Universidade do Minho
Escola de Engenharia

César Carlos Martins Gomes

Ray Tracing Interactivo Híbrido: GPU + CPU

Outubro de 2010

Universidade do Minho
Escola de Engenharia

César Carlos Martins Gomes

Ray Tracing Interactivo Híbrido: GPU + CPU

Tese Mestrado
Mestrado em Informática

Trabalho efectuado sob a orientação do
Professor Doutor Luís Paulo Peixoto dos Santos

Outubro de 2010

Agradecimentos

Quero agradecer: Ao meu orientador, professor Luís Paulo Santos, pelo incentivo, apoio, orientação, disponibilidade e contribuições.
À minha família pelo incansável apoio que me deram, em especial aos meus pais e à minha amada esposa Liliana.
Esta tese foi financiada pela Fundação para a Ciência e Tecnologia através dos projectos "IGIDE - Iluminação Global Interactiva em Ambientes Dinâmicos" - Refª PTDC/EIA/65965/2006, e "PERFORM - Portabilidade e Desempenho em Sistemas Paralelos Heterogéneos" - Ref. PTDC/EIA-EIA/100035/2008.

Resumo

Ray Tracing Interactivo Híbrido: GPU + CPU

O *ray tracing* é um algoritmo de visibilidade utilizado em diversas abordagens de síntese de imagens de alta qualidade. Porém, possuiu um custo computacional elevado comparado com as técnicas aplicadas nas placas gráficas modernas, tais como a projecção.

Esta tese propõe uma solução híbrida entre o *ray tracing* e técnicas aplicadas nas placas gráficas para sintetizar imagens. São propostas três abordagens em particular: i) substituir os raios primários pela projecção na placa gráfica, ii) substituir os raios sombra da iluminação directa por *shadow maps* e iii) num contexto de radiosidade instantânea para a iluminação indirecta difusa substituir os raios sombra por *shadow maps*. Na iluminação directa, apenas com *shadow maps*, é visível o efeito *aliasing* nos contornos das sombras. Este problema é solucionado disparando raios sombra ao longo destes contornos, refinando assim a informação obtida com os *shadow maps* e eliminando o *aliasing*.

As abordagens propostas foram testadas em dois *ray tracers*: o iRT baseado em CPU e o Optix baseado em GPU. Na primeira registaram-se ganhos significativos, enquanto na segunda estes não são tão consensuais. A qualidade da imagem, conforme a métrica PNSR, não foi comprometida.

Abstract

Interactive Hybrid Ray Tracing: GPU + CPU

Ray tracing is a visibility algorithm often used at the core of many rendering methods in order to achieve high quality images. However, ray tracing is computationally very expensive when compared to alternative methods such as the z-buffer supported by current GPUs.

This thesis proposes to combine ray tracing with z-buffering, resulting an hybrid approach to rendering. In particular, three different cases are studied: i) substituting primary rays by perspective projection on the GPU, ii) substituting shadow rays for direct illumination by shadow maps and iii) substituting shadow rays for indirect diffuse illumination in an Instant Radiosity context by shadow maps. For direct illumination, shadow maps result in noticeable aliasing artifacts, thus an oversampling strategy along the shadows edges is proposed, which consists on spawning shadow rays only along these regions; such approach is termed "Shadow Maps & Selective Ray Tracing".

The above described approaches are evaluated on two different base ray tracers: a CPU-based one (iRT) and a GPU-based one (OptiX). Performance gains with the hybrid approaches are significant for the CPU case, but are not so obvious for the OptiX based case. Image quality is not compromised, as demonstrated by the PNSR metric.

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

A iluminação global, em computação gráfica, consiste em iluminar um objecto tendo em consideração todos os outros objectos presentes na cena, permitindo gerar imagens de alta qualidade. Existem várias abordagens, mas todas partem do princípio de simular as interacções dos diversos trajectos de luz desde as fontes de luz até ao observador para gerar imagens de alta qualidade, como se fossem fotos tiradas a uma determinada cena a partir de uma posição: este processo é designado por síntese de imagens (*render*)[1]. Esta simulação acarreta um elevado custo computacional devido às inúmeras interacções que ocorrem entre a fonte de luz e o observador. A maior consequência, com as máquinas actuais, é a inviabilidade de gerar imagens em tempo real permitindo a interacção entre utilizador e computador. Tradicionalmente, nestas abordagens a simulação da propagação do fluxo radiante é feita através da técnica *ray tracing*. E, é aqui que se verifica a maior fatia de tempo despendida a computar.

Por outro lado, as actuais placas gráficas, através do seu *pipeline* gráfico,

[1]Os termos *render* e *rendering* serão utilizados para denotar o processo de síntese de imagem por computador dado a sua utilização generalizada pela comunidade internacional.

oferecem meios para geração de imagens em tempo real. Mas, ao contrário do *ray tracing*, grande parte dos algoritmos usados baseiam-se em métodos empíricos que produzem um resultado semelhante à realidade. O resultado final é uma imagem que não corresponde à realidade e com uma qualidade inferior comparativamente às geradas com *ray tracing*. Certos algoritmos como a projecção em perspectiva ou o *shadow mapping* possuem uma base realística, porém por questões de desempenho e limitações de máquina, parte dos resultados são interpolados introduzindo erros.

Este trabalho consiste em construir uma solução híbrida entre o *ray tracing* e técnicas aplicadas nas placa gráficas modernas. Pretende-se então aplicar a projecção em perspectiva para a determinação da geometria visível, evitando recorrer ao *ray tracing* para a sua determinação. E, aplicar a técnica *shadow mapping* para determinar a visibilidade das fontes de luz, incluindo as fontes de luz virtuais, evitando recorrer ao *ray tracing* para a sua determinação. Espera-se, aplicando esta abordagem, obter um aumento de desempenho e manter a qualidade da imagem.

O presente documento encontra-se dividido em capítulos. Após uma Introdução, surge o capítulo Estado da Arte onde se descreve conceitos e o desenvolvimento da temática até à data. No capítulo Abordagem descreve-se a contribuição deste trabalho. O capítulo Testes apresenta os resultados e compara a qualidade das imagens produzidas e o desempenho da aplicação. Por fim surge a Conclusão onde é recapitulado o documento, é feita uma auto-crítica, e sugestões para trabalho futuro.

Capítulo 2

Estado da Arte

Neste capítulo vamos expor o estado da arte desta temática. É introduzida uma descrição geral da iluminação global, e as técnicas de determinação de visibilidade *ray tracing* e a sua evolução nas placas gráficas, e o OpenGL. Descreve-se por fim a situação da iluminação indirecta, e a utilização do *shadow mapping* para teste de oclusão das fontes de luz em ambientes em que abordam a iluminação global.

2.1 Iluminação Global

A iluminação global, por contraste à iluminação local, consiste em iluminar um objecto tendo em atenção as diversas interacções da luz entre a fonte e todos os objectos presentes na cena. Assim, é possível criar imagens de alta qualidade, a partir de uma cena, através da simulação do trajecto da luz[DBB02]. E, existem várias abordagens tais como *ray tracing* clássico (Whitted), *photon mapping*, *path tracing*, radiosidade, entre outras. Uma das principais diferenças entre as várias abordagens específicas é a aproximação

feita à da equação de *rendering*. A equação de *rendering*, aplicada a um ponto, atende aos fenómenos ópticos, incluindo a luz que incide no ponto, e quando resolvida obtém-se a luz reflectida segundo um ângulo de observação. A simulação do trajecto da luz é, amplamente, feita através do algoritmo *ray tracing*.

2.2 Ray Tracing nas placas gráficas

O *ray tracing*, vulgarizado na computação gráfica em 1980 no artigo "An improved illumination model for shaded display"[Whi80], é um algoritmo que determina a partir de um conjunto de primitivas geométricas e um raio, o primeiro ponto de intersecção, a contar da origem do raio, entre o raio e as primitivas geométricas. Em *ray tracing* cada raio intersectado pode originar novos raios com origem na intersecção, esta é a principal diferença entre o *ray tracing* e o *ray casting*. Apesar de ser amplamente conhecido pelas suas aplicações em Iluminação Global e fenómenos ópticos, o *ray tracing* é também utilizado em outras áreas da física, como detecção de espessura de materiais, colisões ou propagação de ondas.

Desde que o *ray tracing* foi apresentado tem havido investigação para um melhor desempenho do algoritmo. Uma das áreas em que se tem apostado nas últimas décadas é a construção de estruturas de aceleração[ZHWG08]. O objectivo das estruturas de aceleração é permitirem acelerar o processo de determinação dos triângulos intersectados. Para tal impõem uma ordem no espaço 3D para diminuir o número de intersecções raio-geometria. Porém, para cenas não estáticas[1] é necessário construir para cada imagem gerada uma estrutura de aceleração. As abordagens mais comuns são as *kd-trees* ou

[1]Entende-se por cenas não estáticas, cenas em que existem objectos em movimento.

as *bounding volume hierarchy* (BVH) que dividem a cena numa árvore. Com esta abordagem o algoritmo de *ray tracing* é dividido em duas componentes, a travessia e a intersecção. A travessia percorre a estrutura de aceleração, determinando espaços atravessados pelo raio. A geometria contida nestes espaços é intersectada com o raio na componente intersecção.

Até aos dias de hoje para atingir uma taxa de geração de imagens que permitam uma interactividade é necessário recorrer a um *cluster* de computadores. Por outro lado, com o surgimento das modernas placas gráficas que suportam programação para múltiplos propósitos tem surgido novas abordagens que correm o *ray tracing* exclusivamente na placa gráfica, como o Optix da Nvidia[PBD+10]. Os processadores gráficos (GPUs) de hoje são processadores poderosos, com grande capacidade em processamento vectorial através da arquitectura *single instruction, multiple threads* (SIMT)[She09].

