Tássio Knop de Castro

Sombras Realistas em Realidade Aumentada Móvel

Rio de Janeiro
02 de Maio de 2012

Tássio Knop de Castro

Sombras Realistas em Realidade Aumentada Móvel

Dissertação apresentada ao Instituto Nacional de Matemática Pura e Aplicada, para a obtenção de Título de Mestre em Matemática: Opção Computação Gráfica.

Orientador: Luiz Henrique de Figueiredo
Co-Orientador: Luiz Velho

Rio de Janeiro
02 de Maio de 2012

Tássio Knop de Castro
Sombras Realistas em Realidade Aumentada Móvel
47 páginas
Dissertação (Mestrado) - Instituto de Matemática Pura e Aplicada.

1. Realidade Aumentada Móvel;
2. Renderização;
3. Sombras Suaves;

I. Instituto de Matemática Pura e Aplicada.

"There are only two worlds –
your world, which is the real world, and other worlds, the fantasy.
Worlds like this are worlds of the human imagination:
their reality, or lack of reality, is not important.
What is important is that they are there. These worlds provide an alternative.
Provide an escape.
Provide a threat.
Provide a dream, and power; provide refuge, and pain.
They give your world meaning.
They do not exist; and thus they are all that matters."

(Neil Gaiman)

Agradecimentos

Aos meus pais, Lúcia e Jorge, por torcerem por mim em todos os momentos.

À Laís, pelo amor e compreensão.

Aos professores, Luiz Henrique de Figueiredo, pela orientação e motivação em diversos momentos, e Luiz Velho, pelas inspiradoras conversas e grandes idéias ao longo do mestrado.

A todos os amigos do VISGRAF que estiveram próximos ao longo desta dissertação e das publicações durante o mestrado, que muito me ajudaram e compartilharam bons momentos: Leo Koller, Francisco Benavides, Lucas Reis, Marcelo Cicconet, André Máximo, Djalma Lucio, Juliano Kestenberg, Leandro Cruz, Francisco Ganacim, Leo Carvalho, Aldo Zang.

Aos amigos também apaixonados por Computação Gráfica, pelas muitas idéias e valiosas discussões: Thales Sabino, Eder Perez, João Paulo Peçanha, João Paulo Scoralick e Marco Aurélio.

A todos os professores e funcionários do IMPA, por sua responsabilidade e presteza.

A todos os amigos que fiz durante o mestrado.

Resumo

Descrevemos um método para sintetizar objetos virtuais com sombras realistas, tanto em cenas internas quanto em cenas externas. Sintetizamos sombras suaves em tempo real em plataformas móveis e discutimos as possibilidades, limitações e os detalhes de implementação inerentes a tais plataformas.

Palavras-chave: Realidade Aumentada Móvel; Renderização; Sombras Suaves

Abstract

We describe a method to generate virtual objects with realistic shadows, either in indoor and in outdoor scenes. We create smooth shadows in real time on mobile platforms and discuss the possibilities, limitations and specific implementation details of these platforms.

Keywords: Mobile Augmented Reality, Rendering, Smooth Shadows

Lista de Figuras

Sumário

Capítulo 1

Introdução

"Reality is merely an illusion, albeit a very persistent one."

(Albert Einstein)

O uso de sombras é muito importante para aplicações que requerem síntese realista de imagens. As sombras são essenciais para que possamos entender as relações espaciais entre objetos na cena: suas localizações, formas e volumes (Jacobs & Loscos, 2006). A geração de sombras tem sido estudada com muita atenção nos últimos anos, especialmente para aplicações em tempo real, que anseiam por realismo mas estão limitadas por considerações de desempenho.

O desenvolvimento das GPUs, tanto em programabilidade quanto em desempenho, tem favorecido as pesquisas que buscam conciliar realismo e eficiência. Em especial, sombras suaves que possuem aparência convincente e que possam ser geradas em tempo real são o objeto de desejo de muitos pesquisadores da área (Donnelly & Lauritzen, 2006), (Haller *et al.*, 2003), (Reeves *et al.*, 1987).

Quando se trata de dispositivos móveis, como em aplicações de Realidade Aumentada, as limitações são ainda mais severas: tanto processamento quanto memória devem ser considerados com cuidado. Felizmente, nos últimos anos, muitas plataformas móveis estão sendo equipadas com GPUs, o que nos dá a facilidade do suporte a *shaders*

e OpenGL ES.

1.1 Objetivos

Neste trabalho, descrevemos um sistema de Realidade Aumentada Móvel que gera, em tempo real, sombras realistas de objetos virtuais inseridos em ambientes reais, internos e externos, como mostra a Fig. 1.1. Essa tarefa é divida em duas partes:

- Capturar a direção das fontes de luz do ambiente em tempo real, sintetizando as sombras de forma coerente. A aquisição das fontes de luz é descrita no Cap. 2.
- Sintetizar sombras de aparência suave, como é detalhado no Cap. 3.

Até onde sabemos, não existe outro trabalho na literatura focado em estimação da fonte de luz em tempo real e síntese de sombras suaves coerentes com as condições de iluminação para Realidade Aumentada móvel. Acreditamos que esse tipo de realismo será um padrão na indústria em breve, pois tornam a Realidade Aumentada mais poderosa e imersiva em dispositivos amplamente disponíveis, como smartphones e tablets. Além disso, os dispositivos móveis surgem com cada vez mais sensores e hardware que permitem mais realismo e praticidade às aplicações futuras.

1.2 Realidade Aumentada

Azuma (Azuma, 1997) define a Realidade Aumentada como um sistema que:

- combina elementos virtuais com o ambiente real;
- é interativa e tem processamento em tempo real;
- é concebida em três dimensões.

Figura 1.1: Objeto real (à esquerda) e objeto sintetizado (à direita) com sombra realista, calculada automaticamente, em uma cena externa.

Existem inúmeras maneiras de combinar elementos virtuais com o ambiente real. Em particular, um grande objetivo dentro da Realidade Aumentada é a inserção convincente de objetos virtuais no mundo real, de forma que o usuário não consiga distinguir virtual de real. Sistemas com maior grau de realismo podem ser usados para simulações, educação, treinamentos e na indústria do entretenimento, como a indústria de jogos em tempo real.

Como será discutido no Cap. 3, muitas vezes é necessário um equilíbrio sutil entre eficiência e realismo. Portanto, em prol da velocidade existem situações onde busca-se obter resultados convincentes, e não necessariamente exatos.

1.3 Sombras em Realidade Aumentada

Uma boa iluminação é essencial para o entendimento de uma cena. Experimentos como (Kersten *et al.*, 1996) e (Gooch *et al.*, 1999) mostram que mesmo a inclusão de sombras simples é capaz de tornar uma cena muito mais realista, introduzindo relações espaciais entre os objetos iluminados e deixando claras noções de profundidade, altura e movimento.

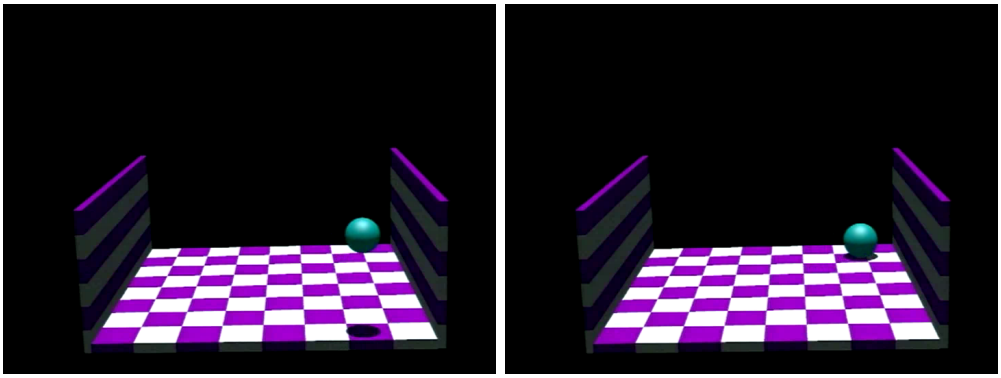

Figura 1.2: Esfera ocupando a mesma posição nas duas imagens e sombra ocupando posições diferentes: a percepção da posição da esfera é consequência direta da posição da sombra. Imagens extraídas do vídeo de (Kersten *et al.*, 1996).

Tomemos a Fig. 1.2 como exemplo. A posição da esfera projetada na tela é a mesma nas duas imagens, o que varia é a posição de sua sombra. A altitude do objeto é inferida a partir da posição de sua sombra. Esse tipo de informação permite que uma pessoa interprete rapidamente a cena, sem ambiguidades, e portanto é de grande valor no contexto de Realidade Aumentada.

1.4 Trabalhos relacionados

(Sugano *et al.*, 2003) realizaram diversos estudos e testes de percepção comparando ambientes de Realidade Aumentada com e sem sombras, e com sombras realistas e não realistas. Eles concluíram, através dos experimentos, que a presença de sombras

aumenta significativamente a sensação de presença e realismo de um objeto virtual. Também observaram que a posição da fonte de luz virtual pode se desviar um pouco da posição da fonte real sem que o observador perceba. Isso é especialmente relevante para nós, pois como focamos em um método de inferir as fontes de luz em tempo real em dispositivos de baixa capacidade de processamento, é vantajoso que só seja necessária uma direção aproximada da luz: se precisássemos de estimativas muito precisas, seria mais difícil manter um alto *frame rate* necessário a aplicações em tempo real.

(Supan *et al.*, 2006) capturaram em tempo real uma esfera espelhada com distância fixa em relação ao marcador. Eles utilizaram duas câmeras, uma para o marcador e outra para a esfera. A partir disso, criaram um número fixo de fontes de luz, e mediram suas cores e intensidades no *cube mapping* produzido a partir da esfera. Eles realizaram *shadow mapping* (Williams, 1978) simples. O nosso método também utiliza uma distância fixa em relação ao marcador, mas com apenas uma câmera. Além disso, nosso método produz sombras mais suaves e com menos *aliasing*, como descrito no Cap. 3.

(Kanbara & Yokoya, 2002) também capturaram uma esfera e realizaram um *sphere mapping*, contando somente com uma correspondência entre a imagem capturada e a geometria da esfera. Eles criaram uma fonte de luz para cada ponto amostrado sobre a esfera. O nosso método também realiza um *sphere mapping*, mas seleciona uma fonte ou as fontes de luz mais representativas da cena: isso é importante devido às limitações de hardware dos dispositivos móveis.

