UNIVERSIDADE DA BEIRA INTERIOR
Covilhã | Portugal

Geração de Sombras em Cenas 3D por Projeção Direta

João Pedro Domingues Mamede

Dissertação de Mestrado em Engenharia Informática

Supervisão de Professor Doutor Abel João Padrão Gomes

Departmento de Informática
Universidade da Beira Interior
Covilhã, Portugal
Outubro 2013
http://www.di.ubi.pt

Agradecimentos

À minha família, e de modo especial aos meus pais que incessantemente e a todo o momento me acompanharam, ajudaram e incentivaram a prosseguir neste meu propósito e em todo o meu percurso académico que culminou com esta dissertação, para eles o meu esforço, o meu carinho e agradecimento. Desejo e prometo fazer de cada momento da minha vida, o melhor.

Ao meu orientador, Professor Doutor Abel J. P. Gomes, agradeço o acompanhamento e esclarecimento das dúvidas e hesitações. Estou grato pelo apoio, pela amizade e pelo estímulo para almejar os objetos e prossecução deste projeto.

Não esqueço todos os meus amigos e colegas que com a sua quota parte contribuiram para o sucesso deste momento da minha vida e do meu percurso académico.

A todos o meu "obrigado".

João Pedro Mamede
Covilhã, Portugal
20 de Outubro de 2013

Resumo

As sombras têm uma importante contribuição para o aumento do realismo na representação gráfica em tempo real de cenas e objetos. Existem vários algoritmos que permitem adicionar sombras a cenas e objetos 3D, sendo os mais populares o mapa de sombras e o volume de sombras.

O trabalho realizado no âmbito desta dissertação de mestrado teve como principal objetivo a criação de um algoritmo de geração de sombras através de projeções diretas mútuas entre facetas dos objetos em cenas 3D. Com esta finalidade foi explorado um algoritmo proposto por Blinn em 1988, o algoritmo de sombras planas. Este algoritmo permite a geração da sombra de um objeto através da projeção dos vértices que o constituem, para o plano base (ou chão) da cena. Porém, este algoritmo apresenta algumas limitações, como é o facto de apenas permitir a geração de sombras em superfícies planares, bem como ignorar as sombras que um objeto faz sobre si próprio. Tem, no entanto, aspetos positivos como a capacidade de gerar sombras livres de efeito de escada (*aliasing*).

Para solucionar as limitações do algoritmo proposto por Blinn, superfícies curvas podem ser aproximadas por faces planas, o que acaba por permitir a aplicação do algoritmo a vários tipos de superfícies. No que respeita à impossibilidade na geração de sombras que um objeto faz sobre si próprio, pode dizer-se que este problema pode ser resolvido analisando cada uma das suas facetas e determinar quais das restantes facetas lhe fazem sombra.

Através das soluções apresentadas é possível obter um algoritmo capaz de calcular sombras para vários tipos de cenas. No entanto, o seu desempenho computacional vai estar dependente da complexidade da cena. Por essa razão é feita a paralelização em GPU do algoritmo, de modo a proporcionar uma maior capacidade de resposta em cenas de maior complexidade.

Foram por isso projetadas duas versões do mesmo algoritmo, sequencial e paralela, através das quais é possível gerar sombras em cenas de diferentes características. Constatou-se, através dos resultados obtidos, que a versão paralela em GPU garante uma maior desempenho e capacidade de resposta em cenas de maior complexidade do que a versão sequencial.

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Esta dissertação enquadra-se na área científica da computação gráfica, mais concretamente no tópico de geração automática de sombras em cenas sintéticas 3D.

Uma sombra não é mais do que a silhueta que um objeto projeta numa superfície quando se interpõe entre a dita superfície e a fonte de luz. As sombras são por isso algo natural que contribuem para a perceção da geometria, da dimensão e da posição dos objetos que nos rodeiam. A sua omnipresença torna-as despercebidas aos seres humanos, mas as sombras são fundamentais na representação virtual de objetos e cenas, pois a sua ausência é imediatamente notada.

Uma zona sombreada não é necessariamente uma zona escura de cor preta. Dependendo do número de fontes de luz existentes, é possível existirem zonas de sombra mais escuras que outras. Nomeadamente, as zonas de sombra comuns a várias fontes de luz vão ser mais escuras que as zonas de sombra provocadas por apenas uma só fonte de luz.

As primeiras técnicas de geração de sombras surgiram numa altura em que não eram computacionalmente exequíveis devido essencialmente às limitações de hardware. No entanto, com a evolução da capacidade computacional das placas gráficas, passou a ser possível a implementação de algoritmos capazes de gerar sombras em tempo real para ambientes 3D, o que antes era impensável.