Uma das primeiras abordagens na utilização do GPU para *ray tracing* [CHH02] é utilizar os *shaders* das placas gráficas para raios primários. Ou seja, em vez de determinar a geometria visível para cada pixel no CPU, utilizamos o *z-buffer* da placa gráfica para a determinar, e programa-se o *shader* para devolver ao CPU a geometria correspondente a cada pixel. O restante algoritmo mantém-se no CPU. Martin Christen [Chr05] adapta também o algoritmo de ray tracing ao pipeline de *shaders* das placas gráficas. No entanto, resultante da procura de eficiência e restrições computacionais dos *shaders*, restringe o algoritmo eliminando as refracções e a recursividade (raios secundários).

Uma das mais recentes abordagens são as plataformas heterogéneas. Hoje um computador pessoal já vem equipado com um processador *multi-core* e uma placa gráfica com vários *cores*. Num nível mais elevado podemos ter *clusters* com várias máquinas, cada uma com vários processadores, em que estes possuem vários *cores*, e vários GPUs com vários *cores*. Existe a ne-

cessidade de tirar proveito de todo este poder computacional sem ter que desenvolver código específico para cada plataforma. E é neste princípio que surge o OpenCL, uma plataforma de computação paralela que, com apenas um *kernel*, adapta-o aos vários recursos da máquina. Aparentemente até à data ainda não foi publicado um *ray tracing* que utiliza esta ferramenta. No entanto já surgiu uma abordagem que parte do mesmo principio. Em "Out-of-core Data Management for Path Tracing on Hybrid Resources"[BBS+09] propôs uma arquitectura para um *path tracing* dividida em dois níveis. No nível mais alto é "implementado o algoritmo básico de *render*". E, no nível mais baixo é implementado a gestão dos recursos (GPUs, CPUs) e base de dados. Neste nível é feita a distribuição de *kernels*[2] para cada recurso conforme o potencial deste. Ou seja, se um dado *kernel* tiver melhor performance no GPU, então, irá ser direccionado para o GPU. No caso do GPU é necessário preparar e organizar os dados a enviar para a placa gráfica, dado que nesta solução toda informação está no sistema principal. A aplicação foi testada num *cluster* em que cada nodo tem diferentes configurações dos recursos.

2.3 OpenGL

O OpenGL, *Open Graphics Library*, é uma interface de programação (API) de placas gráficas [Ope10], que permite desenhar primitivas geométricas no monitor.

O OpenGL utiliza a projecção perspectiva[Ope07] para projectar geometria tridimensional num plano, imagem final a apresentar no monitor, através de

[2]Uma kernel representa uma funcionalidade elementar, por exemplo intersectar triângulos.

uma matriz de transformação. Mas, um dos efeitos ao projectar um ponto no plano é a perda de profundidade, componente z, o que faz com que os últimos objectos a desenhar fiquem à frente, o que não é sempre verdade. Para garantir uma projecção correcta as placas gráficas utilizam o *z-buffer*[Ope07]. O *z-buffer* é uma matriz (que representa uma porção de plano, com as mesmas dimensões da imagem final) onde para cada *pixel* correspondente da imagem final é guardado a profundidade perspectiva. O *z-buffer* é iniciado com todas células com a maior profundidade possível. Para cada projecção, é comparada a profundidade perspectiva com o valor da posição correspondente guardado no *z-buffer*, caso seja menor, tanto no *z-buffer* como na imagem final são substituídos pela projecção mais recente.

O processo de desenhar da placa gráfica é constituído por vários componentes, entre eles o *vertex shader*, *geometry shader* e o *fragment shader* que podem ser programáveis[RLKG+09]. O *vertex shader* computa os vértices isoladamente[3], isto permite fazer transformações aos vértices e à informação associada ao vértice. Normalmente é aqui que se projecta o vértice. Depois de processar os vértices no *vertex shader* o *geometry shader* computa a primitiva geométrica. O *geometry shader* é uma componente mais recente introduzida no DirectX 10 da Microsoft, o seu objectivo é a geração em tempo real de novas primitivas geométricas[MSD10]. Por fim, o *fragment shader* processa potenciais *pixeis* resultantes da projecção duma primitiva geométrica na imagem.

[3]No *vertex shader* ao computar um vértice, este, não é provido de informação em relação aos vértices adjacentes do seu triângulo.

2.4 Iluminação indirecta com luzes virtuais

A iluminação directa é a luz que provém das fontes de luz e incide directamente, sem interagir com a restante cena, num ponto. Por oposição, a iluminação indirecta é toda a luz proveniente das fontes de luz que incide num ponto mas que interage previamente com a cena. Em Iluminação Global a contribuição da iluminação indirecta é um aspecto importante. Uma das primeiras abordagens para calcular a contribuição difusa para um dado ponto é disparar raios com direcções aleatórias, interagindo com a cena, até atingir uma fonte de luz. Se forem disparados alguns raios obtemos uma fraca amostra, muitos raios tornamos o processo lento.E, dada a sua aleatoriedade pode nunca atingir uma fonte de luz, principalmente em cenas que possuam espaços semi-fechados sem uma fonte de luz que o ilumine directamente.
Alexander Keller[Kel97], em 1997, propõe no artigo "Instant Radiosity" caches de radiosidade como um método para acelerar o processo. Previamente disparam-se aleatoriamente raios a partir das fontes de luz. Estes raios ao intersectar com uma superfície criam uma cache de radiosidade. A radiosidade desta cache é calculada em função da fonte de luz precedente e a luz reflectida pela superfície. Esta cache no processo de *render* irá-se comportar como uma fonte de luz hemi-direccional, mas não será visível pois ela não existe, daí a designação luz virtual. O processo repete-se, disparando novos raios a partir da nova luz virtual, até chegar a um limite pré-definido. Assim, para determinar a contribuição difusa, em vez de disparar raios interactivamente até atingir uma fonte de luz, disparam-se raios a testar a visibilidade das luzes virtuais, e as visíveis contribuem para a iluminação difusa indirecta.
Esta abordagem pode tornar o processo de *render* mais rápido. Mas, para obter uma boa qualidade de imagem é necessário criar um grande número de luzes virtuais, o que leva a que seja necessário disparar um grande número

de raios. E disparar raios tem um custo computacional acrescido.

2.5 Shadow Mapping

O Shadow Mapping, introduzido por Lance Williams no artigo *"Casting curved shadows on curved surfaces"*[Wil78] em 1978, é uma técnica de detecção de oclusão de luz: sombras. O algoritmo é dividido em duas partes, a geração do *shadow map* e o teste de oclusão, e para cada fonte de luz é necessário um *shadow map*. A geração do *shadow map* consiste em colocar a câmera na fonte de luz e projectar a cena guardando o *z-buffer*, o *shadow map*, gerado e a matriz desta projecção. O teste de oclusão consiste em para cada ponto do mundo correspondente a um *pixel* da imagem projectar no plano *shadow map* através da matriz de projecção (figura 2.1). Se o valor da cota, z, for superior ao do *shadow map* então está ocludido, caso contrário não está ocludido. Como é omitida a importância das inter-reflexões da luz devido a questões de desempenho, uma oclusão corresponde a não estar iluminado, sombra, caso contrário está iluminado.

O *shadow mapping* é amplamente usado num mundo da computação gráfica, desde a indústria dos jogos, *Computer-Aided Design* (CAD), animação 3D, entre outras aplicações[SWP10]. É também o algoritmo base de muitos algoritmos de sombras existentes[SWP10].

Neste documento vamos analisar o trabalho de *shadow mapping* aplicado à iluminação global.

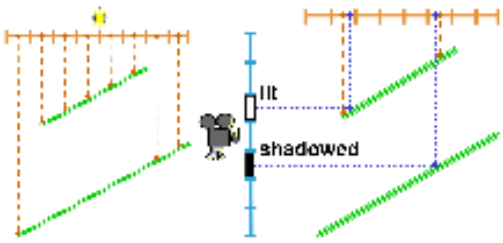

Figura 2.1 – Exemplo do *shadow mapping*. Primeiro é construído o mapa de profundidades, em cima a amarelo. No processo de *render* os pontos visíveis pela câmera são projectados no mapa de profundidade, e comparadas as distâncias. Figura retirada do artigo [SWP10].

2.5.1 Aliasing

O maior problema do *shadow mapping* é o efeito *aliasing*[SWP10], efeito escada, que surge nos contornos das sombras na geometria mais distante da fonte de luz. Este efeito deve-se à resolução finita do mapa de profundidade (figura 2.2). Ou seja, quanto mais longe a geometria a ser observada estiver da fonte de luz e da geometria que a oclui, a sombra que esta projecta sofre um aumento, aumento este que leva a que surja este efeito devido à resolução finita do mapa de profundidade.

Para eliminar o *aliasing* Marc Stamminger e George Drettakis, no artigo *Perspective shadow maps*[SD02], apresentam uma técnica que reduz consideravelmente, ou elimina por completo. A grande diferença em relação ao algoritmo original é quando é criado o mapa de profundidade. Em vez de ser criado previamente ao processo de *render*, é criado após a projecção e com o espaço do *frustum* da câmera. A posição da fonte de luz é também "projectada". Isto implica que o mapa de profundidade seja também criado em função da perspectiva da câmera, colocando-o muito perto da câmera, ponto

Figura 2.2 – Exemplo do efeito *aliasing*. Figura retirada do artigo [SWP10].

de visão. Então, todas as sombras perto da câmera estarão bem definidas e à medida que afasta-se da câmera torna-se menos definida, o que não é problemático porque não é perceptível[SD02]. Porém, cria um novo problema, se existirem objectos entre a fonte de luz e o espaço *frustum* da câmera, que ocultem a fonte de luz criando sombra, esta sombra não é projectada.

Outra abordagem é recorrer ao *ray tracing* para redefinir os contornos das sombras[MB05]. Para tal, após o teste de oclusão, é testado se o ponto em questão está no contorno da sombra. Caso o ponto não esteja no contorno procede-se com o algoritmo padrão do *shadow mapping*, caso contrário, é disparado um raio. Este raio, denominado de raio sombra, caso intersecte geometria entre a fonte de luz e o ponto então o ponto não é iluminado, senão é iluminado pela fonte de luz. Esta abordagem elimina na maioria dos casos o efeito *aliasing*. Em situações onde se verifique uma sub-amostragem[4] esta

[4]A sub-amostragem ocorre quando o mapa de profundidade possuiu uma resolução muito abaixo que a resolução da imagem resultante, ou, os objectos visíveis estão demasiado longe da fonte de luz.

solução não soluciona o problema.