(Pessoa *et al.*, 2010) realizaram a captura HDR do ambiente real, modelando um *irradiance environment map* que contém propriedades necessárias para fazer o shading do objeto virtual. Para produzir as sombras, eles utilizaram o método *shadow mapping* simples, sem *self-shadowing*. A captura HDR traz muitas informações úteis, mas é menos prática tanto em termos de hardware quanto de custo de software quando comparada a uma captura normal. O nosso método utiliza uma câmera de tablet ou

smartphone e realiza *self-shadowing* em tempo real. Existem trabalhos sobre a captura em tempo real de vídeo HDR para dispositivos móveis (Castro *et al.*, 2011), mas até onde sabemos, não existe ainda uma solução ótima em tempo real capaz de realizar tal tarefa; portanto realizamos uma captura convencional.

(Haller *et al.*, 2003) utilizaram *shadow volumes* para Realidade Aumentada, fixando previamente a distância e a posição da fonte de luz. Essa abordagem tem a vantagem de produzir sombras com menos *aliasing* que o *shadow mapping* convencional, mas são mais custosas (Eisemann *et al.*, 2011, pp.44–45) e mais sensíveis à complexidade da geometria (Donnelly & Lauritzen, 2006) que a abordagem que utilizamos: *shadow mapping* (Williams, 1978) e suas extensões *percentage closer filtering* (Reeves *et al.*, 1987) e *variance shadow mapping* (Donnelly & Lauritzen, 2006) adaptadas para dispositivos móveis.

Capítulo 2

Estimação da fonte de luz

Para uma iluminação consistente em Realidade Aumentada, precisamos de informação sobre as fontes de luz da cena real. A seguir, são descritas duas maneiras que implementamos para a aquisição da direção da fonte de luz: uma para cenas internas e outra para cenas externas.

No caso geral, não sabemos a distância da fonte de luz à cena. Existem maneiras de calcular tal distância (Goesele, 2004, pp.79–106), mas, optando pelo caminho mais prático, escolhemos assumir que a luz está distante o suficiente da cena, de modo que ela pode ser modelada como uma luz direcional.

2.1 Ambiente interno

O ambiente interno consiste em um marcador e uma esfera, também chamada de *light probe*, de tamanho e posição conhecidos, inseridos na cena desejada. Amostramos em tempo real a imagem da esfera e dela serão extraídas as informações necessárias para a estimação da direção da fonte de luz.

Uma característica singular do método aqui apresentado é que o objetivo é obter as n fontes de luz mais representativas, visando usar poucas fontes que resultem em

bons resultados. Como nos baseamos no método *shadow mapping* (Williams, 1978) para síntese das sombras, cada fonte adicionada à cena representa mais uma etapa de renderização. Dessa forma, obter poucas fontes com capacidade de representar as condições de iluminação da cena real é uma tarefa crucial para o bom desempenho do algoritmo.

Correspondência geométrica

Optamos por fixar a esfera em uma posição conhecida em relação ao marcador utilizado. Seu raio também é previamente mensurado. Dessa forma, como o marcador é a origem do sistema, obtemos da biblioteca de Realidade Aumentada a matriz de transformação model-view correspondente ao marcador. Basta fazer a translação correspondente à esfera para obtermos exatamente a matriz model-view que faz a correspondência entre uma esfera virtual e a posição da esfera real na tela, o que nos dá uma região I_e na imagem, como ilustra a Fig. 2.1.

Processamento da imagem capturada

Primeiramente, obtemos l_{max}, a luminância mais intensa de I_e. Para cada ponto $\mathbf{p}$ com tal luminância, prosseguimos com um algoritmo do tipo flood-fill para encontrar todos os pontos com luminância l_{max} numa vizinhança 4-conexa de $\mathbf{p}$. O processo se repete até que todas as componentes conexas de luminância máxima sejam encontradas, conforme ilustra a Fig. 2.2. Os conjuntos obtidos podem então ser ordenados a partir da quantidade de pixels, de forma que o conjunto com mais pixels representa a fonte de luz mais significativa da cena e assim por diante.

Para uma mesma cena, notamos que as esferas espelhadas contêm mais componentes conexas com luminância l_{max} do que a esfera difusa, como ilustra a Fig. 2.3. Evidentemente, como a esfera espelhada absorve muito menos luz do que a difusa, existem fontes de luz de diferentes intensidades que, ao serem refletidas e capturadas

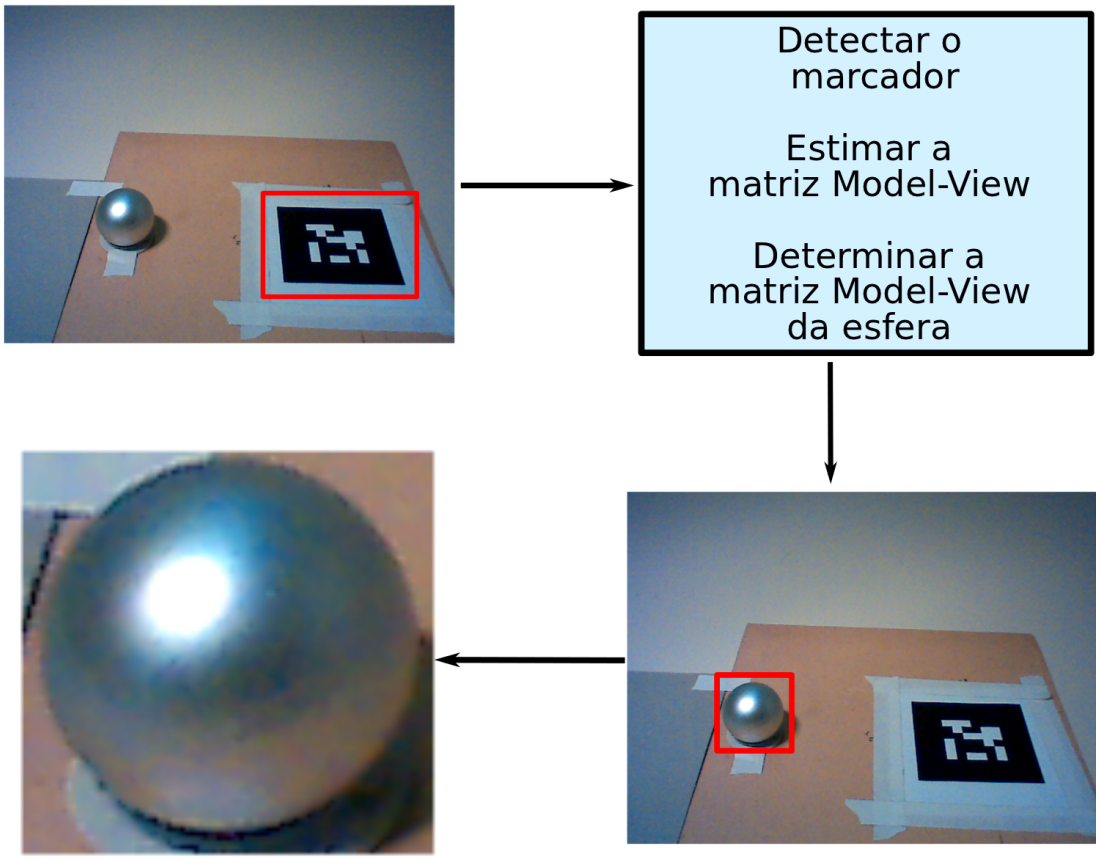

Figura 2.1: A partir de uma relação pré-estabelecida entre o marcador e a esfera amostrada, é imediata a correspondência entre o mundo virtual e a esfera capturada pela câmera. No experimento ilustrado, I_e possui 90×90 pixels.

pela câmera convencional, com um baixo *dynamic range*, são transformadas em pixels de intensidade máxima. Após essa captura, não se pode mais determinar qual fonte é mais intensa.

O que observamos na prática é que a esfera difusa, que reflete uma intensidade menor da luz recebida, atenua a intensidade das luzes capturadas pela câmera, que consegue então mapear diferentes luminâncias em diferentes valores de pixel e preservar a ordem das luminâncias. Isso faz com que tenhamos menos vizinhanças com valor l_{max}. Concluímos, assim, que a esfera difusa é mais vantajosa quando existem várias fontes de luz na cena, e a esfera espelhada é mais adequada quando o objetivo é uma maior

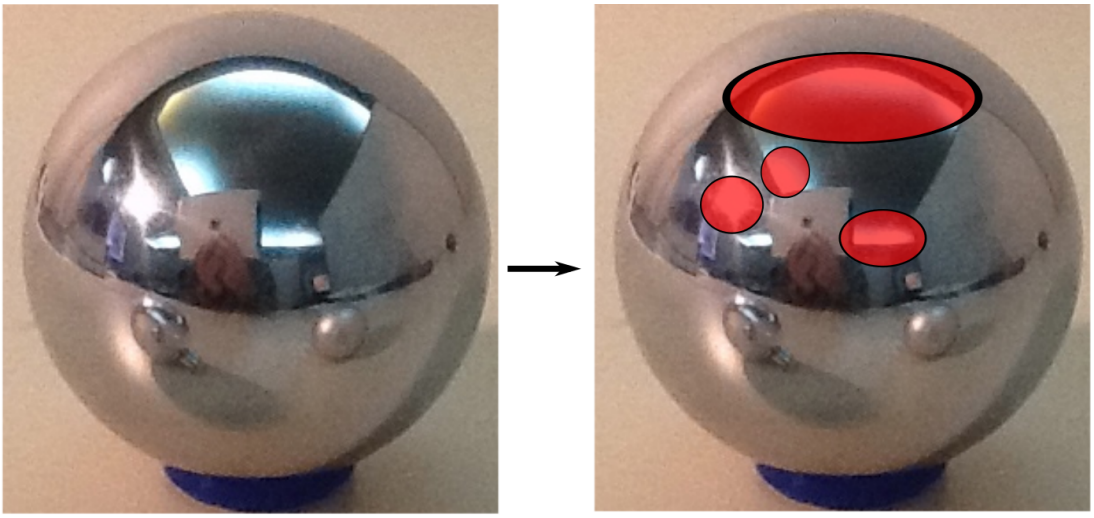

Figura 2.2: Determinando os conjuntos conexos de luminância l_{max}.

precisão na estimativa da direção da fonte de luz e detectar todas as componentes conexas.

Para calcularmos a direção da fonte de luz correspondente a um dado conjunto, testamos duas maneiras:

- *Domínio da esfera:* À medida que geramos a esfera a partir de coordenadas esféricas, salvamos um mapa de luminâncias indexado por u e v. Ou seja, para cada par (u,v), projetamos o vértice correspondente na imagem e salvamos a sua luminância. A partir de então, realizamos no domínio uv o processo descrito anteriormente para obter as vizinhanças com l_{max}. Obtida uma vizinhança, calculamos a média das coordenadas uv correspondentes, o que nos dá um ponto $n = (n_x,n_y,n_z)$ na superfície da esfera.

