novos sistemas, como o *Vulcan*, que prometem um desempenho muito superior aos atuais. Não tivemos cuidados em otimizar a implementações e as camadas do *POMC* foram passadas ao shader em forma de texturas RGB em ponto flutuante de 16bits, sendo que a textura de cada camada tem uma resolução de 6144×4096 pixels (camadas sem compressão). Para aplicações em dispositivos móveis e óculos de realidade virtual é possível utilizar texturas com resolução inferior e manter a qualidade visual devido a que estes equipamentos costumam ter resolução inferior a Full HD (e.g., *Oculus Rift DK2* tem 960×1080 pixels por cada olho) e utilizam um FOV de aproximadamente 110°, enquanto que nos testes realizados usamos câmeras com FOV de 60° na maioria dos casos.

A seguir apresentaremos visualizações produzidas com o método de renderização híbrido. Para realizar os testes, foram selecionadas 13 posições e orientações de câmera nas proximidades da origem do *POMC* escolhido para o teste. As informações das 13 vistas escolhidas encontram-se na tabela (8.1). As vistas *S*1 até *S*4 têm a posição de câmera na origem do panorama de teste. Nas restantes vistas, as posições de câmera variam entre 0.6m a 1m de distância da origem do panorama. Na figura (8-9) mostramos uma vista esquemática da cena representada no panorama em camadas.

Por exemplo se pré-calculamos todos os efeitos de iluminação desde o ponto de origem do panorama e logo utilizamos esta informação para renderizar novas vistas, podemos obter efeitos indesejáveis que advertem ao observador que há alguma coisa errada na cena. No caso

de um movimento continuo de deslocamento da câmera, provocará uma distorção contínua nas reflexões que se traduz em um efeito de arraste muito perceptível e incómodo para o observador. Por outra parte se as informações de reflexão são calculadas em função do novo ponto de vista e remapeadas adequadamente na cena, é possível manter uma consistência e promover uma sensação de imersão natural para o observador. Ilustramos estas situações na figura (8-10) onde mostramos 2 vistas ($S6$ e $S12$) mapeadas com reflexão pré-calculada na origem do *POMC* e reflexão calculada durante o processo de visualização levando em consideração a posição da câmera.

Na figura (8-11) mostramos os efeitos de iluminação *view-dependent* na vista $S10$. Podem ser apreciados os efeitos de utilizar fatores de atenuação baseados em distância dos objetos. A figura (8-11b) mostra os efeitos logrados utilizando um *POMC* filtrado com o algoritmo da seção 8.5.3 para um índice de especularidade $s = 200$.

A figura (8-12) mostra diferentes efeitos de iluminação que podem ser logrados variando os parâmetros de especularidade associados com o material de um dos objetos (*Puff*). O caso ilustrado é simples e o parâmetro afetado é o índice de reflexividade associado com o objeto. Utilizando mapas de propriedades mais avançadas é possível lograr efeitos mais avançados, como por exemplo ter diferentes níveis de reflexividade num mesmo objeto.

As figuras (8-13) e (8-14) mostram varias vistas do *POMC* com iluminação difusa pré-calculada e as mesmas vistas com iluminação *view-dependent*. No caso da figura (8-13) as vistas são geradas a partir de uma posição de câmera que coincide com a origem do *POMC*. Na figura (8-14) as vistas têm a câmera deslocada da origem do *POMC*. Na figura (8-15) são mostrados os resultados para iluminação *view-dependent* das vista $S5$ até $S12$. Nas figuras (8-15d), (8-15g) e (8-15h) um quadro foi substituído por um espelho para mostrar o efeito de reflexões especulares.

Na figura (8-16) realizamos uma comparação de duas vistas $S2$ e $S9$ que observam a mesma porção do *POMC* desde posições de câmera diferentes. Embora sutil, é possível observar algumas variações nas posições dos reflexos das luzes sobre a estátua do *Buda*. Na figura também são mostradas algumas variações dos parâmetros de reflexão para a estátua do *Buda*.

8.7 Conclusões

Este capítulo tem um carácter complementar dos problemas tratados no capítulo 7. Foi possível perceber através dos experimentos realizados a importância dos efeitos de iluminação *view-dependent* para produzir visualizações realistas. Também foi possível observar que certos efeitos têm maior importância ou destaque para o observador, de modo que é necessário focar na correta solução dos mesmos. Este tipo de situações pode ser notado por exemplo nos reflexos sobre o piso do ambiente retratado. Um cálculo incorreto desses reflexos é facilmente percebido pelo observador, enquanto que reflexos sobre superfícies curvas são mais fáceis de tratar pois não permitem, especialmente em situações de movimento de câmera, que o observador tenha tempo de estudá-los. Isto mostra porque técnicas para reflexões planares e mapeamentos de reflexão com mapas de cubo forma e ainda são tão utilizados na indústria. Por outro lado, a técnica híbrida mostrou uma vantagem sobre outros esquemas em termos de usabilidade. Tendo o *POMC* e definidas as camadas e informações geométricas e parâmetros de reflexão o método constrói a estrutura de aceleração de maneira automática e o cálculo de reflexões é implementado através do traçado de raios. Por outro lado as técnicas de cálculo de reflexões planares requerem varias passadas por diversos shaders, para calcular resultados parciais e integra-los na imagem final. A quantidade de passos necessários depende da quantidade de planos diferentes para os quais é necessário calcular reflexões. Para um sistema genérico que possa ser utilizado com diversos panoramas, a opção que se vislumbra mais viável é a do método de renderização híbrido.

(a) Vista *S*6. Reflexão pré-calculada.

(b) Vista *S*6. Reflexão calculada com ray tracing.

(c) Vista *S*12. Reflexão pré-calculada.

(d) Vista *S*12. Reflexão calculada no ray tracing.

Figura 8-10: (a) e (c): Renderização com cálculo de reflexão pré-calculado para a origem do *POMC*. Como as posições de câmera estão deslocadas da origem do *POMC*, os reflexos são incorretos. (b) e (d): Renderização com cálculo de reflexão calculado com o método híbrido (*deferred rendering* + *ray tracing*) durante a renderização.

(a) Vista *S*10. Iluminação difusa.

(b) Vista *S*10. Reflexão com atenuação.

(c) Vista *S*1. *POMC* filtrado + atenuação.

(d) Vista *S*10. Reflexão sem atenuação.

Figura 8-11: Comparação da iluminação difusa com efeitos *view-dependent* para a vista *S*10. Observe o reflexo das lâmpadas do teto no piso nas figuras (a), (b) e (c).