Figura 2.3 – Detecção de contornos da sombra do coelho. Figura retirada do artigo [Chr09].

Em 2009 no artigo "Fast Hard and Soft Shadow Generation on Complex Models using Selective Ray Tracing"[Chr09] propõe uma abordagem similar à anterior. A diferença verifica-se na análise de contornos, enquanto a abordagem anterior analisava os contornos das sombras da imagem final, aqui é analisada com base na diferença de profundidades. Ou seja, ao projectar um determinado ponto P no mapa de profundidades, correspondendo ao elemento E, temos a distância dP de P à fonte de luz L e a distância dE previamente guardada no elemento E. Então, é determinado qual o elemento vE, vizinho de E com maior diferença Δ, $\Delta = max(||dE - i||, i \in$ vizinhos de $E)$. Determina-se a distância dA, tal que, $dA = ||\overrightarrow{LA}|| \wedge \overrightarrow{PA} \cdot \overrightarrow{Np} = 0$, com $\overrightarrow{PA}$ vector com origem na fonte de luz L e direcção o elemento vE do mapa de profundidade, e Np normal da superfície do ponto P. Se $dA \leq dE + \Delta$ então temos um contorno, logo é disparado um raio sombra.

2.5.2 Fontes de luzes omnidirecionais e hemi-direcionais

Uma fonte de luz omnidirecional emite luz em todas as direcções, e uma fonte de luz hemi-direcional emite luz apenas numa semi-esfera centrada numa dada direcção. No entanto, com a projecção standard das placas gráficas

não é possível criar um *view frustum* que abarque uma visão omnidirecional ou hemi-direccional para criar um único *shadow map*. Para ultrapassar este problema existem algumas abordagens. Para as fontes de luz omnidirecionais pode ser criado um cubo centrado na fonte de luz, e cada face representa um mapa de profundidade[SD01]. Outra abordagem é a criação de duas projecções parabólicas.

A projecção parabólica foi proposta em 1998 no artigo "View-independent environment maps" [HS98] para a criação de imagens que abarcassem uma visão 360º em apenas duas passagens de *rendering*. Esta projecção consiste numa nova parametrização (equação 2.1) para projectar os triângulos e respectivos vértices.

$$f(x,y) = \frac{1}{2} - \frac{1}{2}(x^2 + y^2), x^2 + y^2 \leq 1 \tag{2.1}$$

Com esta projecção é possível projectar campos de visão de 180º, na horizontal e vertical. Para uma visão de 360º basta criar duas imagens em dois passos de *rendering*. Posiciona-se a câmera num determinado ponto, e sintetiza a primeira imagem com uma determinada direcção, e, a segunda com a direcção simétrica da primeira. Porém esta abordagem pode introduzir erros na imagem final devido à interpolação. Estes erros devem-se ao *pipeline* da placa gráfica que após ter os vértices de um triângulo projectado, para desenhar o triângulo, interpola linearmente o conteúdo deste, e não em função de uma projecção parabólica. Esta interpolação é fixa e não é possível alterar, e para que o resultado seja maioritariamente correcto é necessário que os triângulos tenham uma dimensão similar, homogénea [HS98]. Recentemente com a introdução do *Geometry shader* no *pipeline* das placas gráficas é utilizada a técnica *Tessellation*[Ngu07] para refinar a geometria, obtendo os resultados pretendidos[LD08]. Outra dificuldade é a projecção num campo de visão de

180º, bastante superior ao normal, o que implica um ângulo sólido grande por *pixel*.

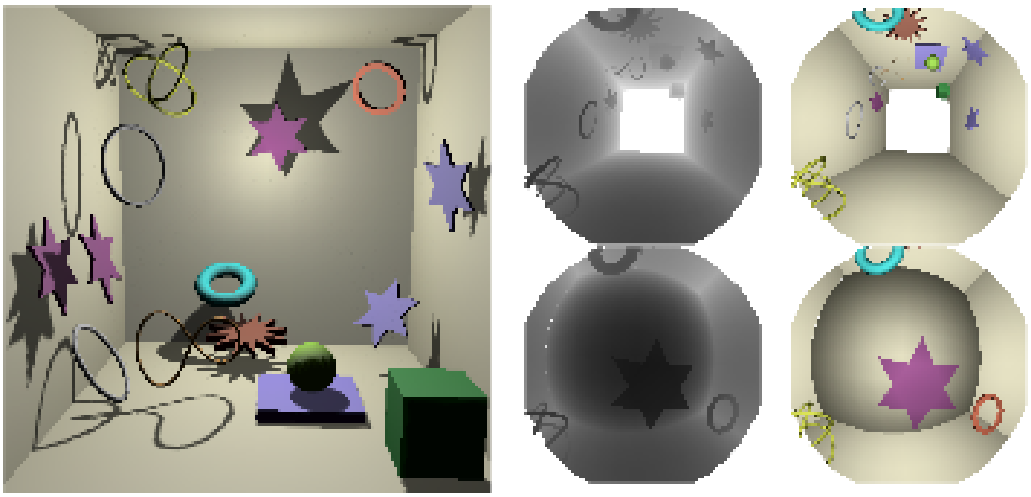

Figura 2.4 – À esquerda, cena iluminada com recurso ao *shadow mapping* parabólico, fonte de luz omnidirecional. Ao centro, representação dos dois mapas de profundidade. À direita, as duas semi-esferas vistas a partir da fonte de luz. Figura retirada do artigo [BApS02].

Com esta abordagem gera-se, facilmente, mapas de profundidade[BApS02] em apenas um passo de *rendering* para uma fonte de luz hemi-direccional e em dois passos para fontes de luz omnidirecional (figura 2.4).
Uma abordagem diferente para as luzes virtuais são os *Reflective Shadow Maps*, introduzido por Carsten Dachsbacher e Marc Stamminger em 2005[DS05]. Esta abordagem é um incremento ao *shadow mapping* para recriar também a iluminação indirecta. A ideia consiste em criar caches de luz indirectas (luzes virtuais) nos pontos projectados do mapa de profundidades. Para tal ao criar o mapa de profundidade são também criados mais três mapas, com as coordenadas mundo dos pontos, as normais e a luz reflectida. Para computar a iluminação indirecta, num determinado ponto, é somada a con-

tribuição da luz reflectida dos pontos vizinhos do mapa. A contribuição de cada ponto vizinho é calculada em função da distância e produto interno entre as normais. O grande problema desta abordagem, como é referido no próprio artigo[DS05], é o facto de não existirem testes de oclusões para a iluminação indirecta. O que resulta em sobrestimar esta componente.

Capítulo 3

Abordagem

Neste trabalho pretende-se construir soluções híbridas entre a técnica *ray tracing* e técnicas aplicadas nas placa gráficas modernas. Numa primeira fase substituir os raios primários com a técnica de projecção. Depois adaptar o *shadow mapping* para a iluminação directa, e adicionalmente estender o *shadow mapping* às luzes virtuais.

3.1 Determinação de geometria visível: raios primários e projecção

A geometria visível a partir de uma determinada câmera, ponto de visão, é toda a geometria delimitada pelo campo de visão da câmera e sem obstáculos entre esta e a câmera.

Com *ray trancing* a determinação da geometria visível é feita com raios primários. Estes raios são gerados com origem na câmera e com uma determinada direcção, segundo a configuração da câmera, e indexados aos *pixeis* respectivos da imagem a ser gerada. O número de raios primários por *pixel* é um parâmetro do *render*. Ao longo desta tese usamos um raio por *pixel*. Para

cada raio determina-se a geometria mais próxima. Em termos computacionais, segundo o algoritmo de *ray tracing*, é testada a intersecção do raio com os elementos geométricos para determinar o elemento mais próximo. Esta solução possuiu um custo computacional elevado.

A projecção em perspectiva, $\mathbb{R}^3 \rightarrow \mathbb{R}^2$, consiste em projectar elementos $3D$ num plano, neste caso uma imagem. Ao projectar elementos $3D$ é necessário salvaguardar a ordem correcta dos elementos em relação à profundidade. Para tal é utilizado o *z-buffer*, um mapa em que cada elemento deste corresponde a um elemento da imagem. Ao projectar um elemento geométrico é calculada também a sua profundidade para cada *pixel*, caso a profundidade seja inferior ao valor do elemento do *z-buffer* correspondente então preenche o *pixel*, senão, está atrás do valor existente, logo não é visível. E, em termos de *hardware* dispomos de placas gráficas com uma arquitectura desenvolvida especificamente para acelerar o processo da projecção.

Propõe-se então substituir no *ray tracer* o processo de disparar raios primários com *ray tracing* pela projecção, utilizando a ferramenta OpenGL, para determinar a geometria vísivel.

3.1.1 Concepção

Esta abordagem vai ser aplicada no iRT. O iRT é *"um motor de Síntese de Imagem interactivo, baseado no algoritmo de Ray Tracing"*, e corre exclusivamente em CPU's. O *ray tracing* do iRT recebendo um raio primário determina o ponto de intersecção mais próximo da origem do raio, e o identificador do triângulo (do tipo inteiro) intersectado. É então necessário configurar o OpenGL para devolver, o ponto de intersecção e o identificador do triângulo intersectado para cada *pixel* da imagem resultante.