- *Domínio da imagem:* Sintetizamos uma esfera invisível sobre o quadro do vídeo (com a posição e dimensões do nosso *light probe*) e processamos toda a região I_e no domínio da imagem. Obtendo a maior vizinhança com l_{max}, fazemos a média das posições de cada pixel daquele conjunto. Isso nos dá coordenadas de janela (x_w,y_w). A partir delas, devemos "desprojetar" o pixel. Obtemos a profundidade

Figura 2.3: Foram utilizadas esferas espelhadas, esferas difusas e de tamanhos distintos nos experimentos. As pequenas são simplesmente bolas de natal.

correspondente àquele pixel lendo a profundidade salva no *z-buffer* na posição inteira mais próxima de (x_w,y_w). Aqui entra a importância da esfera sintetizada, pois o *z-buffer* terá armazenado a distância correta da câmera à esfera. Caso não fosse utilizada tal esfera, o *z-buffer* teria em (x_w,y_w) a distância do plano far. Em seguida, aplicamos a inversa da transformação model-view-projection, encontrando assim um ponto $n = (n_x,n_y,n_z)$ sobre a superfície da esfera. Esse é o ponto da esfera que será utilizado para calcularmos a posição da fonte de luz.

A vantagem de trabalhar no domínio da esfera é que a complexidade dessa etapa do algoritmo só depende de quanto optamos por discretizar a esfera, que é um valor fixo. Portanto, não importa se o *light probe* ocupa uma região maior ou menor da tela, o algoritmo tem o mesmo desempenho.

A vantagem de trabalhar no domínio da imagem é que pixels com l_{max} podem estar

próximos e mesmo assim representarem vizinhanças distintas. Como essa abordagem processa todos os pixels da esfera, é mais fácil detectar de forma mais precisa as regiões conexas. Como o domínio da esfera não avalia pontos realmente vizinhos na imagem, a discretização da esfera deve ser suficiente para compensar a distância entre cada ponto avaliado.

Considerando que o desempenho é uma prioridade nesse método, preferimos trabalhar no domínio da esfera. Independentemente da abordagem escolhida, calculamos um ponto n sobre a esfera que será utilizado nos cálculos descritos a seguir para obter a direção da fonte de luz. Fizemos uma amostragem uniforme sobre o domínio da esfera, pré-fixando a distância entre as amostras.

A transformação model-view MV proveniente do marcador nos dá a posição da câmera. Basta invertermos MV e salvarmos suas coordenadas de translação, obtendo assim as coordenadas de câmera $c = (c_x,c_y,c_z)$.

O último passo consiste em traçar um raio que parte de c até n. Refletimos esse raio em relação à normal no ponto n. O vetor resultante v_r nos dá a direção da fonte de luz (Fig. 2.4). Chamando de s o centro da esfera, temos que $v_r = -2(c_i.n_s)n - c_i$, onde $c_i = \widehat{n - c}$ e $n_s = \widehat{n - s}$.

2.2 Ambiente externo

Para cenas externas durante o dia, a fonte de luz de maior importância é o sol. A maioria dos dispositivos móveis atuais já possui sensores de grande utilidade além da câmera: a partir das coordenadas geográficas e orientação cardeal do dispositivo, podemos calcular a direção da luz do sol. Assumimos que nosso método será utilizado durante o dia e que o tempo está claro.

Para calcular a direção do sol relativa ao observador, é preciso usar um sistema de coordenadas celestial, análogo ao que usamos para latitude e longitude na Terra

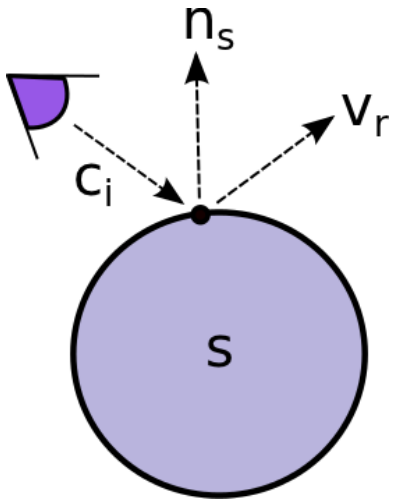

Figura 2.4: Obtendo a direção de uma fonte de luz em um ambiente interno.

(Thompson & Thompson, 2005, pp.116–118). Utilizamos o sistema de coordenadas horizontal, como ilustra a Fig. 2.5: define-se uma esfera centrada no observador, cujo pólo norte é chamado de *zênite* e o pólo sul de *nadir*. Os pontos de referência dessa esfera são o zênite e o ponto mais ao norte do plano fundamental da esfera. Um objeto pode ser localizado nesse sistema através das coordenadas Altitude (Alt) e Azimute (Az), expressas em graus.

A coordenada Alt representa o ângulo entre o plano fundamental da esfera e o objeto. Temos que $0 \leq Alt \leq 90$, aumentando à medida que o objeto se aproxima do zênite. A coordenada Az é o ângulo entre o norte do plano fundamental e o objeto, medido em sentido horário. Portanto $0 \leq Az \leq 360$.

Existem diversos métodos para o cálculo do vetor que indica a posição do sol em relação ao observador. Neste trabalho utilizamos o algoritmo PSA (Blanco-Muriel *et al.*, 2001), que é rápido e eficiente, necessitando somente das coordenadas de latitude/longitude e do dia e hora como dados de entrada. O algoritmo é listado no Apêndice A.

Realizamos experimentos com um dispositivo iPad 2 que, provido de calendário, GPS e relógio, fornece as informações necessárias ao algoritmo e calcula em tempo real a posição do sol, utilizando intensidade constante de iluminação. Consideramos os

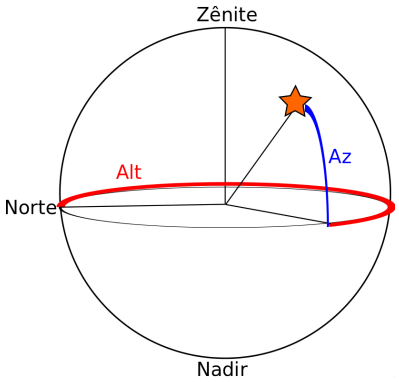

Figura 2.5: Sistema de coordenadas horizontal. Altitude representada pelo arco azul, e Azimute pelo arco vermelho.

resultados bastante coerentes com a iluminação real, como veremos na Seção 4.2.

Capítulo 3

Síntese de sombras

Os principais algoritmos para síntese de sombras em tempo real são o *shadow mapping*, de Williams (Williams, 1978), e o *shadow volumes*, de Crow (Crow, 1977). Na literatura existem várias comparações entre os dois (Eisemann *et al.*, 2011, p.73); numa análise velocidade versus precisão, a principal vantagem do *shadow mapping* é seu menor custo computacional, enquanto o *shadow volumes* possui o ponto forte de gerar sombras sem *aliasing*.

Como este trabalho foca em plataformas móveis, limitadas em termos de recursos computacionais, optamos pelo método mais rápido: o *shadow mapping* e suas extensões para sombras suaves. A seguir, descrevemos os métodos utilizados e os detalhes de implementação específicos das plataformas móveis. Nossos resultados obtidos com cada método podem ser vistos no Cap. 4.

3.1 *Shadow mapping*

Para determinar a iluminação de um certo ponto 3D, basta verificarmos se ele é visível ou não a partir da fonte de luz. O método *shadow mapping* (Williams, 1978) consiste em sintetizar uma imagem I da cena a partir do ponto de vista da fonte de luz (Fig. 3.1).

Figura 3.1: Algoritmo de *shadow mapping* (adaptado de (Eisemann *et al.*, 2011˙ p.33))

Todos os pontos que estiverem contidos em I estarão iluminados, e aqueles não visíveis estarão na sombra. Esse método é bastante eficiente, já que sua complexidade é próxima à complexidade da renderização de uma cena.

O primeiro passo do método de *shadow mapping* é criar uma imagem a partir de uma câmera colocada na posição da fonte de luz. Definimos assim uma transformação projetiva, onde uma transformação perspectiva corresponde a uma luz do tipo spot, e uma transformação ortográfica corresponde a uma luz direcional. O conteúdo dessa imagem é um mapa de profundidades, denominado *shadow depth map* ϕ. Cada pixel de ϕ armazena a profundidade da primeira superfície visível a partir da luz. O cálculo de ϕ é muito eficiente, pois a profundidade já é normalmente calculada pela GPU na etapa do cálculo de visibilidade.

O segundo passo é gerar uma imagem a partir da câmera real. Para cada fragmento de posição $\mathbf{p}$ rasterizado, aplicamos a transformação projetiva da luz, obtendo sua posição $\mathbf{p}^L$ no espaço de clip da luz. Basta compararmos $z(p^L)$, a profundidade do fragmento $\mathbf{p}$ no espaço de clip da luz, com $\phi(p^L)$, a profundidade armazenada em ϕ na posição correspondente a $\mathbf{p}^L$. Como $\phi(p^L)$ é a distância da luz à primeira superfície visível, é imediato que se $z(p^L) > \phi(p^L)$, então $\mathbf{p}$ não está iluminado, e portanto deve

ser sintetizado como sombra. Caso contrário, o ponto está iluminado.

Dessa forma, o *shadow mapping* é uma função $S(z, \phi)$ que retorna 0 caso o fragmento esteja em sombra e 1 caso contrário. Os principais problemas desse método são relacionados à amostragem das profundidades e ao *aliasing*. Esses problemas estão intimamente ligados com o fato de dependermos da resolução de ϕ para o cálculo de visibilidade.

3.1.1 *Depth Bias*

A etapa de amostragem gera um mapa de profundidades ϕ que, como já dissemos, é composto por pixels. Também geramos uma imagem para o ponto de vista da câmera da cena, para compararmos z com ϕ. Essas duas imagens são, naturalmente, discretas, e aí está um importante detalhe: devemos nos lembrar que o valor de cada pixel é determinado unicamente pela amostra da cena correspondente ao centro do pixel. Assim, quando comparamos z com ϕ, dificilmente estaremos comparando o mesmo ponto da cena visto de dois pontos de vista diferentes; estaremos, na verdade, comparando pontos próximos. Isso pode causar problemas quando a amostra da câmera (*view sample*) estiver um pouco mais distante da fonte de luz do que a amostra da luz (*light sample*), como ilustra a Fig. 3.2, porque causa o surgimento incorreto de sombras na superfície do objeto iluminado.

Uma solução para o *z-fighting* é inserir um *depth bias*, que é basicamente um *offset* que aumenta um pouco a profundidade das amostras da luz. Assim aumentamos a probabilidade de que a profundidade da amostra da câmera seja menor do que a profundidade da amostra da luz. A maneira de definir o *depth bias* é muito sutil; quanto maior a curvatura da superfície, vista a partir do ponto de vista da luz, maior será a variação de profundidade entre pixels vizinhos, e consequentemente maior será o *bias* necessário. Já se utilizarmos um *bias* muito elevado, podemos ocasionar um *light leaking*, como ilustra a Fig. 3.3, que é o afastamento da sombra em relação ao objeto iluminado.