1.1 Motivação

Existe um grande número de algoritmos que permitem gerar sombras em ambientes 3D, sendo o mapa de sombras e o volume de sombras os dois algoritmos mais utilizados. Nesta dissertação, pretende-se explorar uma técnica de geração de sombras já existente, a qual consiste na obtenção de sombras através da projeção direta de cada objeto sobre os restantes objetos existentes na cena, obliterando assim o problema de *aliasing* que tem permanecido ao longo dos anos na maioria dos algoritmos existentes na literatura.

Trata-se do algoritmo de sombras planares [2] proposto por Blinn em 1988. Apesar da sua simplicidade, acabou por deixar de ser utilizado devido à popularidade entretanto adquirida por outros algoritmos como o mapa de sombras [5] e o volume de sombras [6], que permitem a geração de sombras sem exigências excessivas de recursos computacionais. A esta situaçã não é estranho o facto de a complexidade computacional do algoritmo de geração de sombras por projeção direta ser $\mathcal{O}(n^2)$.

Pretende-se assim, nesta dissertação, estender o algoritmo de Blinn a quaisquer tipos de geometria, eliminando na medida do possível o efeito de escada (*aliasing*) e, simultanemente, introduzir as auto-sombras.

1.2 Definição do Problema e Objetivos

O objetivo principal desta tese consiste na criação de um novo algoritmo de sombras, tendo como base o já existente algoritmo de sombras planares [2]. Algoritmo esse que tem como principal característica a obtenção de sombras livres de *aliasing*, problema que se encontra na maioria dos algoritmos de sombras atuais.

Pretende-se que o algoritmo a ser desenvolvido remova as limitações existentes no algoritmo em que se baseia. Tendo assim como principais características a possibilidade da sombra de um objeto ser projetada não só sobre superfícies planas mas também sobre superfícies curvas, bem como o facto de contemplar a sombra que um objeto provoca sobre si mesmo.

Uma das principais razões que levaram a baixa utilização do algoritmo de sombras planares, reside no facto deste se basear na geometria dos objetos representados

para obtenção das suas sombras. O que vai torna o algoritmo inaplicável a cenas de elevada complexidade. Pretende-se por isso tirar partido da utilização da placa gráfica na realização dos cálculos envolvidos, de modo a garantir uma maior escalabilidade do algoritmo.

1.3 Organização da Dissertação

Esta dissertação encontra-se estruturada da seguinte forma. No Capítulo 2, apresenta-se o estado da arte relativo aos algoritmos de sombras existentes. É feita uma distinção entre os tipos de sombras existentes, uma retrospetiva dos algoritmos de sombra existentes, métodos de iluminação global que permitem igualmente a obtenção de sombras e, por fim, os aspetos onde é possível tirar partido na utilização da GPU nos algoritmos de sombras expostos.

No Capítulo 3, é exposto o algoritmo de sombras baseadas em projeção direta. Em primeiro lugar é feita uma abordagem sobre a contribuição de outros algoritmos na elaboração do algoritmo desenvolvido nesta dissertação. De seguida são enunciadas as principais funcionalidades que foram implementadas, seguidas da exposição passo-a-passo do algoritmo de sombras baseadas em projeção direta. São também explicadas as decisões tomadas na implementação do algoritmo, bem como as bibliotecas auxiliares utilizadas para atingir o resultado pretendido. Introduz-se ainda a tecnologia CUDA e é explicada a sua utilização na paralelização do algoritmo de modo a melhorar o desempenho do mesmo. Por fim, apresentam-se os resultados visuais e quantitativos obtidos pelo algoritmo de geração de sombras baseadas em projeção direta.

A presente dissertação encerra no Capítulo 4, no qual se apresentam as conclusões mais relevantes do trabalho desenvolvido. Aflora-se ainda o trabalho futuro, bem como aspetos que poderão introduzir melhoramentos no algoritmo descrito nesta dissertação.

Refira-se também que este algoritmo poderá dar origem a um artigo científico a publicar nas atas de uma conferência internacional.

Capítulo 2

Revisão Breve da Literatura

Este capítulo revê de forma breve as técnicas de geração de sombras mais importantes descritas na literatura. Aparte as sombras geradas pelas técnicas de iluminação global (por exemplo, traçagem de raios ou *ray tracing*), há várias técnicas de geração de sombras em computação gráfica, nomeadamente: sombras planares (*planar shadows*), mapas de sombra (*shadow maps*), volumes de sombra (*shadow volumes*).

2.1 Definições e Tipos de Sombras

Uma sombra é uma superfície privada de luz que resulta da interposição de um corpo opaco entre ela e a fonte de luz. A sua extensão vai depender da distância entre a superfície e a fonte de luz, bem como da distância da fonte de luz ao corpo opaco que a bloqueia.

Qualquer sombra pode ser dividida em duas componentes: umbra e penumbra. A umbra diz respeito à zona que se encontra totalmente obstruída em relação à fonte de luz. A penumbra refere-se à transição gradual de uma zona iluminada para uma zona de umbra.