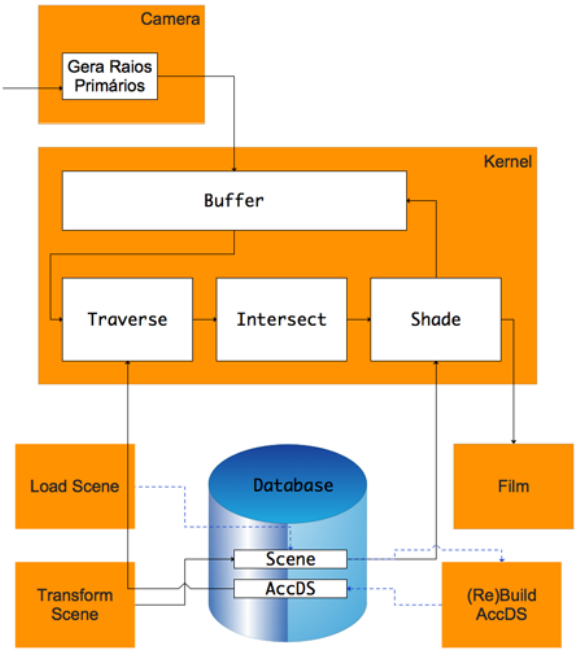

Figura 3.1 – Arquitectura do iRT. A aplicação inicia (transições a azul tracejado) por carregar a cena, e é criada a estrutura de aceleração AccDS. A construção da imagem inicia na câmera que gera os raios primários, e os deposita num *buffer*. Com raios no *buffer* o *kernel* determina para cada raio, através do Traverse e Intersect, a geometria intersectada mais próxima da origem do raio. E, a componente Shade computa a contribuição do raio com a equação de *rendering*, inserindo o resultado na imagem (componente Film). Caso sejam necessários novos raios, o Shade gera-os e insere-os no *buffer*. O processo continua até o Buffer estar vazio. Por último é apresentada a imagem.

Antes de processar a informação é necessário configurar o OpenGL para simular a câmera[1] correspondente no iRT.

O OpenGL apenas computa tipos de dados pré-definidos, como triângulos,

[1] O OpenGL não possuiu uma noção de câmera concreta. http://www.opengl.org/resources/faq/technical/viewing.htm

cores, normais, texturas, entre outras, não permitindo introduzir novos tipos de dados, como o identificador de triangulo. A primitiva geométrica do iRT é o triângulo, descrito por três vértices em coordenadas mundo. Então, basta inserir directamente no OpenGL os triângulos. Para associar ao triângulo o seu identificador, como não é necessário colorir os triângulos com a cor original no OpenGL, então, é através da componente cor que se transporta a informação do identificador do triângulo. Então para cada triângulo é-lhe associado uma cor que corresponde ao identificador do triângulo.

Para produzir o resultado pretendido é necessário programar parte do *pipeline* do OpenGL através do GLSL. No *vertex shader* é passada a informação contida na cor e criada uma variável do tipo *varying* vec3 (contém três variáveis de virgula flutuante) onde é atribuída a coordenada mundo do vértice. Isto permite que no *fragment shader* seja recebida a posição tridimensional, interpolada em relação aos vértices do triângulo, correspondente ao *pixel*, o que corresponde, ao ponto de intersecção, em coordenadas mundo, do raio primário com o triângulo mais próximo da origem do raio, e a informação contida na cor do triângulo. Cada *pixel* da imagem resultante do OpenGL possui quatro variáveis de virgula flutuante, correspondentes ao sistema de cores RGBA, e por norma cada variável com um dominio $D = [0, 1]$. Então, três variáveis são associados às coordenadas mundo, e a última ao identificador do triângulo, já devidamente calculado a partir da cor. Mas, as coordenadas mundo pertencem ao domínio total representado em virgula flutuante, e o identificador do triângulo é um inteiro. Para ultrapassar esta rigidez de domínio, é utilizada a extensão GL_TEXTURE_RECTANGLE_NV no *frame buffer*[2] que permite valores em todo o domínio.

Por último, com a informação produzida no OpenGL, são introduzidos no

[2]Estrutura de memória para onde o OpenGL produz o resultado.

pipeline do iRT raios primários já com o ponto de intersecção calculado e o triângulo correspondente.

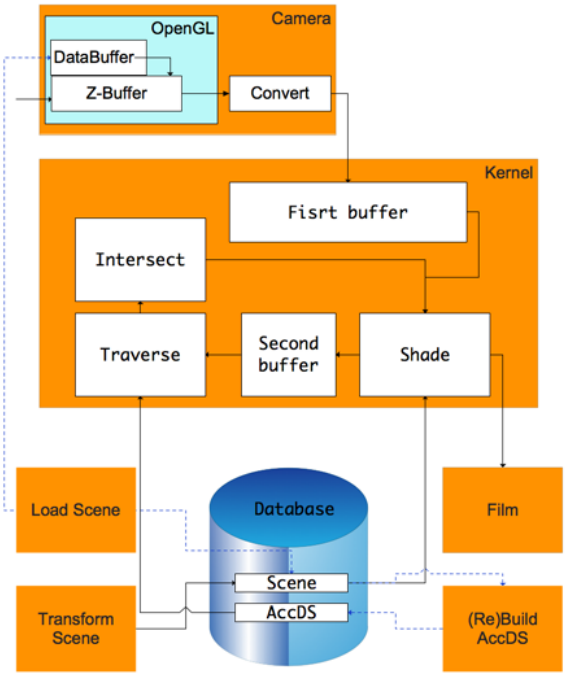

Figura 3.2 – Arquitectura do iRT com a determinação da geometria visível na placa gráfica através do seu *pipeline*. A componente Camera em vez de gerar os raios primários, gera "raios" com a informação da geometria intersectada mais próxima da origem do raio. Este processo é feito na placa gráfica, componente a azul claro, recorrendo ao seu *pipeline* gráfico. A componente Convert cria novos raios com a informação criada pela placa gráfica.Este novo tipo de raio é enviado para a componente Shade, não é necessário recorrer ao Traverse e Intersect.

3.2 Iluminação directa

Uma parte fundamental na iluminação de um ponto é a contribuição directa das fontes de luz. Para cada ponto a iluminar é necessário determinar se esse dado ponto é iluminado directamente pelas fontes de luz. Aqui vamos apenas abordar as fontes de luzes pontuais.

Com *ray tracing* dispara-se um raio sombra para cada fonte de luz a fim de se determinar a visibilidade das fontes de luz a partir do ponto em questão. Se um dado raio disparado para uma fonte de luz intersectar geometria antes da fonte de luz, então, a fonte de luz está ocludida, ou seja, o ponto não é iluminado directamente pela fonte de luz em questão. Esta abordagem é bastante realista, pois determina com exactidão a iluminação directa. Porém, dependendo da resolução da imagem resultante, facilmente se atinge um número elevado (na ordem dos milhares) de pontos a computar. E, existindo n fontes de luz, então, para cada ponto são disparados n raios para determinar a oclusão. O que acarreta um enorme custo computacional.

A técnica *shadow mapping* consiste em criar primeiro um mapa de profundidade a partir da fonte de luz antes do processo de *render*. E, durante o processo de *render*, testar para cada ponto a computar se a distância entre este e a fonte de luz é superior à previamente guardada no elemento correspondente do mapa. Caso a distância seja superior, então existe algum objecto a ocludir o ponto em questão, senão, é o próprio objecto, logo não está ocludido. Ao contrário da versão com *ray tracing*, aqui, em tempo de execução existe um custo computacional reduzido, pois são cálculos de custo constante. Porém, como o mapa possuiu uma dimensão finita, em objectos distantes da fonte de luz surgem problemas de *aliasing*[3].

Propõe-se então, no *ray tracer*, em vez de disparar raios sombra utilizar o

[3]Ver secção seguinte

shadow mapping para determinar a oclusão das fontes de luz.

3.2.1 Shadow Mapping e Ray Tracing Selectivo

Para eliminar o efeito *aliasing* propõe-se disparar raios sombra, utilizando o *ray tracing*, nos contornos das sombras para a fonte de luz respectiva (imagem 3.3), uma vez que o *aliasing* é visível nos contornos das sombras. E, para garantir que abrange correctamente o contorno real da sombra em caso de sub-amostragem, é dilatado o contorno da sombra produzida pelo *shaodw mapping*. Ao disparar um raio sombra, será testado em primeiro lugar se intersecta o triângulo que originou a profundidade no elemento correspondente do mapa de profundidades. Como existe uma probabilidade deste triângulo ser o triângulo que oclui a geometria visível na borda da sombra, procedemos a este cálculo como um despiste. Assim, caso seja o triângulo que oclui, evitamos testar o raio com a restante geometria.
Espera-se assim, apesar de não eliminar totalmente os raios sombra, manter a qualidade da imagem, aspecto essencial. A esta abordagem designaremos de *shadow mapping* e *ray tracing* selectivo (SMeRTS).

3.2.2 Concepção iRT

As fontes de luz omnidirecionais emitem luz em todas as direcções. Para representar a profundidade em todas as direcções é utilizado um cubo centrado na fonte de luz, em que cada face representa um mapa de profundidades. Assim, para cada fonte de luz da cena é criado um cubo, seis mapas de profundidade. A cada elemento do mapa de profundidade é também associado o identificador do triângulo projectado correspondente. A construção dos ma-

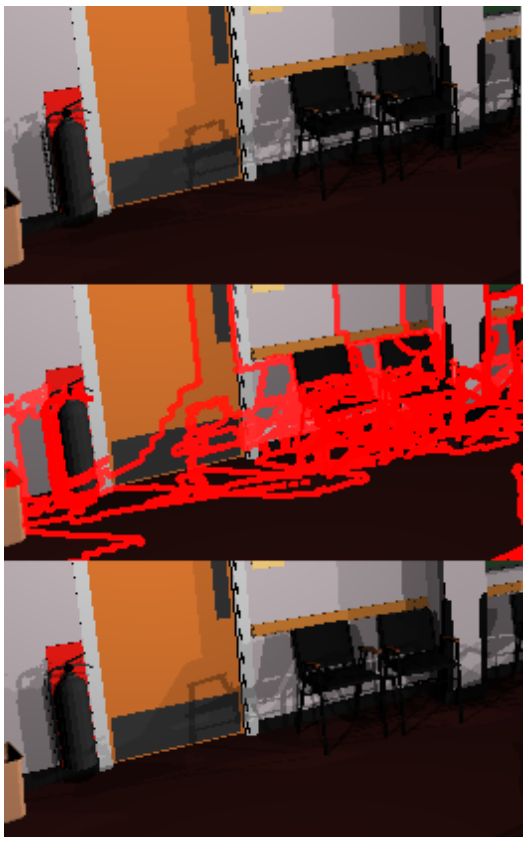

Figura 3.3 – Exemplo da aplicação do *shadow mapping* e *ray tracing* selectivo. Ao alto, imagem gerada com iluminação directa e o teste de oclusão feito exclusivamente com *shadow mapping*. Os contornos das sombras, e seus vizinhos, são marcadas a vermelho, imagem ao centro. Os contornos são refinados recorrendo ao *ray tracing*, gerando a última imagem. O *aliasing* diminui, sendo que os artefactos restantes se devem à utilização da projecção.

pas é feito através do OpenGL, projecção em perspectiva. Além do mapa é também guardada a matriz de projecção para mais tarde projectar no mapa os pontos a computar, determinando a posição correspondente e a distância. Todo o processo é feito na placa gráfica e a informação lá permanece.