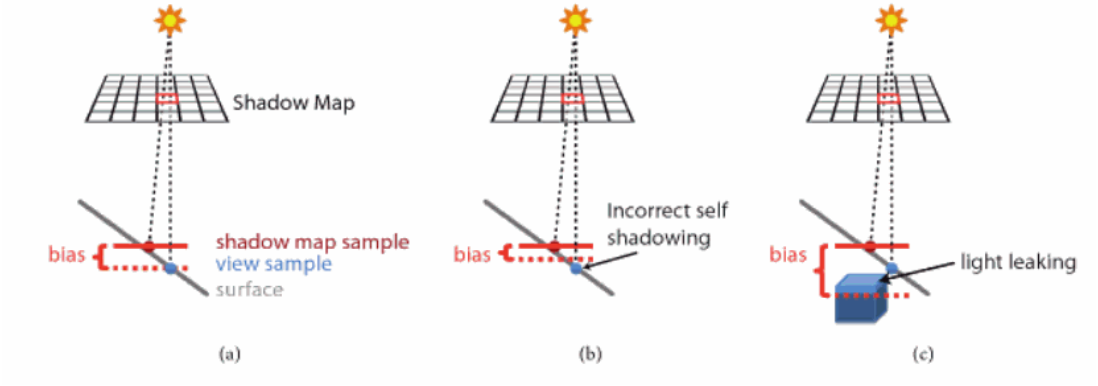

Figura 3.2: *Depth bias* (extraído de (Eisemann *et al.*, 2011˙ p.33)). a) a amostra da câmera está mais distante do que a amostra da luz, portanto será incorretamente sombreada. b) um *bias* muito pequeno não é suficiente para resolver o *z-fighting*. c) um bias muito grande causa o *light leaking*, que é uma distância evidente entre a sombra e o objeto iluminado.

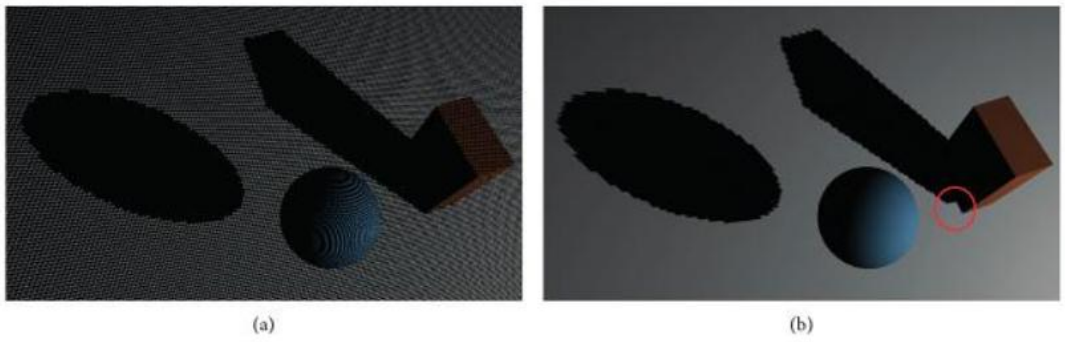

Figura 3.3: *z-fighting* e *light leaking* (extraído de (Eisemann *et al.*, 2011˙ p.33))

A solução mais comum para o *z-fighting* é ajustar manualmente o *depth bias* até que o resultado esteja visualmente aceitável, e é isso o que fazemos no presente trabalho. Uma interessante e simples maneira de amenizar o *z-fighting* é linearizar o *depth buffer*, como é o proposto em (Brabec *et al.*, 2000).

No caso geral, a profundidade é modelada de forma que tenhamos maior precisão nos planos mais próximos do plano *near* da câmera. Isso é muito útil quando queremos eliminar superfícies não visíveis, mas não é necessariamente interessante para o *shadow mapping*, pois pontos distantes da fonte de luz podem estar muito próximos da câmera

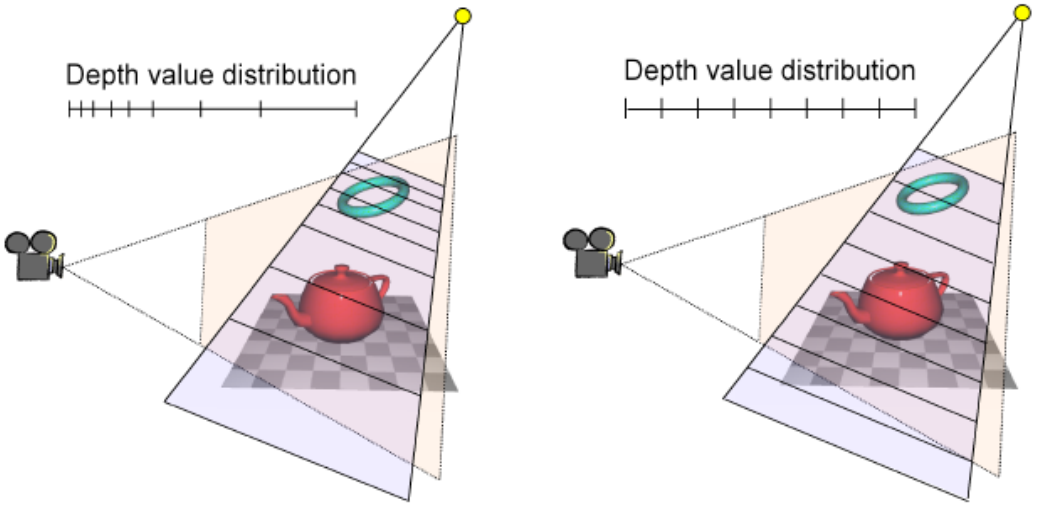

Figura 3.4: Esquerda: distribuição padrão da profundidade. Direita: profundidade linearizada (extraído de (Brabec *et al.*, 2000˙ p.33))

da cena.

Como ilustra a Fig. 3.4, linearizar a distribuição das profundidades deixa a etapa de comparação de z com ϕ mais equilibrada.

3.1.2 *Aliasing*

Como o *shadow mapping* é uma representação da cena baseada em imagem, estamos sujeitos à resolução finita do mapa de profundidades. A limitação da resolução de ϕ acarreta em *aliasing*, que se manifesta na forma de descontinuidades com uma aparência quadriculada nas bordas da sombra (*staircase effect*). Isso acontece porque muitas amostras da câmera podem ser projetadas em um mesmo *texel* do mapa de profundidades e, consequentemente, obter uma resposta similar no cálculo de sombras. Quando as próximas amostras da câmera são comparadas com o texel vizinho, o resultado da função de sombra pode ser completamente diferente, e isso causa a aparência quadriculada da fronteira da sombra no *shadow mapping*, como vemos na Fig. 3.5.

Na prática, muitos jogos atuais utilizam *shadow maps* com resoluções muito altas,

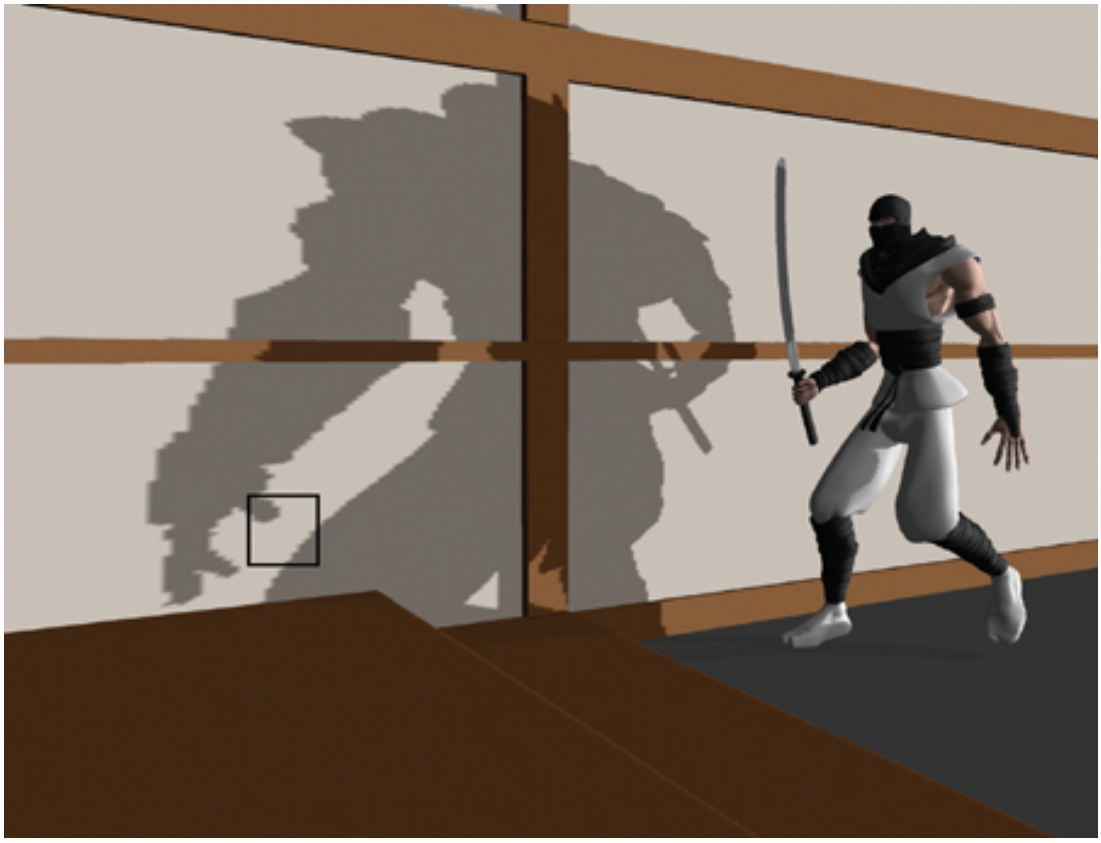

Figura 3.5: Sombra com aliasing (extraído de (Fernando, 2004˙ cap.11))

como 4096^2 e 8192^2, para reduzir a chance de que duas amostras da câmera sejam projetadas em um mesmo texel do *shadow map*. Como nosso objetivo é um aplicativo em tempo real em um dispositivo móvel, essa solução é pouco viável. A seguir, descrevemos maneiras que utilizamos para amenizar o *aliasing*.

3.2 Síntese de sombras suaves

Para aplicações em tempo real, é comum usarmos algum tipo de filtragem para produzir uma sombra mais suave. Isso é particularmente útil para amenizar erros devido à insuficiente resolução de ϕ e tornar o *aliasing* menos evidente. Em geral, os métodos de filtragem suavizam as bordas da sombra projetada, o que gera um resultado

mais agradável aos olhos, pois permite uma transição suave entre áreas iluminadas e áreas em sombra. Entretanto, esse tipo de abordagem não tem embasamento físico: se trata apenas de uma maneira de contornar os problemas introduzidos pelo *aliasing* sem comprometer o desempenho desejado.