(a) Vista $S11$. Iluminação difusa pré-calculada.

(b) Vista $S12$. Iluminação difusa pré-calculada.

(c) Vista $S11$. *Puff* com pouca especularidade.

(d) Vista $S12$. *Puff* com pouca especularidade.

(e) Vista $S11$. Aumento da especularidade.

(f) Vista $S12$. Aumento da especularidade.

Figura 8-12: Vistas $S11$ e $S12$. As figuras (a) e (b) mostram a iluminação difusa da cena. Nas figuras (c) e (d) são incluídos os reflexos do piso e no *Puff*. O *Puff* tem um índice de especularidade baixo. Nas figuras (e) e (f) foi incrementado o índice de especularidade do *Puff*. Também é possível observar a variação nos reflexos na cena ao mudar a posição de câmera da vista $S11$ para a vista $S12$.

(a) Vista $S1$. Iluminação *view-dependent*.

(b) Vista $S1$. Iluminação difusa pré-calculada.

(c) Vista $S2$. Iluminação *view-dependent*.

(d) Vista $S2$. Iluminação difusa pré-calculada.

(e) Vista $S3$. Iluminação *view-dependent*.

(f) Vista $S3$. Iluminação difusa pré-calculada.

(g) Vista $S4$. Iluminação *view-dependent*.

(h) Vista $S4$. Iluminação difusa pré-calculada.

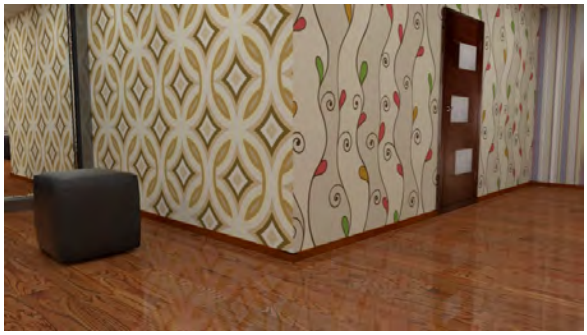
Figura 8-13: Vistas com câmera posicionada na origem do *POMC*. Na coluna da esquerda incluímos as reflexões calculadas pelo método híbrido. Na coluna da direita somente mostramos a informação difusa pré-calculada no *POMC*. Na vista $S3$ foi adicionado um objeto especular (espelho na parede). Os reflexos são calculados utilizando um fator de atenuação baseado no quadrado da distância (observe a atenuação dos reflexos no piso).

(a) Vista *S*5. Iluminação *view-dependent*.

(b) Vista *S*5. Iluminação difusa pré-calculada.

(c) Vista *S*6. Iluminação *view-dependent*.

(d) Vista *S*6. Iluminação difusa pré-calculada.



(e) Vista *S*7. Iluminação *view-dependent*.

(f) Vista *S*7. Iluminação difusa pré-calculada.

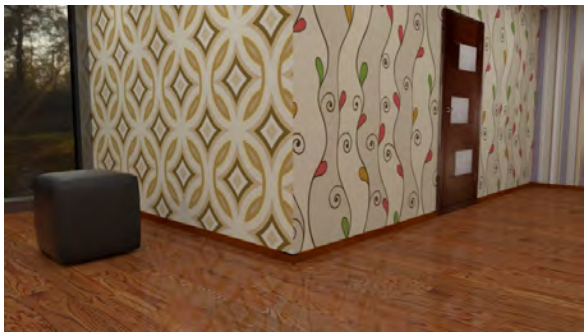
(g) Vista *S*8. Iluminação difusa pré-calculada.

(h) Vista *S*8. Iluminação *view-dependent*.

Figura 8-14: Vistas com câmera posicionada fora da origem do *POMC*. As reflexões são calculadas pelo método híbrido e não são atenuadas pela distância. Na coluna da direita colocamos a vista da cena somente com a iluminação difusa pré-calculada no *POMC*.

(a) Vista $S5$. Iluminação *view-dependent*.

(b) Vista $S6$. Iluminação *view-dependent*.



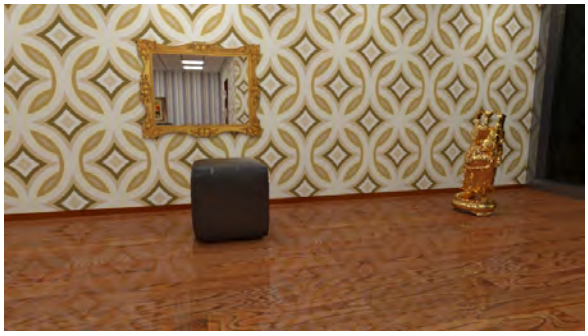
(c) Vista $S7$. Iluminação *view-dependent*.

(d) Vista $S8$. Iluminação *view-dependent*.

(e) Vista $S9$. Iluminação *view-dependent*.

(f) Vista $S10$. Iluminação *view-dependent*.

(g) Vista $S11$. Iluminação *view-dependent*.



(h) Vista $S12$. Iluminação *view-dependent*.

Figura 8-15: Vistas com câmera posicionada fora da origem do *POMC*. As reflexões são calculadas pelo método híbrido utilizando um fator de atenuação baseado no quadrado da distância do raio refletido. Nas vista $S11$ e $S12$ foi adicionado um objeto especular na parede.

(a) Vista *S*2. Iluminação difusa pré-calculada.

(b) Vista *S*9. Reflexão de Phong com mapa convolucionado.

(c) Vista *S*2. *Buda* com índice de reflexão alto.

(d) Vista *S*9. *Buda* com índice de reflexão alto.

(e) Vista *S*2. *Buda* com índice de reflexão médio.

(f) Vista *S*9. *Buda* com índice de reflexão médio.

(g) Detalhe da figura (c).

(h) Detalhe da figura (d).

(i) Detalhe da figura (e).

(j) Detalhe da figura (f).

Figura 8-16: Exemplos de cálculo de reflexões utilizando diferentes índices de reflexão. Nas figuras os índices das estatuas do *Buda* foram renderizadas com diferentes parâmetros de reflexividade. Na figura (b) foi utilizado um mapa convolucionado com a técnica descrita na seção 8.5.3 para recriar a aparência de um metal escovado. O mesmo mapa foi utilizado para o mapeamento de reflexão do piso, que aparece com um brilho sutil.

Capítulo 9

Considerações finais

9.1 Conclusões

Foi apresentada uma breve análise de alguns métodos de amostragem para mapas de iluminação e apresentamos um esquema híbrido para cálculos de iluminação direta. Este esquema mostrou-se competitivo, especialmente quando utilizado com mapas de iluminação procedentes de ambientes reais.