O iRT durante o processo de *rendering*, após determinar a geometria visível

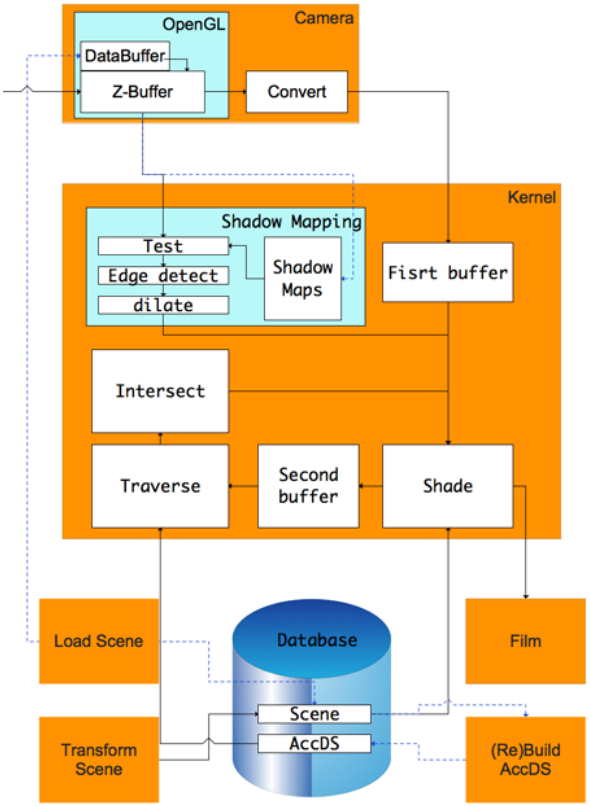

Figura 3.4 – Arquitectura do iRT agora com *shadow mapping*. Inicialmente, tracejado a azul, o iRT constrói os mapas de profundidade para cada fonte de luz e guarda-os (Shadow Maps). Após determinar a geometria visível a placa gráfica, a azul claro, testa a visibilidade para as fontes de luz, detecta os limites e assinala-os e os seus vizinhos. Esta informação é depois consultada pelo Shade.

pela câmera, inicia o processo de teste de oclusão na placa gráfica, programado em CUDA. Este processo recebe a informação produzida pelo OpenGL. Os elementos da informação recebida são projectados nos cubos das fontes de luz, e é realizado o teste de oclusão. Isto produz um mapa, mapa de oclusões,

de igual dimensão à imagem final, para cada fonte de luz. Cada elemento deste mapa indica se está ocluído ou não, e é-lhe associado o identificador do triângulo do mapa de profundidades que oclui. Para uma visualização sem tratamento do efeito *aliasing*, os mapas são enviados para a memória principal. O iRT ao iluminar os pontos visíveis, para cada fonte de luz em vez de disparar um raio sombra verifica na informação recebida se está ocluído.
A eliminação do efeito *aliasing* é realizada da seguinte forma ainda na placa gráfica. Para cada mapa resultante do algoritmo *shadow mapping*, determinam-se os contornos das sombras, ou seja o contorno entre zonas ocluídas e zonas não ocluídas. Para determinar este contorno é realizada uma convolução com o kernel de Laplace (tabela 3.1).

Tabela 3.1 – Kernel de Laplace

0	-1	0
-1	4	-1
0	-1	0

Os elementos do mapa de oclusões que estão num contorno são assinalados como elementos indefinidos[4]. Com os contornos das sombras determinados, é necessário expandi-los em espessura. Esta expansão é necessária para abranger a área afectada pelo *aliasing*. Então, no mapa de oclusões, para cada elemento indefinido os seus vizinhos também são assinalados como indefinidos. Uma vez processados, os mapas são enviados para a memória principal. Agora o iRT ao computar a iluminação, nos pontos visíveis, para cada fonte de luz antes de disparar um raio sombra verifica o estado na informação re-

[4]O elemento não é imediatamente assinalado para evitar erros ao computar os elementos vizinhos. Só no fim de processar o mapa é então assinalado como elemento duvidoso.

cebida. Se está ocluído então o ponto não é iluminado pela fonte de luz. Se não está ocluído, é iluminado pela fonte de luz. Por fim, se for indefinido então constrói um raio sombra e testa a intersecção com o triângulo associado no mapa de oclusões, caso seja intersectado então está ocluído, senão é disparado o raio sombra.

3.2.3 Concepção Nvidia Optix

Com o surgimento de novas tecnologias, nomeadamente o *ray tracer* Nvidia Optix, decidiu-se aplicar esta abordagem numa nova aplicação que utiliza o Optix. A concepção não é similar à do iRT. O mapa de profundidade da fonte de luz é agora representado por dois mapas de profundidades parabólico, em vez dum cubo, diminuindo o tempo da construção. Estes mapas de profundidades parabólico são construídos no OpenGL com base no algoritmo do artigo "*Shadow mapping for hemispherical and omnidirectional light sources*"[BApS02] e com a divisão de triângulos no *geometry shader* para diminuir os erros das projecções. Devido ao grande ângulo sólido por *pixel* a resolução dos mapas de profundidade é superior à da concepção do iRT para atenuar o aumento do ângulo. Este processo só é executado ao iniciar, ou, quando existem alterações na cena. Construídos os mapas de profundidade, estes são transferidos para o motor de *ray tracing* Optix. Ainda de salientar que não foi possível atribuir um identificador de triângulo aos triângulos guardados na estrutura de dados do Optix. Este facto inviabilizará o pré teste de intersecção do raio sombra com o triângulo associado no mapa de oclusões.

Um dos presentes problemas do Optix é a inexistência de interoperabilidade com o CUDA na placa gráfica. E, para utilizar o CUDA, aproveitando o tra-

balho produzido na concepção anterior, seria necessário enviar a informação do Optix na placa gráfica para a memória principal e reenviar a informação para a placa gráfica para o CUDA. Findo o processo em CUDA, o processo repete-se na ordem inversa. Esta solução não é viável devido à sua redundância e tempo despendido. Então, no processo de *render*, ao determinar a geometria visível pela câmera é guardada a informação essencial, associada ao *pixel*, para mais tarde computar a sua iluminação directa. Isto deve-se ao facto do Optix não ser transparente na ordem e tempo em que processa os raios. Ou seja, não existe a garantia que o raio primário do *pixel* vizinho já tenha determinado o ponto visível, inviabilizando o teste de contorno da sombra. Após terminar o processo de *ray tracing*, ainda com o Optix é analisada a informação previamente guardada. Esta análise consiste em detectar para cada fonte de luz e *pixel* se o *pixel* está num "raio" de vizinhança do contorno da sombra. Este processo é equivalente ao processo detecção de contornos e dilatação. Ao verificar que está numa vizinhança do contorno da sombra, dispara um raio sombra para determinar com exactidão a oclusão. Caso não esteja na vizinhança, projecta o ponto no mapa de profundidade a fim de determinar a oclusão da fonte de luz.

3.3 Iluminação Indirecta

Neste trabalho não vamos abordar a distribuição das luzes virtuais, mas sim, o cálculo de oclusão para efeitos de iluminação do ponto. Ao iluminar um ponto, tal como para a iluminação directa, é necessário determinar a contribuição das luzes virtuais. Para tal, é necessário determinar se as fontes de luz virtual estão ocluídas. Mas, de lembrar que, as fontes de luz virtual, como estão localizadas em superfícies, apenas contribuem na direcção da semi-esfera

orientada pela normal da superfície. Este processo é muito semelhante à secção anterior. Porém, enquanto que uma dada cena normalmente possuiu algumas fontes de luzes, o número de luzes virtuais, para atingir o efeito desejado, atinge a ordem das centenas, ou milhares, pois, normalmente, é um múltiplo do número de fontes de luz.

Para esta abordagem, como emite apenas na direcção de uma semi-esfera, é utilizado um mapa de profundidades parabólico, em vez dos cinco mapas de profundidades exigidos pela abordagem do semi-cubo. Com a abordagem do semi-cubo e tendo em conta o número de luzes virtuais, isto implicaria um processo moroso e um consumo elevado de memória. E, como se prevê a existência elevada de um elevado número de luzes virtuais, o que diminuiu a contribuição por luz virtual, espera-se que potenciais efeitos inerentes ao elevado ângulo sólido e de *aliasing* sejam diluídos, mantendo a qualidade da imagem. Aqui não é aplicada a abordagem SMeRTS.

3.3.1 Concepção

Esta abordagem será aplicada numa aplicação que já utiliza luzes virtuais e recorre ao Nvidia Optix como motor de *ray tracing*. Então, para cada luz virtual cria-se um mapa de profundidades parabólico e respectiva a matriz de projecção. Este mapa de profundidades parabólico é construído em OpenGL[BApS02] e com a divisão de triângulos no *geometry shader* para diminuir os erros das projecções. Este processo só é executado ao iniciar ou quando existem alterações na cena. Construídos os mapas de profundidade, estes são transferidos para o motor de *ray tracing* Optix. Ao iluminar um ponto em vez de disparar raios sombra para as luzes virtuais, executa o teste de oclusão com os mapas de profundidade das luzes virtuais.