Não faz sentido, contudo, filtrar o mapa de profundidades ϕ. Isso causaria problemas principalmente nas regiões de descontinuidades, onde objetos em sombra poderiam ser erroneamente iluminados e vice-versa.

Implementamos duas maneiras de gerar sombras suaves a partir do *shadow mapping*. O método mais conhecido é o *percentage closer filtering* (PCF), proposto por Reeves et al. em 1987 (Reeves *et al.*, 1987). Sua abordagem consiste em filtrar não a profundidade ϕ, mas a função de geração de sombra S. Outro método que chamou nossa atenção é o *variance shadow mapping* (VSM) proposto por Donnely e Lauritzen em 2006 (Donnelly & Lauritzen, 2006). Essa abordagem consiste em estimar a probabilidade de um determinado fragmento estar em sombra, e é um método que no caso geral é mais eficiente e que dá resultados tão bons quanto o PCF.

Percentage closer filtering

Como mencionado acima, devemos filtrar não a profundidade ϕ, mas a função de geração de sombra S:

$$S_{pcf}(z,\phi) = \sum_{p_i \in K} k(p_i - p) S(z(p_i), \phi(p_i)), \tag{3.1}$$

onde k é um filtro passa-baixa bidimensional com janela K.

Como pode-se ver pela Eq. 3.1, não é possível pré-computar esse resultado, pois a filtragem é feita a partir do teste de profundidade (descrito na Seção 3.1), que deve ser reavaliado a todo instante em uma cena dinâmica.

Esse método requer um equilíbrio delicado, pois se usarmos uma janela grande (em

geral, maior que 3×3), os custos dessa técnica começam a pesar demais, enquanto uma janela pequena não deixa a sombra suficientemente suave. Para que o desempenho se mantenha em tempo real, utilizamos uma janela 3×3.

Variance shadow mapping

Esse poderoso método é um belo exemplo do uso de estatística para estimar as sombras e pré-computar o *shadow map*: em vez de amostrar cada pixel no mapa, é possível estimar a distribuição das profundidades dentro da janela do filtro, usando somente a média μ e a variância σ das amostras de profundidade de ϕ. Para obter tais valores, primeiramente é armazenado, além do mapa de profundidades ϕ, um mapa Φ com o quadrado das profundidades. Em seguida, um simples filtro passa-baixa aplicado a ϕ e Φ é suficiente para estimar μ e σ localmente. Na aplicação desenvolvida, utilizamos um *blur* gaussiano e experimentamos diversos tamanhos de filtro. Após essa filtragem, é possível calcular o primeiro e o segundo momento M_1 e M_2, onde $M_1 = \mu$ e $M_2 = \sigma + M_1^2$.

Para estimar a intensidade das sombras, é usada a desigualdade de Chebyshev, que impõe limites para a distribuição. Mais precisamente, temos que ao observar uma distribuição de profundidades no *shadow map* com média μ e variância σ, a probabilidade $P(d \leq z)$ de uma profundidade aleatória z ser maior do que a profundidade d presente no mapa ϕ tem o seguinte limite superior:

$$P(d \leq z) \leq p(d) = \frac{\sigma^2}{\sigma^2 + (d - \mu)^2} \tag{3.2}$$

Isso nos dá a estimativa de que uma fração $p(d)$ do *shadow map* observado na janela atual está mais distante da luz do que outro objeto na posição $\mathbf{p}$. Essa versão da desigualdade de Chebyshev é válida para $d > \mu$. Caso contrário, consideramos que o ponto $\mathbf{p}$ está totalmente iluminado.

3.3 Detalhes de implementação

Os três métodos de geração de sombra descritos nesta seção (*Shadow Mapping*, PCF e VSM) foram implementados em três plataformas, um desktop e duas móveis: um iPad 2, um Nokia N900 e um desktop. Tanto o iPad 2 quanto o Nokia possuem uma GPU da família PowerVR SGX, com suporte a OpenGL ES 2.0.

As aplicações para o Nokia e o desktop foram desenvolvidas em C++; para o iPad 2 utilizamos Objective-C++. No Nokia e no iPad 2, a textura possui suporte para somente 8 bits por canal de cor, enquanto no desktop é possível usar 32 bits por canal. O *z-buffer* nos dispositivos móveis pode possuir no máximo 24 bits, enquanto no desktop podemos usar até 32 bits. Além do mais, nos dispositivos móveis a textura é armazenada no formato de *unsigned byte*, enquanto que no desktop podemos utilizar pontos flutuantes nativamente. Isso significa que, se utilizarmos somente uma textura para salvar ϕ e Φ, no desktop podemos utilizar um canal de cor para armazenar ϕ e outro canal para Φ e ainda assim teremos uma boa precisão (32 bits para cada canal). Já nos dispositivos móveis, se utilizarmos somente uma textura, só teremos 8 bits para cada valor de ϕ e 8 bits para cada valor de Φ. Esse detalhe técnico ilustra claramente algumas das dificuldades inerentes aos *smartphones* e *tablets*.

Uma possível solução para obter mais precisão nos dispositivos móveis seria utilizar *Multiple Render Targets* (Rost *et al.*, 2009, p.239), que é uma maneira de escrever, durante a execução de um *shader*, em mais de uma textura ao mesmo tempo. Isso é possível no desktop, mas não no OpenGL ES 2.0, pois ele suporta somente um *color attachment*.

Uma outra maneira possível é utilizar dois *framebuffer objects* (FBO), um para ϕ e outro para Φ. Isso sim é permitido em todas as plataformas que utilizamos, mas possui a desvantagem de, para cada FBO utilizado, é necessária mais uma etapa de renderização, o que prejudica o desempenho global do método.

Poderíamos utilizar a solução mais simples, que consiste em utilizar 8 bits para z e 8 bits para z^2. Contudo, não estaríamos explorando da melhor maneira o potencial do hardware, já que a textura possui 32 bits disponíveis.

A solução que utilizamos consiste em fazer o empacotamento de bits, também chamado de *bit packing*, dos valores z e z^2. Essa solução consiste em rearranjar os bits de um dado valor de forma que ele possa ser representado através de um conjunto de variáveis que contém menos bits. No nosso caso, por exemplo, queremos fazer com que z e z^2 sejam armazenados em uma textura de 32 bits: utilizamos um *depth buffer* de 16 bits e, para salvar z e z^2 na textura RGBA do *shadow map*, que possui 8 bits por canal, empacotamos os bits de z nos dois primeiros canais da textura (R e G) e empacotamos z^2 nos dois últimos canais (B e A). O código para o *bit packing* é listado no Apêndice B.

A desvantagem mais notável ao utilizarmos *bit packing* é que teremos mais processamento na etapa de *blur*, pois para cada pixel do *shadow map*, temos que desempacotar, filtrar e empacotar novamente todos os pixels na janela do filtro gaussiano. Em contrapartida, temos uma boa precisão para armazenar z e z^2 em uma só textura, que é uma solução com bom desempenho e precisão.

Capítulo 4

Resultados

Para a Realidade Aumentada, utilizamos as bibliotecas Vuforia AR (Vuforia AR SDK, 2012) e ARToolKitPlus (ARToolKitPlus, 2012), que através de um marcador ou de uma determinada imagem, fornecem a transformação model-view que leva um objeto virtual à correspondente posição no mundo real.

Para o ambiente interno, realizamos experimentos com esferas espelhadas e esferas difusas, grandes e pequenas, como mostra a Fig. 2.3. Os dispositivos utilizados foram um iPad 2 com iOS 5, um Nokia N900 com sistema operacional Maemo, e uma versão desktop do algoritmo para o sistema operacional Ubuntu.

Não é fácil estabelecer métricas que comparem a eficácia de um método baseado em Realidade Aumentada e síntese de sombras, portanto avaliamos a sensação visual de realismo e imersão gerados.

O objetivo dos experimentos, tanto nas cenas internas quanto nas cenas externas, foi avaliar a precisão e o desempenho da inferência da direção da fonte de luz e da qualidade visual da sombra gerada.

4.1 Cena interna

Experimento I

Para o primeiro experimento, geramos uma esfera virtual de mesma posição e dimensões da esfera real observada. Utilizamos uma única fonte de luz real, do tipo spot. A sobreposição das esferas resultou em uma imagem usada como *ground-truth*. A sombra da esfera virtual está sintetizada na cor branca para efeitos de comparação, conforme vemos na Fig. 4.1. Tanto a sombra quanto a componente especular sintetizadas podem ser observadas para comparação com as informações do mundo real.

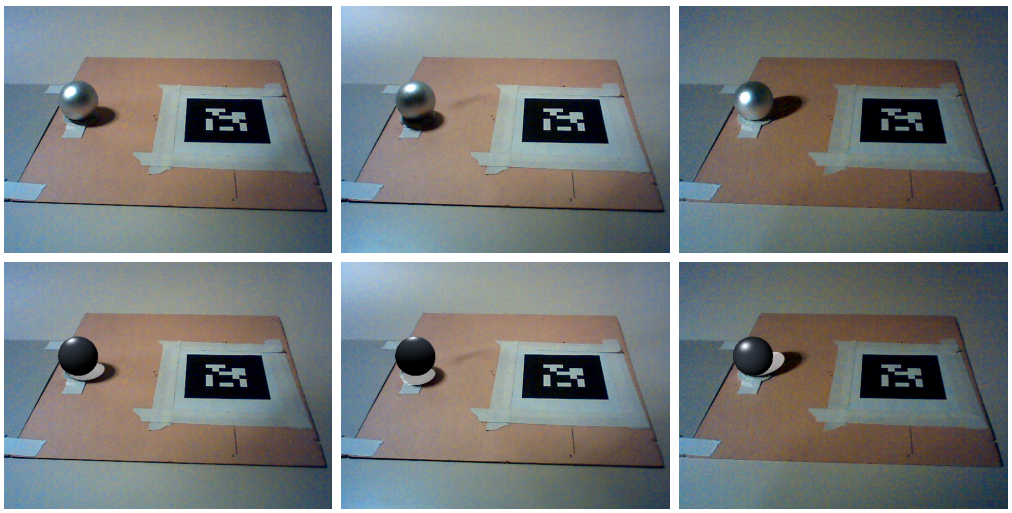

Figura 4.1: *Ground-truth*. A primeira linha representa a imagem capturada, e a segunda linha é o *ground-truth* obtido. Nesse experimento, utilizamos a biblioteca ARToolKitPlus (ARToolKitPlus, 2012).