Estudamos o problema de renderização de cenas de realidade mista, usando como dados iniciais uma fotografia da cena real e um mapa de iluminação capturado da mesma cena. O método de renderização que foi proposto consegue lidar com situações variadas e complexas de forma eficiente. Como foi mostrado, o método consegue lidar com inter-reflexões entre objetos sintéticos e reais de maneira consistente. Em outros artigos da área, como os de Debevec [17] e Karsch [35] não são tratados casos com os níveis de complexidade que apresentamos aqui (ver seção 4.4). Outro dos pontos altos do método é o fato de realizar a renderização em um único passo, sem necessidade de varias etapas de renderização para estimar *brdf*'s ou pós-processamento e composição de resultados parciais. Estas características fazem que o método seja especialmente útil em um pipeline de produção de animações.

Um aspecto que em certas circunstancias pode ser negativo quando usamos mapas de iluminação é o fato da iluminação do mapa ser tratada como iluminação direcional, devido à falta de informação posicional das amostras. A solução destes e outros problemas foram explorados no capítulo 5, com a apresentação do conceito de mapas de iluminação *RGB-D*.

Também apresentamos uma solução para o problema de introduzir elementos sintéticos dentro de um panorama omnidirecional *HDR*. Neste caso o panorama teve duas funções, foi

usado como mapa de iluminação e como fundo da cena. Para conseguir resultados com alto grau de realismo, foi necessário introduzir o conceito de mapa de iluminação *RGB-D*, através do qual dotamos ao panorama (mapa de iluminação) de uma estrutura geométrica subjacente, armazenada em um canal de profundidade *D*. Também mostramos como esta estrutura adicional no mapa de iluminação permite realizar cálculos de iluminação, sombras e reflexões e refrações que resultam consistentes com o ambiente real representado no panorama.

Foi apresentada uma nova formulação para imagens omnidirecionais, os *panoramas omnidirecionais com múltiplas camadas* (*POMC*), que permitem reconstruir amostras da função plenóptica em uma vizinhança de um ponto. Estes conceitos abrem novas possibilidades de entender os usos dos panoramas em computação gráfica.

Exploramos o uso de algoritmos de determinação de superfícies visíveis para gerar novas vistas da função plenóptica desde posições de câmera diferentes ao centro do *POMC*. Mostramos como os cálculos e determinação correta da visibilidade permite lograr reconstruções com a estrutura geométrica correta, logrando também efeitos de paralaxe quando movimentamos a câmera. Este tipo de aplicação é importante no contexto de visualização interativa, pois permite incrementar o realismo devido aos efeitos de paralaxe geométrica que aparecem quando resolvemos corretamente o problema de cálculo de superfícies visíveis.

Uma das aplicações possíveis para os *POMC* que estudamos é o problema de renderização de novas vistas com posições de câmera em uma vizinhança da origem do *POMC*. Como vimos, em este tipo de aplicações são importantes no contexto de visualização interativa, e as soluções que exploram a informação geométrica codificada no *POMC* permitem incrementar o realismo devido aos efeitos de paralaxe geométrica que são logrados quando resolvemos corretamente o problema de cálculo de superfícies visíveis.

Mostramos alguns métodos que podem ser utilizados para incluir efeitos de iluminação *view-dependent*, e mostramos como os mesmos ajudam a incrementar o grau de fotorrealismo de uma cena. A variação de iluminação associada ao movimento de câmera permite ao observador ter uma interpretação 3D da cena mais precisa, mesmo que a visualização não seja em estéreo.

Os conceitos introduzidos também abrem novas oportunidades para pesquisas futuras que discutiremos rapidamente a seguir.

9.2 Trabalhos futuros

Boa parte dos problemas estudados em esta tese ainda podem ser explorados e abrem novas perguntas que podem direcionar pesquisas futuras. Faremos uma breve menção de algumas destas possíveis direções a seguir:

- Na renderização de cenas de realidade mista, ha varias formas de complementar os resultados obtidos. Por um lado devemos pesquisar outros algoritmos, como photon mapping, bidirectional path tracing, igi, etc. Também é importante estudar o problema de captura de mapas de iluminação *RGB-D*. Este problema é relevante, pois sua solução pode aportar conhecimentos necessários para tratar os problemas relacionados à construção de panoramas em camadas.

- Pretendemos estudar melhor os problemas relacionados à construção de panoramas em camadas, utilizando equipamentos como *Tango Project* e *Structure Sensor*. Também queremos explorar trabalhos como KinectFusion [46] e Livedense [45] que permitam o uso de dispositivos de baixo custo para capturar informação geométrica e depois integrá-la aos panoramas. Para panoramas destinados a usos específicos, como visualização estereográfica, podemos pensar em construir o panorama em camadas aumentando um panorama *RGB-D* utilizando técnicas de síntese de textura. Esta é uma assinatura pendente que pretendemos explorar, dado que existem inúmeros panoramas na internet que poderiam ser expandidos.

- Outro problema a ser estudado é a visualização em estéreo a partir de um panorama em camadas e a manipulação de disparidade durante o processo de visualização.

- Estudar métodos para a obtenção de malhas simplificadas para representar a geometria das camadas. Pretendemos estudar métodos de simplificação que utilizem a informação de profundidade e textura do panorama durante o processo de simplificação, a fim de manter a qualidade geométrica. Um aspecto de importância a ser estudado são as formas de codificar malhas e texturas para minimizar o espaço de armazenamento dos mesmos, para viabilizar aplicações que utilizem vídeos de panoramas em camadas.

- Estudar problemas relacionados a panoramas com camadas com variação temporal.

- Continuar as pesquisas de algoritmos para realizar o cálculo de superfícies visíveis e efeitos view-dependent de maneira eficiente.

Apêndice A

Mapeamentos para imagens omnidirecionais

Para trabalhar com iluminação de cenas artificias através de iluminação extraída de um ambiente real, é necessário armazenar a iluminação do ambiente real. Uma forma de fazer isto é obtendo uma imagem de sondagem de luz (*Light Probe*). Uma imagem de sondagem de luz (*Light Probe*) é uma imagem omnidirecional de alta faixa dinâmica que registra as condições de incidência da iluminação num ponto determinado do espaço.