Os mapas de profundidade são guardados em texturas. Porém, uma das limitações da placa gráfica usada é o número de texturas que está limitado a 128, e não é possível criar em Optix vectores de texturas. E, o Optix utiliza texturas para guardar informação acerca do material da geometria. Tendo em conta estes factores o número de luzes virtuais foi fixado em 45.

Capítulo 4

Testes e resultados

Neste capítulo vamos apresentar os testes e resultados, incidindo principalmente no desempenho e qualidade, e será feita uma comparação com a versão referência. A versão referência corresponde ao *Ray Tracing* clássico[Whi80], ou seja, com determinação de geometria visível pela câmera, iluminação directa e reflexão especular. No Optix contempla também a iluminação indirecta difusa. Para aferir o desempenho serão utilizadas duas métricas. A primeira métrica incide no número de imagens geradas por segundo (*frames per sencond*, fps). Tendo em conta que o sistema operativo pode lançar tarefas alheias à aplicação em questão, afectando o desempenho, a aplicação será executada n vezes, medindo o seu desempenho, e no fim é calculada a média desta métrica.

$$FPS = \frac{\sum_{i=1}^{n} FPS_i}{n} \tag{4.1}$$

A segunda é o número de raios eliminados, dada a sua importância e impacto no desempenho. Como o números de raios é fixo para cada trio cena, posição da câmera e resolução da imagem apenas uma execução é necessária para medir esta métrica.

A qualidade da imagem será aplicada às abordagens de iluminação directa e indirecta para comparar a imagem referência com a versão optimizada. É utilizada a métrica *Peak signal-to-noise ratio* (PSNR)[TM10], em decibéis (dB). Esta métrica é usada para comparação de qualidade de imagem entre a imagem referência e a imagem gerada pela nossa abordagem. Para calcular o PSNR é necessário calcular primeiro o *mean-squared error* (MSE)(equação 4.2), onde M e N referem-se às dimensões da imagem.

$$MSE = \frac{\sum_{M,N}[Imagem_1(m,n) - Imagem_2(m,n)]^2}{M * N} \tag{4.2}$$

E, depois resolver a equação do PSNR (4.3), onde MAX refere-se ao valor mais alto possível por *pixel*. Um PSNR de valor infinito indica que as imagens em comparação são exactamente iguais. Por oposto, 0 dB indica imagens completamente díspares entre si. Para a nossa análise os valores acima de 20 dB são considerados aceitáveis, e com 30 dB ou mais são bons valores.

$$PSNR = 10 \cdot \log_{10}(\frac{MAX^2}{MSE}) \tag{4.3}$$

Porém o PSNR é um valor escalar, enquanto as quantidades que descrevem um *pixel* são vectoriais: vermelho, verde e azul (RGB). Então, o RGB é convertido em luminância (Y) [TM10], valor escalar. Este conversão tem em conta a percepção do sistema visual humano, dando maior importância à componente verde, depois a vermelha e por fim o azul. A conversão é dada pela equação 4.4.

$$Y = 0.2126R + 0.7152G + 0.0722B \tag{4.4}$$

4.1 Ambiente

Os testes foram executados numa máquina de 64*bits* com o seguinte *hardware*:

- Processador: Intel i5 750, quatro cores a 2.66GHz com 8MB de cache.
- Memória principal: 4GB DDR3-1333
- Plataforma da *motherboard*: Intel P55 Express Chipset
- Placa Gráfica: NVIDIA GeForce 480GTX
  - Memória: 1536 MB GDDR5
  - Número de CUDA cores: 480
  - Largura de banda da memória: 177.4 GB/sec
  - Bus: PCI Express x16

A aplicação iRT é executada num ambiente Linux. A aplicação foi compilada com a optimização O3 e versão multi-processador, com quarto *threads* para obter o maior potencial do processador. Descrição do software:

- Sistema Operativo: Fedora 11 64bits
- Compilador: Intel C/C++ Compiler (ICC) 11 64bits para Linux
- Compilador para placa gráfica: Nvidia CUDA 2.3 e respectivo SDK
- Controlador e interface da placa gráfica: OpenGL 3.0 64bits (versão 190.533 da Nvidia), GLSL 1.3, GLU e GLX.

A aplicação que utiliza luzes virtuais é executada no ambiente Microsoft Windows. A aplicação foi compilada com a optimização O2. Descrição do software:

- Sistema operativo: Microsoft Windows 7 64bits
- Compilador e IDE: Microsoft Visual Studio 9.0
- Compilador para placa gráfica: Nvidia CUDA 2.3 e respectivo SDK
- Motor de *ray tracing*: Nvidia Optix 2.0 beta 5 e respectivo SDK

A dimensão das imagens foi fixada em 640x480 pixeis para todos os testes.

4.2 Cenas

Foram definidas três cenas para testes, a conferencia, o escritório e a Cornell-Box. A cena conferência é uma cena constituída por diversos materiais, em que alguns possuem a componente especular, e 190.951 triângulos. A cena conferência é a mais complexa. A cena escritório é constituída por 20.769 triângulos. Por fim a cena CornellBox, delimitada por um cubo, possuindo um coelho e um espelho colocado na parede, totalizando 69.465 triângulos.

4.3 Determinação de geometria visível: raios primários e projecção

Os testes desta abordagem foram executados no iRT. O iRT é baseado no algoritmo clássico de *ray tracing* estilo Whitted, ou seja, dispara raios pri-

mários para detecção de geometria visível pela câmera, raios especulares para determinar a reflexão e raios sombra para detectar a oclusão das fontes de luz. É disparado um raio primário por pixel, logo, são disparados $640 * 480 * 1 = 307.200$ raios primários. Com a nossa abordagem, projecção, estes raios foram evitados. Na tabela 4.1 podemos verificar o desempenho das duas abordagens. A projecção na placa gráfica para as três cenas varia entre os 0.0030 e os 0.0019 segundos.

A eliminação dos raios primários permitiu reduzir o número de raios disparados. Esta redução de números de raios, com a projecção, aumentou ligeiramente o desempenho, mas ainda está longe dos mínimo expectável 25 fps exigidos pelo sistema visual humano. Ainda que a projecção no GPU seja bastante rápida, o restante processo no CPU é bastante moroso. Ao analisarmos a tabela 4.1 verifica-se também que o número de raios sombra para as três cenas é extremamente elevado.

4.4 Iluminação directa no iRT

Os testes para iluminação directa focam-se no número de raios utilizados, desempenho e qualidade da imagem. Os teste foram feitos, tal como a análise anterior, no iRT. Em comparação estão duas versões, a referência e a abordagem SMeRTS. Na versão referência a determinação da geometria visível pela câmera é determinada com a projecção no GPU, e o teste de oclusão das fontes de luz, iluminação directa, e a componente especular são feitas exclusivamente com *ray tracing*. A abordagem com SMeRTS, a determinação da geometria visível pela câmera é feita também através da projecção no GPU, o teste de oclusão das fontes de luz é feito recorrendo ao SMeRTS e a componente especular é determinada exclusivamente com *ray tracing*.

Tabela 4.1 – Comparativo de desempenho com *ray tracing* ou projecção para determinação de geometria visível pela câmera. A utilização da projecção, em vez da utilização dos raios primários, permitiu aumentar ligeiramente o desempenho.

Cenas	Métrica	*Ray Tracing*	Projecção
Conferência	Desempenho	0,33 fps	0,48 fps
	Nº de raios primários	307.200	0
	Nº de raios sombra	1.483.811	1.454.268
	Nº de raios especulares	92.857	73.974
	Nº total de raios	1.883.868	1.528.242
Escritório	Desempenho	0,68 fps	0,78 fps
	Nº de raios primários	307.200	0
	Nº de raios sombra	2.868.147	2.823.805
	Nº de raios especulares	62.693	55.698
	Nº total de raios	3.238.040	2.879.503
CornellBox	Desempenho	2,82 fps	2,86 fps
	Nº de raios primários	307.200	0
	Nº de raios sombra	1.859.094	1.908.247
	Nº de raios especulares	247.788	260.816
	Nº total de raios	2.414.082	2.169.063

A qualidade das imagens da abordagem SMeRTS, segunda linha da tabela 4.2, é relativamente baixa, exceptuando a da cena CornellBox, 34,12 dB. Na cena escritório (figura 4.1) as grelhas do lado esquerdo estão muito próximas e possuem uma geometria muito pequena. Nesta situação, a nossa abordagem não alcançou um resultado satisfatório, o que se reflectiu na qualidade

Tabela 4.2 – Comparativo de desempenho entre versão referência e versão com SMeRTS em iluminação directa no iRT. Com o SMeRTS verificou-se um aumento de desempenho. A qualidade da imagem da cena escritório está abaixo do aceitável. Por outro lado, a qualidade da imagem da cena CornellBox é boa, e a conferência aceitável. A redução do número de raios sombra na versão SMeRTS em relação à versão referência é assinalável.

Cenas	Métrica	Versão referência	versão SMeRTS
Conferência	Desempenho	0,48 fps	1,44 fps
	Nº de fontes de luz	4	4
	Qualidade da imagem	∞	26,55 dB
	Nº de raios sombra	1.454.268	317.616
	Nº de raios especulares	73.974	73.974
	Nº total de raios	1.528.242	391.590
Escritório	Desempenho	0,78 fps	3,13 fps
	Nº de fontes de luz	9	9
	Qualidade da imagem	∞	19,29 dB
	Nº de raios sombra	2.823.805	503.185
	Nº de raios especulares	55.698	55.698
	Nº total de raios	2.879.503	558.883
CornellBox	Desempenho	2,86 fps	4,75 fps
	Nº de fontes de luz	5	5
	Qualidade da imagem	∞	34,12 dB
	Nº de raios sombra	1.908.247	665.338
	Nº de raios especulares	260.816	260.816
	Nº total de raios	2.169.063	926.154

da imagem, 19,29 dB. A mesma situação surge na cena conferência (imagem 4.2) no topo das cadeiras, o que compromete a qualidade, 26,55 dB.