É importante ressaltar que se fosse usada uma luz pontual virtual, a distância da fonte teria que ser também calculada ou estimada. Optando pela escolha mais simples e estável, a imagem foi sintetizada com uma luz direcional. Como o experimento foi feito controlando uma luz spot, é inevitável que haja um pequeno desvio entre as sombras. Contudo, como se menciona na Seção 1.4, pequenos desvios na direção da fonte de luz são aceitáveis e não prejudicam a imersão em um ambiente de Realidade Aumentada

(Sugano *et al.*, 2003). Consideramos a estimação da fonte de luz visualmente bastante satisfatória para uma aplicação móvel em tempo real.

Experimento II

Modelos virtuais foram inseridos na cena, como ilustra a Fig. 4.2. O marcador foi ocultado sinteticamente para facilitar a visualização das sombras. Para o sombreamento dos modelos, utilizou-se o *variance shadow mapping* (Donnelly & Lauritzen, 2006), descrito na Seção 3.2. É interessante notar também a corretude do *self-shadowing*, mais evidente no segundo modelo.

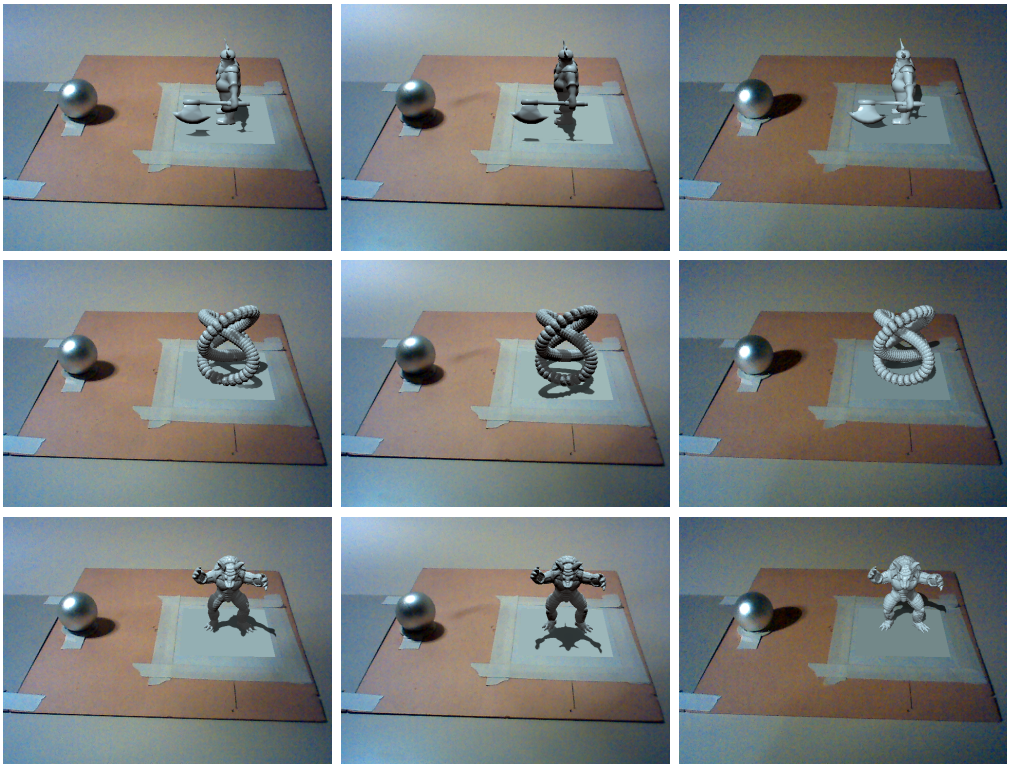

Figura 4.2: Inserção de modelos virtuais utilizamos a esfera difusa como *light probe*. O sombreamento utilizado é o *variance shadow mapping*. Os modelos foram obtidos de Stanford 3D Scanning Repository (Stanford 3D Scanning Repository, 2012) e Open Asset Import Library (Open Asset Import Library, 2012).

4.2 Cena externa

Através de um scanner 3D, capturamos o modelo Cristo, da Fig. 4.3, e no experimento foram colocados lado a lado objetos virtual e real, para que objetos e sombras pudessem ser comparados.

A sombra do objeto virtual foi calculada de forma automática, como descrito na Seção 2.2. Optamos por alinhar o marcador com o norte antes de executar o aplicativo de Realidade Aumentada. Esse alinhamento é feito através da bússola presente no dispositivo móvel. Vale ressaltar que esse alinhamento prévio é feito por praticidade, mas é plenamente possível fazer essa calibração durante a execução do programa, de forma que a orientação do marcador em nada influencie a direção da sombra. A Fig. 4.4 mostra o resultado do experimento utilizando o iPad 2, que possui uma tela maior que o Nokia N900. Consideramos os resultados muito bons, principalmente para um aplicativo móvel executado em tempo real

4.3 Síntese da sombra

Realizamos um experimento utilizando o *variance shadow mapping* com *bit packing*. Variamos a resolução do *shadow map* para avaliar o desempenho e a qualidade visual dos resultados. A posição da fonte de luz foi sintetizada de modo a simular variadas posições do sol durante o decorrer do dia, e a Fig. 4.5 mostra alguns resultados obtidos. A Tabela 4.1 exibe o desempenho (*frame rate*) obtido para o *variance shadow mapping* tanto no iPad quanto no Desktop, habilitando ou não o *bit packing*.

Resolução (pixels)	FPS iPad (sem *bit packing*)	FPS iPad (com *bit packing*)	FPS Desktop
512× 512	31.88	22.59	53.40
1024× 1024	19.50	12.01	46.02
2048× 2048	06.37	03.65	36.94

Tabela 4.1: Desempenho em função da resolução

Em seguida, a Fig. 4.6 mostra uma comparação lado a lado dos métodos de síntese de sombra implementados.

A Fig. 4.7 mostra alguns *screenshots* da aplicação desenvolvida para o Nokia N900, realizando somente o *shadow mapping* simples com *self-shadowing*.

Figura 4.3: Configuração da cena externa: objeto real e marcador da biblioteca Vuforia AR (Vuforia AR SDK, 2012). Esse marcador não é um fiducial tradicional: é uma imagem que contém muitas altas frequências, pois a biblioteca faz o *tracking* de Realidade Aumentada analisando os *features* da imagem em tempo real.

Figura 4.4: Comparando, de vários ângulos, modelo virtual e real numa cena externa com o iPad 2. A suavidade da sombra depende do *blur* aplicado. Nesse experimento, aplicamos um filtro gaussiano de tamanho 7×7 e utilizamos um *shadow map* de 1024×1024 pixels.

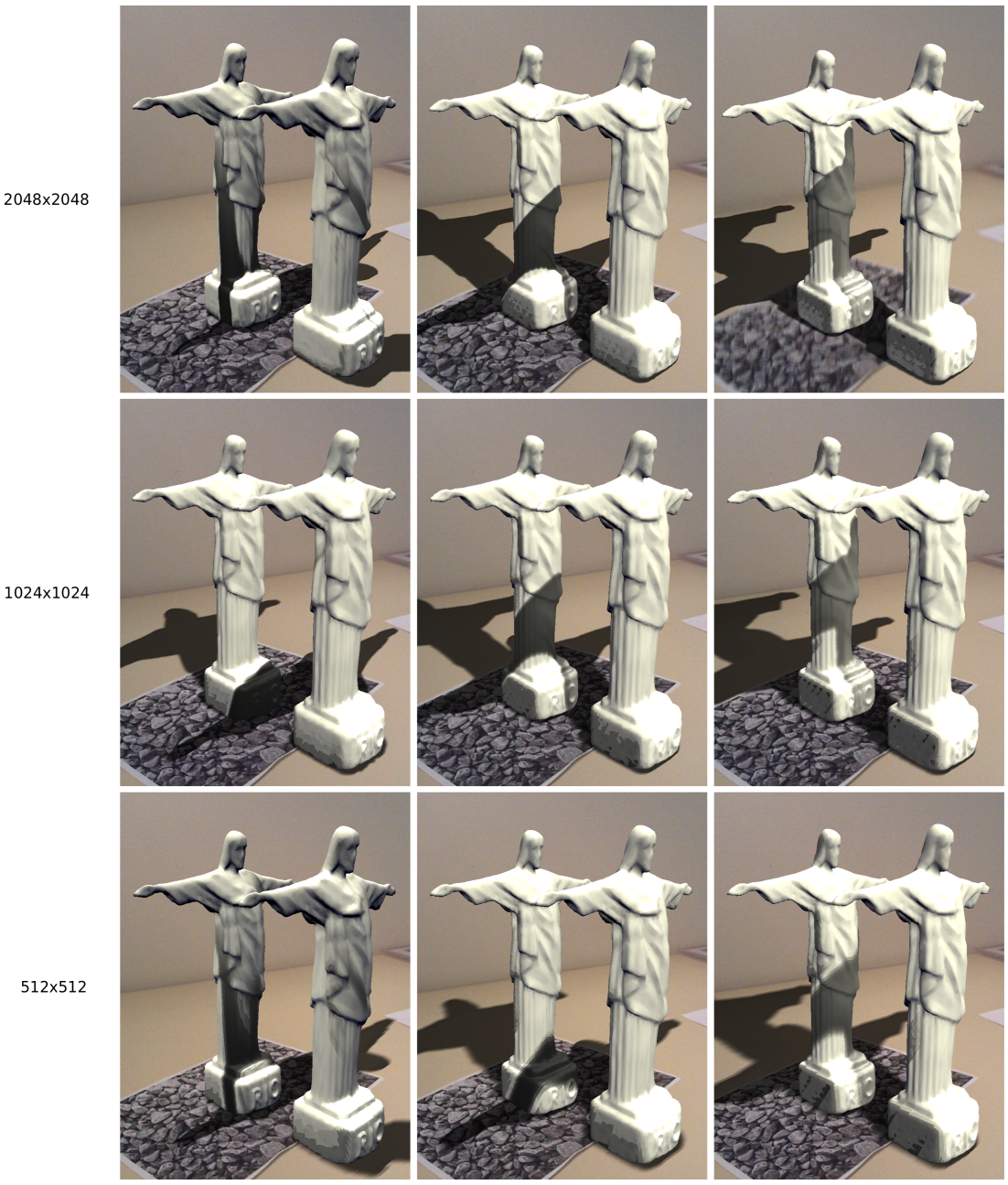

Figura 4.5: Variando a resolução do *Shadow Map*: 2048×2048 (resolução máxima suportada pelos dispositivos móveis), 1024×1024 e 512×512. Os dois objetos são virtuais e estão sintetizados sobre uma mesa real. Observamos que a sombra gerada com resolução máxima possui aparência bastante suave, com pouco aliasing e, à medida que diminuímos a resolução do *Shadow Map*, a aparência da sombra se torna menos suave e o *frame rate* aumenta consideravelmente.