Uma vez capturada, a imagem de sondagem de luz (*Light Probe*) precisa ser armazenada num arquivo de imagem utilizando um mapeamento de imagem omnidirecional, obtendo assim uma representação digital de toda a luz da cena real numa imagem formatada, denominada *mapa de iluminação*. Nesta seção vamos descrever dois tipos de parametrizações communmente utilizads para imagens omnidirecionais, o mapeamento equiretangular e o mapeamento de cubo. A seguir apresentamos as fórmulas para determinar adequadamente as coordenadas de imagem (u,v), correspondentes a uma direção unitária do mundo $\omega = (x,y,z)$, e vice-versa. As coordenadas do mundo real estão no sistema de coordenadas canônico $e_1 = (1,0,0)$, $e_2 = (0,1,0)$ e $e_3 = (0,0,1)$.

A.1 Imagem digital

Definiremos uma imagem contínua como uma aplicação $f : U \subset \mathbb{R}^2 \to C$, onde o domínio U é o suporte da imagem, e C é um espacio vetorial. O conjunto imagem de f, $f(U) \subset C$, é chamado de *valores da imagem* ou *gamute de cores* da imagem.

A representação mais comum de imagem em computação gráfica consiste em tomar um subconjunto discreto $U' \subset U$ do domínio da imagem, um espaço de cor associado a um dispositivo gráfico, e a imagem representada pela amostragem da função imagem f no subconjunto U'. Neste caso a imagem $f(x,y)$, será espacialmente continua ou discreta se as coordenadas (x,y) de cada ponto variam no conjunto U ou U' respectivamente. Cada ponto (x_i,y_i) do subconjunto discreto U' é chamado de *elemento de imagem* ou *pixel*.

O caso mais utilizado de discretização espacial de uma imagem consiste em tomar o domínio como sendo um retângulo

$$U = [a,b] \times [c,d] = \{(x,y) \in \mathbb{R}^2 : a \leq x \leq b \quad \text{e} \quad c \leq y \leq d\}, \tag{A.1}$$

e discretizar o retângulo usando os pontos de um reticulado bidimensional $\Delta = (\Delta_x, \Delta_y)$,

$$\Delta = \{(x_i,y_j) \in U : x_i = i \cdot \Delta_x, \;\; y_j = j \cdot \Delta_y, \;\; i,j \in \mathbb{Z}, \;\; \Delta_x, \Delta_y \in \mathbb{R}\}. \tag{A.2}$$

Cada pixel (x_i,y_j) da imagem pode portanto ser representado por coordenadas inteiras (i,j). Assim a imagem pode ser representada de forma conveniente no formato matricial, na qual a imagem está associada a uma matriz A de ordem $m \times n$, $A = (a_{ij}) = (f(x_i,y_j))$.

A.2 Formato Equiretangular

Para o mapeamento equiretangular utilizamos um dominio de imagem $u \in [0,1]$, $v \in [0,1]$. A parametrização para o mapeamento equiretangular é dado pelas coordenadas esféricas $(\phi,\theta) \in [0,2\pi] \times [0,\pi]$ da esfera unitaria $\mathbb{S}^2$. A relação entre uma direção $\omega = (x,y,z)$ e as coordenadas esféricas (ϕ,θ) é dada pela equação

$$(\phi,\theta) = \left(\arctan 2\left(\frac{y}{x}\right), \arccos(z)\right). \tag{A.3}$$

A relação das coordenadas de imagem com as coordenas esféricas é dada por

$$(u,v) = \left(\frac{\phi}{2\pi}, \frac{\theta}{\pi}\right). \tag{A.4}$$

As coordenas esféricas e a direção $\omega = (x,y,z)$ podem ser obtidas a partir das coordenadas de imagem (u,v) através das equações

$$
\begin{aligned}
(\phi,\theta) &= (2\pi u, \pi v),\\
x &= \sin\theta\cdot\cos\phi,\\
y &= \sin\theta\cdot\sin\phi,\\
z &= \cos\theta.
\end{aligned}
\tag{A.5}
$$

No mapeamento equiretangular uma direção azimute sobre o plano xy da esfera de direções, é medido a partir da direção $e_1 = (1,0,0)$ e mapeado sobre a coordenada horizontal; a elevação z é mapeada na coordenada vertical da imagem. Este mapeamento achata a esfera numa área retangular. A borda horizontal superior da imagem corresponde à direção $e_3 = (0,0,1)$ da esfera de direções, ou polo Norte; a borda horizontal inferior da imagem corresponde com a direção $-e_3 = (0,0,-1)$ da esfera, ou polo Sul. O formato, naturalmente tem uma relação de aspecto 2:1, e introduz a menor distorção para regiões próximas à linha do horizonte (plano equatorial da esfera de direções). As áreas de igual elevação absoluta $|z|$ na esfera de direções, sofrem mesma deformação, que aumenta de forma progressiva a medida que a elevação se aproxima dos polos.

O mapeamento equiretangular é conveniente porque tem suporte de imagem retangular e não tem costuras (o único problema são os pontos singulares, polos Norte e Sul), e porque as fórmulas do mapeamento são simples e intuitivas. Embora geralmente as linhas retas tornam-se curvas neste formato, os paralelos e meridianos da esfera de direções correspondem a linhas retas horizontais e verticais respectivamente no mapa equiretangular.

Para usar este mapeamento em imagens digitais, devemos lembrar que no computador, o sistema de coordenadas natural para uma imagem está invertido, de modo que a origem $(0,0)$ está localizada no canto superior esquerdo da imagem em vez de estar no canto inferior esquerdo. Tendo isto em consideração, podemos relacionar as equação (A.3) com a imagem digital de dimensões $W \times H$ neste formato da seguinte forma

$$
(u,v) = \left(\frac{i}{W}, \frac{j}{H}\right), \tag{A.6}
$$

logo da equação (A.5) obtemos

$$
(\phi,\theta) = \left(2\pi\frac{i}{W}, \pi\frac{j}{H}\right). \tag{A.7}
$$

A.2.1 Deformação de área e representação do Mapa de Iluminação

Em aplicações como cálculo de iluminação e amostragem de mapas de iluminação, a irradiância está associada com a área emissora, portanto é necessario levar em consideração deformações de área induzidas pelas diferentes parametrizações da esfera unitaria $\mathbb{S}^2$.