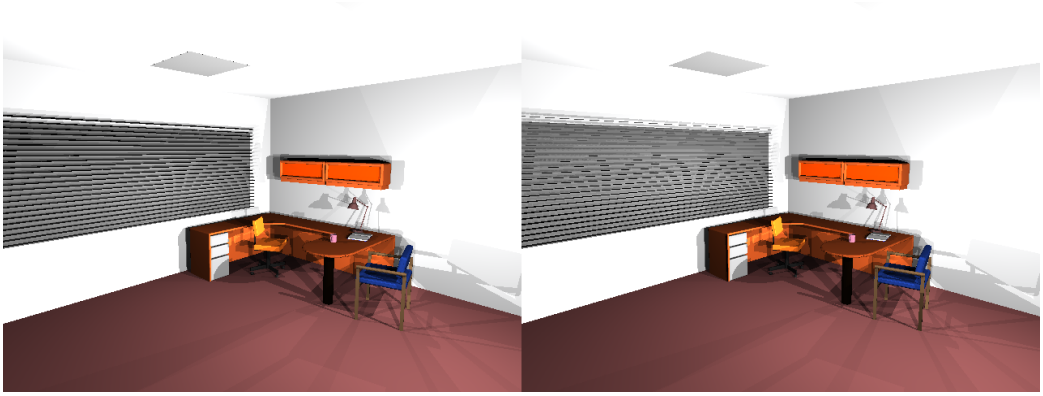

Figura 4.1 – Cena escritório com iluminação directa no iRT. À esquerda, a imagem referência, com a determinação das sombras recorrendo exclusivamente ao *ray tracing*. À direita, imagem gerada pela abordagem SMeRTS.

O número de fontes de luz tem um forte impacto no número raios sombra criados. Na versão referência o número de raios sombra atinge um número extremamente elevado, ultrapassando a barreira dos milhões. Ao utilizar apenas o *shadow mapping* estes raios seriam totalmente eliminados. Mas, com a nossa abordagem SMeRTS esses raios não são totalmente eliminados para refinar os contornos das sombras. Na tabela 4.2 podemos verificar que a abordagem SMeRTS permitiu reduzir mais de um milhão de raios sombra para as cenas conferência e CornellBox e mais de dois milhões de raios sombra para a cena escritório. De assinalar que na tabela 4.2 estão também contabilizados os raios sombra despoletados pelos raios especulares. E, a iluminação directa desses raios especulares não é tratada pela nossa abordagem SMeRTS.

A abordagem SMeRTS permitiu um aumento de desempenho do processo

Figura 4.2 – Cena conferência com iluminação directa no iRT. À esquerda, a imagem referência, com a determinação das sombras recorrendo exclusivamente ao *ray tracing*. À direita, imagem gerada pela abordagem SMeRTS.

de *render*. O processo do SMeRTS na placa gráfica para cada fonte de luz no GPU tem uma duração inferior a 10ms. Mas, não é suficiente para elevar o desempenho até aos 25 fps, isto porque o processo no CPU tem um contributo pesado no tempo total para gerar a imagem.

4.5 Iluminação directa em Nvidia Optix

Nesta secção são apresentados os resultados da iluminação directa no Nvidia Optix. A nossa aplicação no Nvidia Optix é do estilo Whitted mais a iluminação indirecta recorrendo às luzes virtuais. Tal como na análise anterior, serão comparadas duas versões: versão referência e versão com a abordagem SMeRTS. A abordagem referência contempla toda a determinação de geometria visível, iluminação directa, iluminação indirecta e o a componente especular com *ray tracing*. A abordagem SMeRTS é diferente da referência na iluminação directa, onde o teste da oclusão das fontes de luz é feito com

a abordagem SMeRTS.

Devido a limites no número de texturas suportadas pela placa gráfica não foi possível testar a cena conferência. E, o número de fontes de luz, em cada cena, foi reduzido para suportar as luzes virtuais.

A qualidade das imagens geradas com a abordagem SMeRTS, tabela 4.3, relativamente à imagem referência é alta. Mas, o problema das grelhas da cena escritório persiste.

Tabela 4.3 – Comparativo de desempenho entre versão referência e versão com SMeRTS em iluminação directa no Optix. O número de raios sombra assinalados na tabela corresponde unicamente aos raios disparados para as fontes de luz, iluminação directa.

Cenas	Métrica	Versão referência	versão SMeRTS
Escritório	Desempenho	3,37 fps	3,33 fps
	Nº de fontes de luz	2	2
	Qualidade da imagem	∞	39,06 dB
	Nº de raios sombra	598.870	31.100
CornellBox	Desempenho	4,44 fps	4,21 fps
	Nº de fontes de luz	1	1
	Qualidade da imagem	∞	45,91 dB
	Nº de raios sombra	305.155	3.339

A redução do número de fontes de luz também reduziu, face à secção anterior, o número de raios sombra disparados. Porém, contrariamente, o desempenho piorou com a abordagem SMeRT (tabela 4.3). Numa primeira análise este resultado parece paradoxal! Mas não, estes resultados devem-se à arquitectura do GPU, processamento massivo em paralelo. Os actuais GPUs das

Nvidia processam centenas de *threads* ao mesmo tempo. Mas, essas *threads* são agrupadas em *warps*, e as *threads* de cada *warp* estão sincronizadas, ou seja, estão a executar a mesma instrução, ou, caso o processo seja divergente, estão em espera para que todas as *threads* do *warp* sincronizem. Ora, no Optix cada raio é atribuído a uma *thread*, e os seus descendentes também estão associados a essa *thread*. Se num determinado momento, com a abordagem SMeRTS, num *warp*, uma *thread* dispara um raio sombra para testar a oclusão da fonte de luz, todas as *threads* vizinhas caso fiquem pelo teste do *shadow mapping* ficam em espera até ser determinada a oclusão pelo raio sombra. Ou seja, processar alguns raios, ou muitos raios, desde que não ultrapasse os limites do GPU, tem o mesmo custo temporal. Então basta existirem alguns raios sombra para que o desempenho seja semelhante à versão referência. E se juntarmos ainda a computação prévia do SMeRTS, projecção no mapa de profundidade e detecção dos contornos da sombras, ficamos com um processo mais lento.

4.6 Iluminação Indirecta

A iluminação indirecta, através das luzes virtuais, é aplicada no Optix. A versão referência é a mesma da secção anterior. A nossa abordagem difere da versão referência no teste de oclusão das fontes de luz virtual à qual recorremos exclusivamente ao *shadow mapping*. Aqui também, devido a limites de memória da placa gráfica, não foi possível testar a cena conferência.

Ao analisarmos a tabela 4.4 verificamos que a versão referência, para as duas cenas, utiliza cerca de 12 milhões de raios sombra para testar a oclusão das fontes de luz virtuais. Este facto reflecte-se no desempenho, obtendo-se valores baixos. A abordagem do *shadow mapping* evitou todos estes raios

sombra, e, aumentou significativamente o desempenho. A CornellBox atinge uma média de 56,62 fps. E, a qualidade da imagem não foi significativamente afectada, possuindo valores bastante bons: 42,86 dB para a cena escritório e 34,72 dB para a cena CornellBox.

Tabela 4.4 – Comparativo de desempenho e qualidade da iluminação indirecta. Na versão referência são utilizados milhões de raios sombra para detectar a oclusão das luzes virtuais. O *shadow mapping* proporcionou um aumento de desempenho considerável em relação à versão referência. Os raios sombra assinalados referem-se apenas aos raios sombra disparados para as fonte de luz virtuais, iluminação indirecta.

Cenas	Métrica	Referência	Shadow Mapping
Escritório	Desempenho	3,37 fps	27,04 fps
	Nº de fontes de luz	2	2
	Nº de fontes de luz virtuais	45	45
	Qualidade da imagem	∞	42,86 dB
	Nº de raios sombra	12.971.207	0
CornellBox	Desempenho	4,44 fps	56,62 fps
	Nº de fontes de luz	1	1
	Nº de fontes de luz virtuais	45	45
	Qualidade da imagem	∞	34,72 dB
	Nº de raios sombra	12.225.116	0

4.7 Resultados cumulativos no Optix

Nesta secção é comparada a versão referência e a versão cumulativa com SMeRTS para iluminação directa e *shadow mapping* na iluminação indirecta,

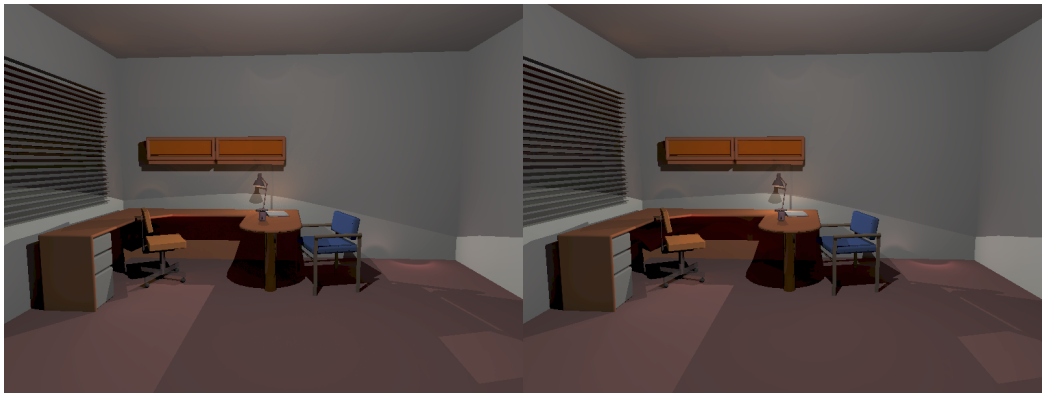

Figura 4.3 – Cena escritório com iluminação global no Optix. À esquerda, a imagem referência, com a determinação das sombras das luzes virtuais recorrendo exclusivamente ao *ray tracing*. À direita, imagem gerada com *shadow mapping* para as luzes virtuais.

a que designamos por SMeRTS_&_SM. Na versão referência a determinação de geometria visível, iluminação directa, iluminação indirecta e o a componente especular com *ray tracing*. A abordagem SMeRTS_&_SM diferencia da referência na iluminação directa, onde o teste da oclusão das fontes de luz é feito com a abordagem SMeRTS e na iluminação indirecta o teste de oclusão das fontes de luz virtuais é feito exclusivamente com *shadow mapping*.
Na tabela 4.5 podemos verificar que o número de raios sombra foi consideravelmente reduzido. Conforme se verificou na secção 4.5, a aplicação da abordagem SMeRTS não implicou um aumento de desempenho, o que se verifica aqui também na cena escritório em relação à secção anterior. Porém, na cena CornellBox regista-se uma queda de desempenho em relação à secção anterior. Esta queda de desempenho pode estar relacionada com vários factores da arquitectura da placa gráfica ou do Optix. Este comportamento, diferente do esperado, será alvo de um estudo futuro, com vista a identifi-

Tabela 4.5 – Comparativo de desempenho entre versão referência e versão SMeRTS_&_SM no Optix. O número de raios sombra assinalados na tabela corresponde ao total dos raios disparados, quer para as as fontes de luz, quer para as fontes de luz virtuais.