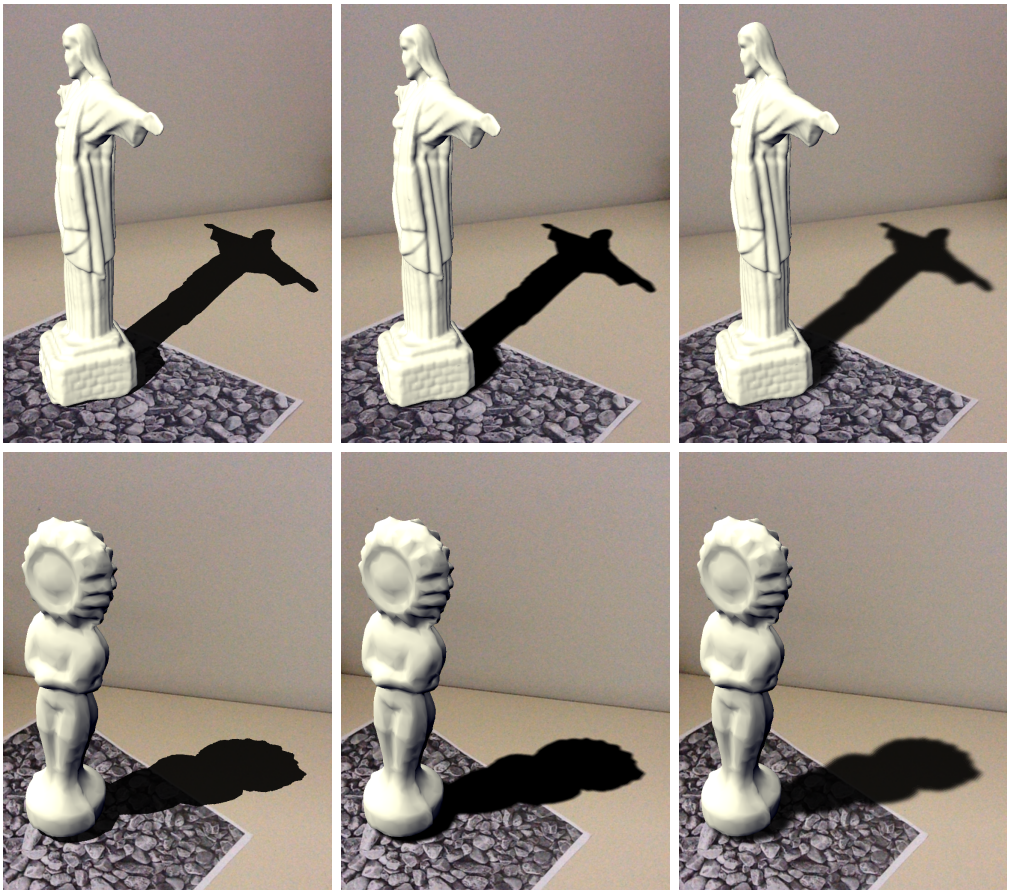

Figura 4.6: Comparando os três métodos de síntese de sombras descritos: *shadow mapping*, *percentage closer filtering* e *variance shadow mapping*. Nessa comparação, utilizamos no *percentage closer filtering* um blur de janela 8×8 para mostrar sua eficácia na suavização da sombra e, como esperado, o desempenho foi comprometido. Os demais métodos, *shadow mapping* e *variance shadow mapping* foram executados em tempo real.

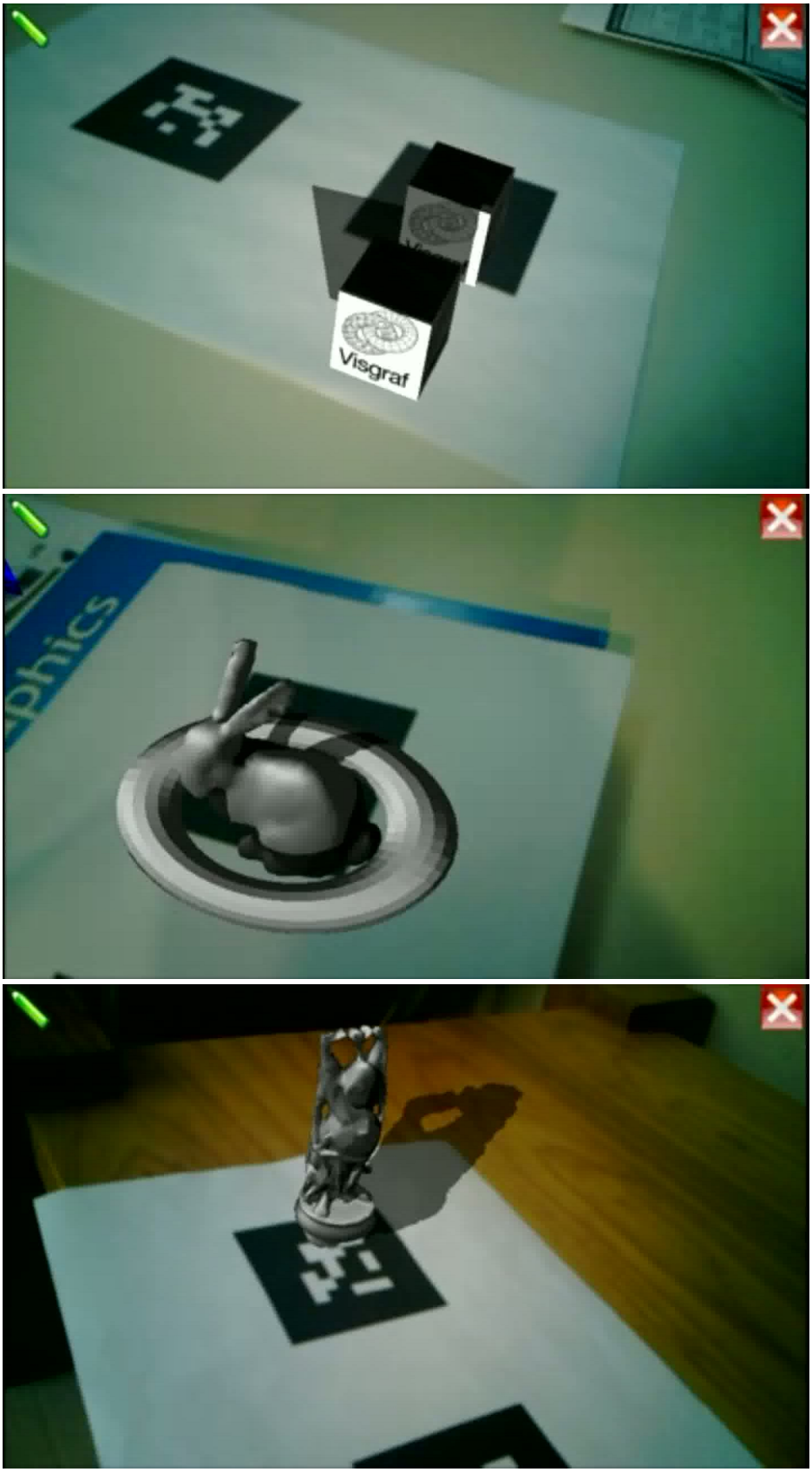

Figura 4.7: *Shadow mapping* simples com *self-shadowing* geradas no Nokia N900. A tela do aparelho possui resolução de 800×480 pixels.

Capítulo 5

Conclusão e Trabalhos Futuros

Apresentamos neste trabalho um sistema de Realidade Aumentada Móvel capaz de estimar rapidamente a direção das fontes de luz da cena e, além disso, sintetizar sombras coerentes com o ambiente e de aparência suave. O método gerou resultados satisfatórios e em tempo real em todas as plataformas utilizadas. O método é aplicável tanto a cenas externas quanto a cenas internas, sendo que nas cenas internas precisamos amostrar as informações de luz através de uma esfera.

Testamos diferentes maneiras de geração de sombras e, comparando o equilíbrio entre eficácia visual e desempenho dos métodos, constatamos que o método *variance shadow mapping* é bastante vantajoso comparado ao seus predecessores *shadow mapping* e *percentage closer filtering*.

Para as cenas externas, calculamos automaticamente a posição do sol relativa ao observador a partir das informações fornecidas pelos sensores do dispositivo móvel. Assumimos que nosso método será utilizado durante o dia e que o tempo está claro. Um possível trabalho futuro é estimar a transparência da sombra e intensidade da luz do sol incidente acessando, através da internet, informações meteorológicas correntes correspondentes às coordenadas geográficas em questão.

Os dispositivos móveis estão cada vez mais acessíveis e têm evoluído significativa-

mente em termos de sensores e poder de processamento: acreditamos que essas características contribuirão para o surgimento de experiências de Realidade Aumentada cada vez mais mais realistas, imersivas e interativas.

Agradecimentos

Esse trabalho foi desenvolvido no Laboratório Visgraf no IMPA, que é patrocinado pelo CNPq, FAPERJ, FINEP, e IBM Brasil.

Referências Bibliográficas

ARToolKitPlus. 2012. https://launchpad.net/artoolkitplus.

Azuma, Ronald T. 1997. A survey of augmented reality. *Presence: Teleoperators and virtual environments*, **6**(4), 355–385.

Blanco-Muriel, Manuel, Alarcón-Padilla, Diego C., López-Moratalla, Teodoro, & Lara-Coira, MartÍn. 2001. Computing the solar vector. *Solar energy*, **70**(5), 431–441.

Brabec, Stefan, Annen, Thomas, & Seidel, Hans-Peter. 2000. Practical shadow mapping. *Journal of graphics tools*, **7**, 9–18.

Castro, T. K., Chapiro, A., Cicconet, M., & Velho, L. 2011. Towards Mobile HDR Video. *Pages 75–76 of:* Day, A., Mantiuk, R., Reinhard, E., & Scopigno, R. (eds), *Eg 2011 areas papers*. Llandudno, UK: Eurographics Association.

Crow, Franklin C. 1977. Shadow algorithms for computer graphics. *Siggraph comput. graph.*, **11**(July), 242–248.

Donnelly, William, & Lauritzen, Andrew. 2006. Variance shadow maps. *Pages 161–165 of: Proceedings of the 2006 symposium on interactive 3d graphics and games*.

Eisemann, Elmar, Schwarz, Michael, Assarsson, Ulf, & Wimmer, Michael. 2011. *Real-time shadows*. AK Peters (CRC Press).

Fernando, Randima. 2004. *Gpu gems: Programming techniques, tips and tricks for real-time graphics*. Pearson Higher Education.

Goesele, M. 2004. *New acquisition techniques for real objects and light sources in computer graphics*. Books on Demand GmbH.

Gooch, Bruce, Sloan, Peter-Pike J., Gooch, Amy, Shirley, Peter, & Riesenfeld, Richard. 1999. Interactive technical illustration. *Pages 31–38 of: Proceedings of the 1999 symposium on interactive 3d graphics*. I3D '99. New York, NY, USA: ACM.