No caso do mapeamento equiretangular, a relação entre a diferencial de área de um conjunto de direções $d\omega$ e a diferencial de área do par (ϕ, θ) , figura (A-1a), é dada por:

$$d\omega = \sin(\theta) d\theta d\phi.$$

Consideremos a função de luminância $L : \mathbb{R}^3 \to \mathbb{R}$ definida por $L(r,g,b) := 0.2125 \cdot r + 0.7154 \cdot g + 0.0721 \cdot b$. Então, para um mapa de iluminação $\mathbb{M}$, definimos a função $L_{\mathbb{M}}(\omega) := L(M(\omega))$ que devolve a luminância de $\mathbb{M}$ na direção ω. A luminância total do mapa $\mathbb{M}$ é dada pela integral

$$E = \int_{\mathbb{S}^2} L_{\mathbb{M}}(\omega) d\omega = \int_0^{2\pi} \int_0^{\pi} L_{\mathbb{M}}(\phi, \theta) \sin(\theta) d\theta d\phi, \tag{A.8}$$

A deformação de área deve ser levada em conta no cálculo de iluminação e na construção de *PDFs* (*Funções de Densidade de Probabilidade*) necessárias para realizar estimativas de Monte Carlo da iluminação.

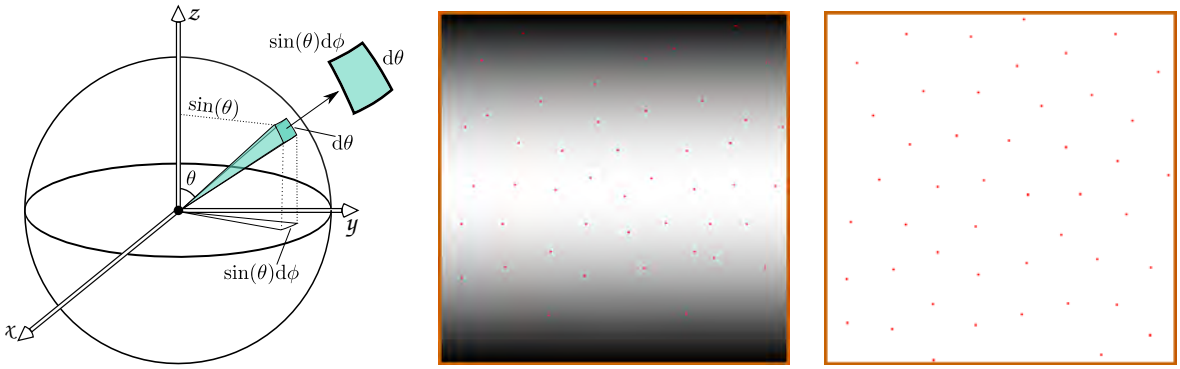
No caso de um mapa de iluminação de tamanho $W \times H$ pixels, se $M_o(\bar{x}, \bar{y})$ é a função de radiância do Mapa Esférico associado, a função de compensação pela deformação de área é definida por:

$$F(M_o(\bar{x}, \bar{y})) := M_o(\bar{x}, \bar{y}) \cdot \sin\left(\frac{\bar{y}}{H}\pi\right) = M_o(\bar{x}, \bar{y}) \cdot \sin(\theta). \tag{A.9}$$

A equação (A.9) também é aplicada à luminância para obter as *PDFs* utilizadas na amostragem:

$$F(L_{M_o}(\bar{x}, \bar{y})) := L_{M_o}(\bar{x}, \bar{y}) \cdot \sin\left(\frac{\bar{y}}{H}\pi\right). \tag{A.10}$$

Na figura (A-1b) vemos o efeito de amostrar um mapa isoluminante e sua transformação. Como resultado, as posições das amostras mudam quando considerarmos a deformação induzida pelo mapeamento esférico.



(a) **(b)** Com correção de área. **(c)** Sem correção de área.

Figura A-1: (a) O diferencial de área dA subtendido por um diferencial de angulo sólido é o produto do diferencial das arestas $\sin(\theta)d\phi$ e $d\theta$. (b) e (c): Amostragem Hierárquica por Importância com Mosaicos de Penrose [47] de um mapa isoluminante com e sem correção de área.

A.3 Mapeamento de Cubo

O formato de mapa de cubo é caracterizado por projetar a esfera $\mathbb{S}^2$ sobre um cubo centrado na origem. Como as faces do cubo são planas, qualquer arranjo das faces do cubo sobre um suporte planar é um mapeamento planar da esfera $\mathbb{S}^2$. Existem varias formas de combinar a posição das 6 faces do cubo sobre um dominio retangular. As clássicas sao as disposições em forma de cruz vertical e cruz horizontal, figura (A-2b), que representam desdobramentos do cubo. Para esta tese no entanto, escolhemos uma organização diferente, colocando as faces numa disposição retangular 3×2 para facilitar o armazenamento.

Usamos um dominio retangular $u \in [0,1]$, $v \in [0,1]$ para realizar a parametrização. As formulas do mapeamento do cubo são 6 no total, uma por cada face do cubo. Devemos determinar a que face do cubo corresponde uma direção para aplicar entao a equação de transformação correspondente. Consideraremos o sistema de coordenadas canônico, e um cubo alinhado com estes eixos, como apresentado na figura (A-3).

Dada uma direção $\omega = (x,y,z) \in \mathbb{S}^2$, sua projeção no cubo é o ponto

$$(\bar{x},\bar{y},\bar{z}) = \frac{(x,y,z)}{\max\{|x|,|y|,|z|\}}. \tag{A.11}$$

A seguir analizaremos as 6 possiveis soluções para a equação (A.11).

$$\max\{|x|,|y|,|z|\} = x \quad \Longrightarrow \quad (\bar{x},\bar{y},\bar{z}) = \left(1, \frac{y}{x}, \frac{z}{x}\right), \tag{A.12}$$

$$\max\{|x|,|y|,|z|\} = -x \quad \Longrightarrow \quad (\bar{x},\bar{y},\bar{z}) = \left(-1, \frac{y}{-x}, \frac{z}{-x}\right), \tag{A.13}$$

$$\max\{|x|,|y|,|z|\} = y \quad \Longrightarrow \quad (\bar{x},\bar{y},\bar{z}) = \left(\frac{x}{y}, 1, \frac{z}{y}\right), \tag{A.14}$$

$$\max\{|x|,|y|,|z|\} = -y \quad \Longrightarrow \quad (\bar{x},\bar{y},\bar{z}) = \left(\frac{x}{-y}, -1, \frac{z}{-y}\right), \tag{A.15}$$

$$\max\{|x|,|y|,|z|\} = z \quad \Longrightarrow \quad (\bar{x},\bar{y},\bar{z}) = \left(\frac{x}{z}, \frac{y}{z}, 1\right), \tag{A.16}$$

$$\max\{|x|,|y|,|z|\} = -z \quad \Longrightarrow \quad (\bar{x},\bar{y},\bar{z}) = \left(\frac{x}{-z}, \frac{y}{-z}, -1\right), \tag{A.17}$$