Cenas	Métrica	Referência	SMeRTS_&_SM
Escritório	Desempenho	3,37 fps	28,15 fps
	Nº de fontes de luz	2	2
	Nº de fontes de luz virtuais	45	45
	Qualidade da imagem	∞	37,45 dB
	Nº de raios sombra	13.570.077	31.100
CornellBox	Desempenho	4,44 fps	38,71 fps
	Nº de fontes de luz	1	1
	Nº de fontes de luz virtuais	45	45
	Qualidade da imagem	∞	34,41 dB
	Nº de raios sombra	12.530.271	3.339

car claramente as causas do mesmo. A qualidade da imagem, com a versão SMeRTS_&_SM, para a cena escritório é de 37,45 dB e 34,41 dB para a cena CornellBox.

Capítulo 5

Conclusão

Nesta tese propôs-se três abordagens para diminuir o número de raios a disparar num *ray tracer*, com o objectivo de acelerar o processo de *render*. A primeira abordagem consiste eliminar os raios primários do *ray tracing*, determinação da geometria visível pela câmera, substituindo o processo na placa gráfica com a técnica da projecção. Esta abordagem, como se verificou, permitiu aumentar ligeiramente a performance. A segunda abordagem, SMeRTS, permitiu reduzir o número de raios sombra para testar a oclusão das fontes de luz. A utilização do *shadow mapping* aliada à técnica de disparar raios sombras nos contornos das sombras para refinar, eliminando o efeito *aliasing*, permite uma iluminação directa da cena com qualidade aceitável. A sua aplicação em diferentes arquitecturas, iRT e Nvidia Optix, produziu resultados diferentes em termos de desempenho. Na primeira registou-se um aumento de desempenho, na segunda uma ligeira diminuição devido a questões da arquitectura do GPU. Por último, a aplicação do *shadow mapping* à iluminação indirecta permitiu eliminar por completo os raios sombra necessários para determinar a oclusão das fontes de luz virtual. A baixa contribuição de cada luz virtual não exige a aplicação da abordagem SMeRTS, e de facto a

qualidade das imagens com esta abordagem é boa. Esta abordagem permite um aumento de desempenho considerável.

Apesar do ligeiro aumento do desempenho registado com a projecção, na determinação da geometria visível pela câmera, esta abordagem não deve ser relegada. A sua utilização futura aliada com o Optix pode resultar em desempenhos superiores. Nesta tese não foi aplicado devido ao enclausuramento da informação da cena por parte da versão do Optix utilizada, não permitindo partilhá-la com o OpenGL. Possuiu um contra, a existência do efeito *aliasing*, ainda que existam técnicas aplicadas nas placas gráficas para o eliminar, essas técnicas baseiam-se em métodos empíricos, dificilmente transponíveis para um contexto híbrido com *ray tracing*. A solução mais usual é o aumento de resolução da imagem, mas é necessário ter em atenção os custos que essa decisão acarreta. A qualidade das imagens da abordagem SMeRTS é satisfatória, mas devido a certas situações de reduzida dimensão da geometria, ficou um pouco aquém do expectável. A aplicação do SMeRTS no Optix ficou aquém das expectativas. A sua aplicação numa arquitectura como a do GPU, não é uma solução eficaz. Por outro lado, a aplicação do *shadow mapping* às luzes virtuais aumentou consideravelmente o desempenho, acima dos 25 fps, mantendo uma boa qualidade da imagem. Esta qualidade deve-se à reduzida contribuição de cada luz virtual para a iluminação global, diminuindo a percepção dos problemas do *shadow mapping*. Mas, o conjunto das luzes virtuais tem uma contribuição essencial para uma boa iluminação da cena.

Para trabalho futuro, a optimização do desempenho e qualidade da iluminação directa parece-me o mais pertinente. O impacto da iluminação directa é de facto muito importante, e a sua qualidade não pode ser descurada. Uma análise mais profunda ao mapa de profundidades para extrair potenciais per-

turbações, erros ou diferenciação dos objectos podem ser os próximos passos. A utilização da projecção para determinação da geometria visível pela câmera numa aliança OpenGL e Optix, será também um dos próximos tópicos a abordar. Por fim, um estudo aprofundado para identificar claramente as causas da baixa de desempenho da cena CornellBox na secção 4.7.

Bibliografia

[BApS02] Stefan Brabec, Thomas Annen, and Hans peter Seidel. Shadow mapping for hemispherical and omnidirectional light sources. 2002.

[BBS+09] Brian C. Budge, Tony Bernardin, Shubhabrata Sengupta, Ken Joy, and John D. Owens. Out-of-core data management for path tracing on hybrid resources. In *Proc. Eurographics 2009 (to appear)*, Munich, Germany, March 2009. IDAV The Institute for Data Analysis and Visualization.

[CHH02] Nathan A. Carr, Jesse D. Hall, and John C. Hart. The ray engine. In *HWWS '02: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 37–46, Aire-la-Ville, Switzerland, Switzerland, 2002. Eurographics Association.

[Chr05] Martin Christen. *Ray Tracing on GPU*. PhD thesis, University of Applied Sciences Basel, Switzerland, January 2005.

[Chr09] Christian Lauterbach, Qi Mo, and Dinesh Manocha. Fast hard and soft shadow generation on complex models using selective

ray tracing. In *UNC CS Technical Report TR09-004*, Chapel Hill, North Carolina, USA, 2009.

[DBB02] Philip Dutre, Kavita Bala, and Philippe Bekaert. *Advanced Global Illumination.* A. K. Peters, Ltd., Natick, MA, USA, 2002.

[DS05] Carsten Dachsbacher and Marc Stamminger. Reflective shadow maps. In *I3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games*, pages 203–231, New York, NY, USA, 2005. ACM.

[HS98] Wolfgang Heidrich and Hans-Peter Seidel. View-independent environment maps. In *HWWS '98: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, pages 39–ff., New York, NY, USA, 1998. ACM.

[Kel97] Alexander Keller. Instant radiosity. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 49–56, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[LD08] Haik Lorenz and Jürgen Döllner. Dynamic mesh refinement on gpu using geometry shaders. In *Proceedings of the 16-th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision 2008*, February 2008.

[MB05] Marc Stamminger Marcel Beister, Manfred Ernst. A hybrid gpu-cpu renderer. In *Vision, Modeling, and Visualization 2005*, pages 415–420, Erlangen, Deutschland, 2005.

[MSD10] Microsoft MSDN. Shader Stages (Direct3D 10), 2010.

[Ngu07] Hubert Nguyen. *Gpu gems 3*. Addison-Wesley Professional, 2007.

[Ope07] OpenGL Architecture Review Board, and Shreiner, Dave and Woo, Mason and Neider, Jackie and Davis, Tom. *OpenGL(R) Programming Guide: The Official Guide to Learning OpenGL(R), Version 2.1*. Addison-Wesley Professional, 2007.

[Ope10] OpenGL. What is OpenGL?, 2010.

[PBD+10] Steven G. Parker, James Bigler, Andreas Dietrich, Heiko Friedrich, Jared Hoberock, David Luebke, David McAllister, Morgan McGuire, Keith Morley, Austin Robison, and Martin Stich. Optix: a general purpose ray tracing engine. *ACM Trans. Graph.*, 29(4):1–13, 2010.

[RLKG+09] Randi J. Rost, Bill Licea-Kane, Dan Ginsburg, John M. Kessenich, Barthold Lichtenbelt, Hugh Malan, and Mike Weiblen. *OpenGL Shading Language*. Addison-Wesley Professional, 2009.

[SD01] NVIDIA Corporation Sim Dietrich. Shadow Techniques, 2001.

[SD02] Marc Stamminger and George Drettakis. Perspective shadow maps. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 557–562, New York, NY, USA, 2002. ACM.

[She09] Michael C. Shebanow. Pervasive massively multithreaded gpu processors. In *CF '09: Proceedings of the 6th ACM conference on Computing frontiers*, pages 227–227, New York, NY, USA, 2009. ACM.

[SWP10] Daniel Scherzer, Michael Wimmer, and Werner Purgathofer. A survey of real-time hard shadow mapping methods. In *State of the Art Reports Eurographics*, pages

[TM10] Inc. The MathWorks. Compute peak signal-to-noise ratio (PSNR) between images, 2010.

[Whi80] Turner Whitted. An improved illumination model for shaded display. *Commun. ACM*, 23(6):343–349, 1980.

[Wil78] Lance Williams. Casting curved shadows on curved surfaces. *SIGGRAPH Comput. Graph.*, 12(3):270–274, 1978.

[ZHWG08] Kun Zhou, Qiming Hou, Rui Wang, and Baining Guo. Real-time kd-tree construction on graphics hardware. In *SIGGRAPH Asia '08: ACM SIGGRAPH Asia 2008 papers*, pages 1–11, New York, NY, USA, 2008. ACM.