Haller, Michael, Drab, Stephan, & Hartmann, Werner. 2003. A real-time shadow approach for an augmented reality application using shadow volumes. *Pages 56–65 of: Proceedings of the acm symposium on virtual reality software and technology*.

Jacobs, Katrien, & Loscos, Céline. 2006. Classification of illumination methods for mixed reality. *Comput. graph. forum*, **25**(1), 29–51.

Kanbara, Masayuki, & Yokoya, Naokazu. 2002. Geometric and photometric registration for real-time augmented reality. *Pages 279–280 of: Proceedings of the 1st international symposium on mixed and augmented reality*.

Kersten, D., Knill, D., Mamassian, P., & Bülthoff, I. 1996. Illusory motion from shadows. *Nature*, 31.

Open Asset Import Library. 2012. http://assimp.sourceforge.net.

Pessoa, Saulo, Moura, Guilherme, Lima, Joao, Teichrieb, Veronica, & Kelner, Judith. 2010. Photorealistic rendering for Augmented Reality: A global illumination and BRDF solution. *Pages 3–10 of: 2010 ieee virtual reality conference (vr)*. IEEE.

Reeves, William T., Salesin, David H., & Cook, Robert L. 1987. Rendering antialiased shadows with depth maps. *Siggraph comput. graph.*, **21**(August), 283–291.

Rost, Randi J., Licea-Kane, Bill, Ginsburg, Dan, Kessenich, John M., Lichtenbelt, Barthold, Malan, Hugh, & Weiblen, Mike. 2009. *Opengl shading language*. 3rd edn. Addison-Wesley Professional.

Stanford 3D Scanning Repository. 2012. http://graphics.stanford.edu/data/3Dscanrep.

Sugano, N., Kato, H., & Tachibana, K. 2003. The effects of shadow representation of virtual objects in augmented reality. *Pages 76–83 of: Proceedings of the 2nd ieee/acm international symposium on mixed and augmented reality.*

Supan, Peter, Stuppacher, Ines, & Haller, Michael. 2006. Image based shadowing in real-time augmented reality. *International journal of virtual reality*, **5**(3), 1–7.

Thompson, Robert Bruce, & Thompson, Barbara Fritchman. 2005. *Astronomy hacks*. First edn. O'Reilly.

Vuforia AR SDK. 2012. https://ar.qualcomm.at/qdevnet/sdk/ios.

Williams, Lance. 1978. Casting curved shadows on curved surfaces. *Siggraph comput. graph.*, **12**(August), 270–274.

Apêndice A

Cálculo da Posição do Sol

O código de (Blanco-Muriel *et al.*, 2001), que utilizamos para o cálculo da posição do sol, é listado a seguir:

sunpos.h

```
1  // This file is available in electronic form at http://www.psa.es/sdg/
       sunpos.htm
2
3  #ifndef __SUNPOS_H
4  #define __SUNPOS_H
5
6
7  // Declaration of some constants
8  #define pi     3.14159265358979323846
9  #define twopi (2*pi)
10 #define rad    (pi/180)
11 #define dEarthMeanRadius      6371.01     // In km
12 #define dAstronomicalUnit     149597890   // In km
13
14 struct cTime
15 {
16         int iYear;
```

```
17          int iMonth;
18          int iDay;
19          double dHours;
20          double dMinutes;
21          double dSeconds;
22  };
23
24  struct cLocation
25  {
26          double dLongitude;
27          double dLatitude;
28  };
29
30  struct cSunCoordinates
31  {
32          double dZenithAngle;
33          double dAzimuth;
34  };
35
36  void sunpos(cTime udtTime, cLocation udtLocation, cSunCoordinates *
        udtSunCoordinates);
37
38  #endif
```

sunpos.cpp

```
1  // This file is available in electronic form at http://www.psa.es/sdg/
       sunpos.htm
2
3  #include "sunpos.h"
4  #include <math.h>
5
6  void sunpos(cTime udtTime,cLocation udtLocation, cSunCoordinates *
       udtSunCoordinates)
7  {
8          // Main variables
9          double dElapsedJulianDays;
10         double dDecimalHours;
11         double dEclipticLongitude;
12         double dEclipticObliquity;
13         double dRightAscension;
14         double dDeclination;
15
16         // Auxiliary variables
17         double dY;
18         double dX;
19
20         // Calculate difference in days between the current Julian Day
21         // and JD 2451545.0, which is noon 1 January 2000 Universal Time
22         {
23                 double dJulianDate;
24                 long int liAux1;
25                 long int liAux2;
26                 // Calculate time of the day in UT decimal hours
27                 dDecimalHours = udtTime.dHours + (udtTime.dMinutes
28                         + udtTime.dSeconds / 60.0 ) / 60.0;
29                 // Calculate current Julian Day
30                 liAux1 =(udtTime.iMonth-14)/12;
```

```
31                  liAux2=(1461*(udtTime.iYear + 4800 + liAux1))/4 + (367*(
                        udtTime.iMonth
32                          - 2-12*liAux1))/12- (3*((udtTime.iYear + 4900
33                  + liAux1)/100))/4+udtTime.iDay-32075;
34                  dJulianDate=(double)(liAux2)-0.5+dDecimalHours/24.0;
35                  // Calculate difference between current Julian Day and JD
                        2451545.0
36                  dElapsedJulianDays = dJulianDate-2451545.0;
37          }
38
39          // Calculate ecliptic coordinates (ecliptic longitude and
                obliquity of the
40          // ecliptic in radians but without limiting the angle to be less
                than 2*Pi
41          // (i.e., the result may be greater than 2*Pi)
42          {
43                  double dMeanLongitude;
44                  double dMeanAnomaly;
45                  double dOmega;
46                  dOmega=2.1429-0.0010394594*dElapsedJulianDays;
47                  dMeanLongitude = 4.8950630+ 0.017202791698*
                        dElapsedJulianDays; // Radians
48                  dMeanAnomaly = 6.2400600+ 0.0172019699*dElapsedJulianDays;
49                  dEclipticLongitude = dMeanLongitude + 0.03341607*sin(
                        dMeanAnomaly )
50                          + 0.00034894*sin( 2*dMeanAnomaly )-0.0001134
51                          -0.0000203*sin(dOmega);
52                  dEclipticObliquity = 0.4090928 - 6.2140e-9*
                        dElapsedJulianDays
53                          +0.0000396*cos(dOmega);
54          }
55
```

```
56          // Calculate celestial coordinates ( right ascension and
                declination ) in radians
57          // but without limiting the angle to be less than 2*Pi (i.e., the
                result may be
58          // greater than 2*Pi)
59          {
60                  double dSin_EclipticLongitude;
61                  dSin_EclipticLongitude= sin( dEclipticLongitude );
62                  dY = cos( dEclipticObliquity ) * dSin_EclipticLongitude;
63                  dX = cos( dEclipticLongitude );
64                  dRightAscension = atan2( dY,dX );
65                  if( dRightAscension < 0.0 ) dRightAscension =
                        dRightAscension + twopi;
66                  dDeclination = asin( sin( dEclipticObliquity )*
                        dSin_EclipticLongitude );
67          }
68
69          // Calculate local coordinates ( azimuth and zenith angle ) in
                degrees
70          {
71                  double dGreenwichMeanSiderealTime;
72                  double dLocalMeanSiderealTime;
73                  double dLatitudeInRadians;
74                  double dHourAngle;
75                  double dCos_Latitude;
76                  double dSin_Latitude;
77                  double dCos_HourAngle;
78                  double dParallax;
79                  dGreenwichMeanSiderealTime = 6.6974243242 +
80                          0.0657098283*dElapsedJulianDays
81                          + dDecimalHours;
82                  dLocalMeanSiderealTime = (dGreenwichMeanSiderealTime*15
83                          + udtLocation.dLongitude)*rad;
```

```
84                  dHourAngle = dLocalMeanSiderealTime - dRightAscension;
85                  dLatitudeInRadians = udtLocation.dLatitude*rad;
86                  dCos_Latitude = cos( dLatitudeInRadians );
87                  dSin_Latitude = sin( dLatitudeInRadians );
88                  dCos_HourAngle= cos( dHourAngle );
89                  udtSunCoordinates->dZenithAngle = (acos( dCos_Latitude*
                        dCos_HourAngle
90                          *cos(dDeclination) + sin( dDeclination )*
                            dSin_Latitude));
91                  dY = -sin( dHourAngle );
92                  dX = tan( dDeclination )*dCos_Latitude - dSin_Latitude*
                        dCos_HourAngle;
93                  udtSunCoordinates->dAzimuth = atan2( dY, dX );
94                  if ( udtSunCoordinates->dAzimuth < 0.0 )
95                          udtSunCoordinates->dAzimuth = udtSunCoordinates->
                            dAzimuth + twopi;
96                  udtSunCoordinates->dAzimuth = udtSunCoordinates->dAzimuth/
                        rad;
97                  // Parallax Correction
98                  dParallax=(dEarthMeanRadius/dAstronomicalUnit)
99                          *sin(udtSunCoordinates->dZenithAngle);
100                 udtSunCoordinates->dZenithAngle=(udtSunCoordinates->
                        dZenithAngle
101                         + dParallax)/rad;
102         }
103 }
```

Apêndice B

Bit Packing

Algumas funções para o que utilizamos para o *bit packing* são listadas a seguir:

Packing

```
1  vec4 packFloatTo4x8(in float val) {
2    const vec4 bitSh = vec4(256.0f*256.0f*256.0f, 256.0f*256.0f, 256.0f, 1.0
         f);
3    const vec4 bitMsk = vec4(0.0f, 1.0f/256.0f, 1.0f/256.0f, 1.0f/256.0f);
4    vec4 result = fract(val * bitSh);
5    result -= result.xxyz * bitMsk;
6    return result;
7  }
8
9  vec4 pack2FloatTo4x8(in vec2 val) {
10   const vec2 bitSh = vec2(256.0f, 1.0f);
11   const vec2 bitMsk = vec2(0.0f, 1.0f/256.0f);
12   vec2 res1 = fract(val.x * bitSh);
13   res1 -= res1.xx * bitMsk;
14   vec2 res2 = fract(val.y * bitSh);
15   res2 -= res2.xx * bitMsk;
16   return vec4(res1.x,res1.y,res2.x,res2.y);
17 }
```

Unpacking

```
1  float unpack4x8ToFloat(in vec4 val) {
2    const vec4 unshift = vec4(1.0f/(256.0f*256.0f*256.0f), 1.0f/(256.0f
         *256.0f), 1.0f/256.0f, 1.0f);
3    return dot(val, unshift);
4  }
5
6  vec2 unpack4x8To2Float(in vec4 val) {
7    const vec2 unshift = vec2(1.0f/256.0f, 1.0f);
8    return vec2(dot(val.xy, unshift), dot(val.zw, unshift));
9  }
```