Destas equações podemos extrair as coordenadas (u,v) na imagem

$$(u,v) = \left(\frac{s+s_0}{3}, \frac{t+t_0}{2}\right), \tag{A.18}$$

onde s_0, t_0, s e t dependem da face do cubo sobre a qual está sendo projetada a direção ω.

$$\max\{|x|,|y|,|z|\} = x \quad \Longrightarrow \quad s_0 = 0, \quad t_0 = 0, \quad s = \frac{\frac{-y}{x}+1}{2}, \quad t = \frac{\frac{-z}{x}+1}{2}, \tag{A.19}$$

$$\max\{|x|,|y|,|z|\} = -x \quad \Longrightarrow \quad s_0 = 0, \quad t_0 = 1, \quad s = \frac{\frac{y}{-x}+1}{2}, \quad t = \frac{\frac{-z}{-x}+1}{2}, \tag{A.20}$$

$$\max\{|x|,|y|,|z|\} = y \quad \Longrightarrow \quad s_0 = 1, \quad t_0 = 0, \quad s = \frac{\frac{x}{y}+1}{2}, \quad t = \frac{\frac{-z}{y}+1}{2}, \tag{A.21}$$

$$\max\{|x|,|y|,|z|\} = -y \quad \Longrightarrow \quad s_0 = 1, \quad t_0 = 1, \quad s = \frac{\frac{-x}{-y}+1}{2}, \quad t = \frac{\frac{-z}{-y}+1}{2}, \tag{A.22}$$

$$\max\{|x|,|y|,|z|\} = z \quad \Longrightarrow \quad s_0 = 2, \quad t_0 = 0, \quad s = \frac{\frac{-x}{z}+1}{2}, \quad t = \frac{\frac{-y}{z}+1}{2}, \tag{A.23}$$

$$\max\{|x|,|y|,|z|\} = -z \quad \Longrightarrow \quad s_0 = 2, \quad t_0 = 1, \quad s = \frac{\frac{x}{-z}+1}{2}, \quad t = \frac{\frac{-y}{-z}+1}{2}, \tag{A.24}$$

Para realizar a conversão no sentido inverso, desde coordenadas de imagem $(u,v) \in [0,1] \times [0,1]$ para direções na esfera $\mathbb{S}^2$, definimos

$$s_o = \lfloor 3u \rfloor, \tag{A.25}$$

$$t_o = \lfloor 2v \rfloor, \tag{A.26}$$

$$s = 2(3u - s_0) - 1, \tag{A.27}$$

$$t = 2(2v - t_0) - 1, \tag{A.28}$$

Então a conversão é dada por

$$3t_0 + s_0 = 0 \qquad \Longrightarrow \qquad \omega = (1, -s, -t), \tag{A.29}$$

$$3t_0 + s_0 = 1 \qquad \Longrightarrow \qquad \omega = (s, 1, -t), \tag{A.30}$$

$$3t_0 + s_0 = 2 \qquad \Longrightarrow \qquad \omega = (-s, -t, 1), \tag{A.31}$$

$$3t_0 + s_0 = 3 \qquad \Longrightarrow \qquad \omega = (-1, s, -t), \tag{A.32}$$

$$3t_0 + s_0 = 4 \qquad \Longrightarrow \qquad \omega = (-s, -1, -t), \tag{A.33}$$

$$3t_0 + s_0 = 5 \qquad \Longrightarrow \qquad \omega = (s, -t, -1), \tag{A.34}$$

Para uma imagem digital de um mapa de cubo de tamanho $3N \times 2N$ pixels, a relação entre o pixel (i, j) e as coordenadas (u, v) do dominio de parametrização é a seguinte

$$(u, v) = \left(\frac{i}{3N}, \frac{j}{2N} \right). \tag{A.35}$$

O mapeamento de cubo foi proposto em 1986 por Ned Greene, [30], e é preferido entre os métodos de mapeamento de ambiente, devido a sua relativa simplicidade. Os resultados de utilizar mapeamento de cubo em reflexoões porduzem resultados semelhantes ao de um traçado de raios, mas tem a vantagem de ser computacionalmente mais eficiente.

O mapeamento de cubo também é superior ao formato equiretangular devido a que introduz menor distorão no mapeamento esférico, mas em contrapartida temos que lidar com as 6 cartas provenientes das 6 faces do cubo de projeção. Dependendo to tipo de textura e da filtragem realizada, as dirtorsões nos polos resultam imperceptíveis no mapeamento equiretangular, mas em alguns casos podem ser desagradáveis.

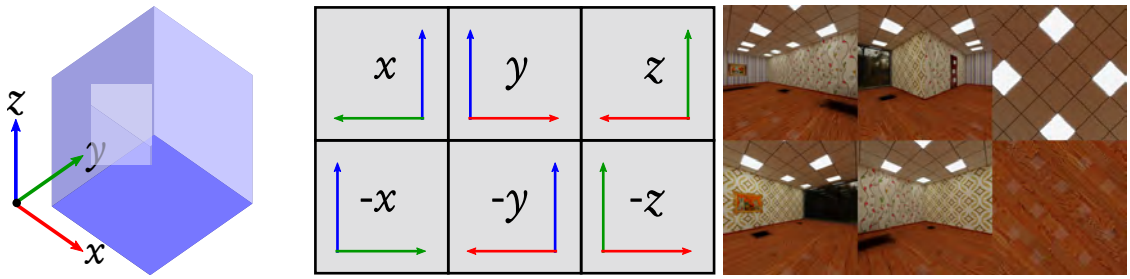
No formato de mapa de cubo (figura (A-3b)) a cena é representada como seis vistas quadradas em perspectiva, cada uma com um campo de visão de 90 graus, o que é equivalente a

projetar o ambiente em um cubo e, em seguida, desdobrá-lo. Os seis quadrados desdobrados de maneira mais natural num formato de cruz vertical ou horizontal, mas também podem ser representado num retângulo 3×2 ou 6×1 para evitar perdas na área da imagem resultante. O mapeamento não conserva a área, áreas nos cantos das faces do cubo ocupam significativamente mais área de imagem por ângulo sólido que as áreas no centro de cada face. No entanto, ao contrário do mapeamento equiretangular, este alongamento é limitado: áreas angulares que são mapeadas nos cantos do cubo estão sobre-representadas por um fator de até $3\sqrt{3}$ de área em relação a regiões do centro.

O mapeamento de cubo requer seis fórmulas diferentes para fazer a conversão entre as direções do mundo e as coordenadas de imagem, dependendo da face do cubo na qual o pixel está localizado. Embora as equações incluem ramificações condicionais, elas podem ser mais eficiente na avaliação do que outros mapeamentos (que envolvem funções transcendentes tais como *asin* e $atan2$). Este formato de imagem é, por vezes, o mais conveniente para trabalhar, porque as linhas retas no ambiente permanecem retas na imagem (Embora geralmente há descontinuidades direcionais nos limites de face).

(a) Mapeamento equiretangular. **(b)** Mapa de cubo (cruz horizontal).

Figura A-2: Dois mapeamentos de ambiente amplamente utilizados em computação gráfica.



(a) Faces do mapeamento de cubo 3×2. **(b)** Mapeamento de cubo.

Figura A-3: Mapeamento de cubo 3×2.

Referências Bibliográficas

[1] B. Madeira A. Zang A. Schulz, M. Cicconet and L. Velho. Techniques for cg music video production: the making of dance to the music / play to the motion. Technical report, Technical Report TR-2010-04, Laboratorio VISGRAF - IMPA, March 2010.

[2] Edward H. Adelson and James R. Bergen. The plenoptic function and the elements of early vision. In *Computational Models of Visual Processing*, pages 3–20. MIT Press, 1991.

[3] Sameer Agarwal, Ravi Ramamoorthi, Serge Belongie, and Henrik Wann Jensen. Structured importance sampling of environment maps. *ACM Trans. Graph.*, 22(3):605–612, July 2003.

[4] Arthur Appel. Some techniques for shading machine renderings of solids. In *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*, AFIPS '68 (Spring), pages 37–45, New York, NY, USA, 1968. ACM.

[5] James F. Blinn. Models of light reflection for computer synthesized pictures. *SIGGRAPH Comput. Graph.*, 11(2):192–198, July 1977.

[6] James F. Blinn and Martin E. Newell. Texture and reflection in computer generated images. *Commun. ACM*, 19(10):542–547, October 1976.

[7] David Blythe, David Blythe, and David Blythe. Advanced graphics programming techniques using opengl, siggraph 1999 course notes, 1999.

[8] David Burke, Abhijeet Ghosh, and Wolfgang Heidrich. Bidirectional importance sampling for direct illumination. In *Proceedings of the Sixteenth Eurographics Conference on Rendering Techniques*, EGSR '05, pages 147–156, Aire-la-Ville, Switzerland, Switzerland, 2005. Eurographics Association.

[9] Loren Carpenter. The a-buffer, an antialiased hidden surface method. *SIGGRAPH Comput. Graph.*, 18(3):103–108, January 1984.

[10] Edwin Earl Catmull. *A Subdivision Algorithm for Computer Display of Curved Surfaces.* PhD thesis, 1974. AAI7504786.

[11] David Cline, Parris K. Egbert, Justin F. Talbot, and David L. Cardon. Two stage importance sampling for direct lighting. In *Proceedings of the 17th Eurographics Conference on Rendering Techniques*, EGSR '06, pages 103–113, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.

[12] J. COEN and P. DEBEVEC. Lightgen, hdrshop plugin. *http://www.ict.usc.edu/ jcohen/-lightgen/lightgen.html.*, 2001.

[13] R. L. Cook and K. E. Torrance. A reflectance model for computer graphics. *ACM Trans. Graph.*, 1(1):7–24, January 1982.

[14] A. Criminisi, P. Perez, and K. Toyama. Object removal by exemplar-based inpainting. In *CVPR Proceedings. IEEE Computer Society*, volume 2, pages II–721–II–728 vol.2, June 2003.

[15] Franklin C. Crow. Summed-area tables for texture mapping. *SIGGRAPH Comput. Graph.*, 18(3):207–212, January 1984.

[16] Mark De Berg, Marc Van Kreveld, Mark Overmars, and Otfried Cheong Schwarzkopf. *Computational geometry*. Springer, 2000.

[17] Paul Debevec. Rendering synthetic objects into real scenes: Bridging traditional and image-based graphics with global illumination and high dynamic range photography. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '98, pages 189–198, New York, NY, USA, 1998. ACM.

[18] Paul Debevec. A median cut algorithm for light probe sampling. In *ACM SIGGRAPH 2005 Posters*, SIGGRAPH '05, New York, NY, USA, 2005. ACM.

[19] Paul E. Debevec and Jitendra Malik. Recovering high dynamic range radiance maps from photographs. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '97, pages 369–378, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[20] Michael Deering, Stephanie Winner, Bic Schediwy, Chris Duffy, and Neil Hunt. The triangle processor and normal vector shader: A vlsi system for high performance graphics. *SIGGRAPH Comput. Graph.*, 22(4):21–30, June 1988.

[21] Paul J. Diefenbach and Norman I. Badler. Multi-pass pipeline rendering: Realism for dynamic environments. In *Proceedings of the 1997 Symposium on Interactive 3D Graphics*, I3D '97, pages 59–ff., New York, NY, USA, 1997. ACM.

[22] A.A. Efros and T.K. Leung. Texture synthesis by non-parametric sampling. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 2, pages 1033–1038 vol.2, 1999.

[23] D Felinto, AR Zang, and Luiz Velho. Production framework for full panoramic scenes with photorealistic augmented reality. In *Informatica (CLEI), 2012 XXXVIII Conferencia Latinoamericana En*, pages 1–10. IEEE, 2012.

[24] Dalai Quintanilha Felinto, Aldo René Zang, and Luiz Velho. Production framework for full panoramic scenes with photorealistic augmented reality. *CLEI Electronic Journal*, 16(3):8–8, 2013.

[25] Raphael A. Finkel and Jon Louis Bentley. Quad trees a data structure for retrieval on composite keys. *Acta informatica*, 4(1):1–9, 1974.

[26] Efi Fogel, Ophir Setter, and Dan Halperin. Exact implementation of arrangements of geodesic arcs on the sphere with applications. In *24th European Workshop on Computational Geometry*, pages 83–86, 2008.

[27] Ron Fosner. All aboard hardware t & l. *Game Developer*, 7(4):30–41, April 2000.

[28] E. GOLDSTEIN and R. NAGLE. 3d visual simulation. *Simulation 16*, pages 25–31, Jan. 1971.

[29] Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen. The lumigraph. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, pages 43–54, New York, NY, USA, 1996. ACM.

[30] Ned Greene. Environment mapping and other applications of world projections. *IEEE Comput. Graph. Appl.*, 6(11):21–29, November 1986.

[31] Tim Hall. A how to for using opengl to render mirrors. *comp.graphics.api.opengl newsgroup*, August 1996.

[32] Paul S. Heckbert. Fundamentals of texture mapping and image warping. Technical Report UCB/CSD-89-516, EECS Dep., University of California, Berkeley, Jun 1989.

[33] Mathematical Applications Group Inc. 3d simulated graphics offered by service bureau. *Datamation*, 13(1):69, 1968.

[34] James T. Kajiya. The rendering equation. *SIGGRAPH Comput. Graph.*, 20(4):143–150, August 1986.

[35] Kevin Karsch, Varsha Hedau, David Forsyth, and Derek Hoiem. Rendering synthetic objects into legacy photographs. *ACM Trans. Graph.*, 30(6):157:1–157:12, December 2011.

[36] Douglas S. Kay. Transparency, refraction and ray tracing for computer synthesized images. Master's thesis, Cornell University, 1979.

[37] Thomas Kollig and Alexander Keller. Efficient illumination by high dynamic range images. In *Proceedings of the 14th Eurographics Workshop on Rendering*, EGRW '03, pages 45–50, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[38] P. Kuliyev. High dynamic range imaging for computer imagery applications - a comparison of acquisition techniques. In *Bachelor thesis at the Department of Imaging Sciences and Media Technology*, Cologne, 2009. University of Applied Sciences.

[39] Andrew Lauritzen. Deferred rendering for current and future rendering pipelines. *SIGGRAPH Course: Beyond Programmable Shading*, pages 1–34, 2010.

[40] Marc Levoy and Pat Hanrahan. Light field rendering. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, pages 31–42, New York, NY, USA, 1996. ACM.

[41] Lin Liang, Ce Liu, Ying-Qing Xu, Baining Guo, and Heung-Yeung Shum. Real-time texture synthesis by patch-based sampling. *ACM Trans. Graph.*, 20(3):127–150, July 2001.

[42] Leonard McMillan and Gary Bishop. Plenoptic modeling: An image-based rendering system. In *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '95, pages 39–46, New York, NY, USA, 1995. ACM.

[43] Xing Mei, Marc Jaeger, and Baogang Hu. An effective stratified sampling scheme for environment maps with median cut method. In *Proceedings of the International Conference on Computer Graphics, Imaging and Visualisation*, CGIV '06, pages 384–389, Washington, DC, USA, 2006. IEEE Computer Society.

[44] G. S. Miller and C. R. Hoffmanm. Illumination and reflection maps: Simulated objects in simulated and real environments. In *Curse Notes for Advanced Computer Graphics Animation*. SIGGRAPH, 1984.

[45] R. A. Newcombe and A. J. Davison. Live dense reconstruction with a single moving camera. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 1498–1505, June 2010.

[46] Richard A Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J Davison, Pushmeet Kohi, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on*, pages 127–136. IEEE, 2011.

[47] Victor Ostromoukhov, Charles Donohue, and Pierre-Marc Jodoin. Fast hierarchical importance sampling with blue noise properties. *ACM Trans. Graph.*, 23(3):488–495, August 2004.

[48] M. Pharr and G. Humphreys. *Physically Based Rendering: From theory to Implementation*. Morgan Kaufmann Publishers, Burlington, USA, 2nd edition, 2010.

[49] Matt Pharr and Greg Humphreys. Infinite area light source with importance sampling. *Pysically Based Rendering: From Theory to Implementation, PBRT Plugins from the Authors, http://pbrt. org/plugins/infinitesample. pdf*, 2004.

[50] Matt Pharr and Greg Humphreys. *Physically based rendering: From theory to implementation*. Morgan Kaufmann, 2004.

[51] Bui Tuong Phong. Illumination for computer generated pictures. *Commun. ACM*, 18(6):311–317, June 1975.

[52] Fabio Policarpo, Francisco Fonseca, and CheckMate Games. Deferred shading tutorial.

[53] Randi J Rost, Bill Licea-Kane, Dan Ginsburg, John M Kessenich, Barthold Lichtenbelt, Hugh Malan, and Mike Weiblen. *OpenGL shading language*. Pearson Education, 2009.

[54] R. Tsai. A versatile camera calibration technique for high-accuracy 3d machine vision metrology using off-the-shelf tv cameras and lenses. *IEEE Journal on Robotics and Automation*, 3(4):323–344, August 1987.

[55] Roger Y Tsai. An efficient and accurate camera calibration technique for 3d machine vision. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition, 1986*, 1986.

[56] Luiz Velho, Jonas Gomes, and M Sobreiro. Color image quantization by pairwise clustering. In *Proceedings of the 10th Brazilian symposium on computer graphics and image processing*, pages 203–207. Citeseer, 1997.

[57] Turner Whitted. An improved illumination model for shaded display. *Commun. ACM*, 23(6):343–349, June 1980.

[58] Lance Williams. Pyramidal parametrics. *SIGGRAPH Comput. Graph.*, 17(3):1–11, July 1983.

[59] A. Zang. Hybrid sampling machine (*HSM*). VisGraf Laboratory, *IMPA*. Software, 2009. http://www.impa.br/~zang/hsm.

[60] A. R. Zang and L. Velho. Arluxrender project. Visgraf Laboratory, *IMPA*, 2011. http://www.impa.br/∼zang/arlux.

[61] Aldo René Zang. Esquema híbrido para amostragem de mapas de iluminação em renderizações foto-realistas. Master's thesis, Rio de Janeiro, 2009.

[62] Aldo René Zang, Dalai Felinto, and Luiz Velho. Augmented reality using full panoramic captured scene light-depth maps. In *SIGGRAPH Asia 2012 Posters*, SA '12, pages 28:1–28:1, New York, NY, USA, 2012. ACM.

[63] Aldo René Zang, Dalai Felinto, and Luiz Velho. Rendering synthetic objects into full panoramic scenes using light-depth maps. In *GRAPP & IVAPP 2013: Proceedings of the International Conference on Computer Graphics Theory and Applications and International Conference on Information Visualization Theory and Applications, Barcelona, Spain, 21-24 February, 2013.*, pages 209–212, 2013.

[64] Aldo René Zang and Luiz Velho. Um framework para renderizações foto-realistas de cenas com realidade aumentada. In *XXXVII Latin American Conference of Informatics (CLEI), Quito*, 2011.

[65] Aldo René Zang and Luiz Velho. Panoramas *RGBD* omnidirecionais com múltiplas camadas e suas aplicações. In *CONFERENCE ON GRAPHICS, PATTERNS AND IMAGES, 28. (SIBGRAPI)*, Porto Alegre, 2015. Sociedade Brasileira de Computação.