Em termos genéricos, os algoritmos de geração de sombras encontram-se divididos em duas categorias: sombras vincadas (*hard shadows*) e sombras esbatidas (*soft shadows*). As sombras vincadas são constituídas por apenas uma componente, a umbra. Por outro lado, as sombras esbatidas apresentam quer a umbra quer a penumbra. A diferença entre elas está essencialmente relacionada com a área da fonte de luz presente na

cena. Se for uma fonte de luz composta por apenas um ponto emissor vai dar origem a sombras vincadas. Mas se a fonte de luz for composta por uma área emissora vai originar sombras esbatidas.

Quando uma cena é iluminada por uma fonte de luz formada por apenas um ponto emissor, os objetos nela representados são completamente iluminados ou encontram-se completamente na sombra [7]. Ou seja, é facilmente visível a fronteira entre uma zona sombreada e uma zona iluminada. Esta é a característica mais fácil de identificar nos algoritmos de sombras vincadas. No entanto, existem métodos que permitem simular sombras esbatidas com base em sombras vincadas, como será referido nas próximas secções.

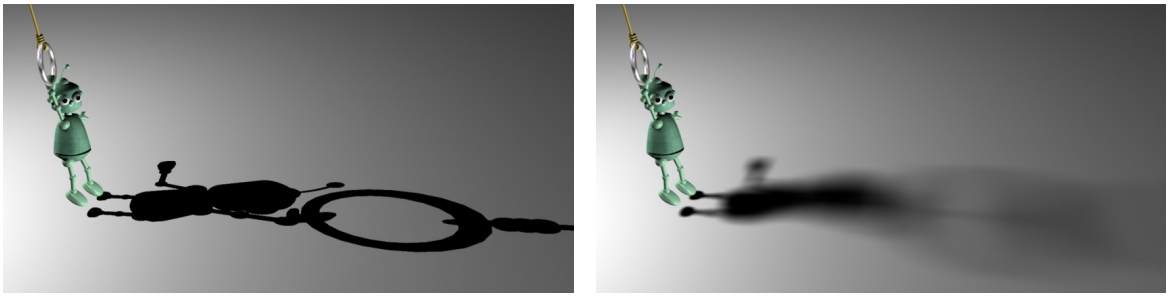


Figura 2.1: Sombras vincadas e sombras esbatidas [1]

No caso de uma cena ser iluminada por uma fonte de luz composta por um área emissora, a fronteira que divide uma zona iluminada de uma zona sombreada já não é tão percetível. Isto é devido à área da fonte de luz, a qual vai originar três zonas distintas, zonas iluminadas, zonas de umbra e zonas de penumbra [7]. Consideram-se por isso mais realistas as sombras resultantes de fontes de luz compostas por uma área emissora.

2.2 Sombras Planares

O algoritmo de sombras planares proposto por Blinn [2], consiste em obter sombras através da projeção dos vértices de um objeto num plano. Este algoritmo foi bastante popular durante vários anos pois é relativamente simples de implementar e os resultados obtidos são sombras vincadas projetadas no plano [7].

2.2.1 Algoritmo

O algoritmo consiste em transformar uma forma tridimensional numa forma bidimensional e desenhá-la num plano. Este processo aparece ilustrado na Figura 2.2. Em primeiro lugar desenha-se o objeto e depois a sua sombra. O desenho da sua sombra é feito aplicando uma matriz de projeção específica (veja-se próximo capítulo). Quando se desenha a sombra de objeto deve ser utilizada a cor preta de modo a assemelhar-se a um sombra. Caso contrário, são usadas as cores originalmente definidas o que vai fazer com que o plano de projeção se assemelhe a um espelho.

Na Figura 2.2 é possível ver o processo referido anteriormente. Onde a posição da fonte de luz é representada por L, o ponto pertencente ao objeto por P e o mesmo ponto projetado no plano representado por S.

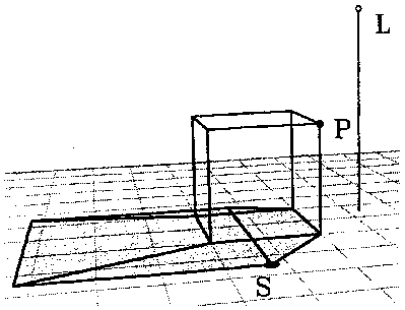


Figura 2.2: Projeção de um objeto num plano [2].

2.2.2 Problemas do algoritmo

Como se sabe, este algoritmo apresenta num conjunto de limitações, nomeadamente:

- A grande desvantagem deste algoritmo está no facto de apenas ser aplicável a superfícies planas. Esta limitação impede o realismo de cenas mais complexas, pois como se sabe as sombras são projetadas em todo tipo de superfícies, independentemente da geometria da superfície.
- Outra limitação reside no problema de inversão de sombra no plano de projeção, o que acontece quando a fonte de luz se interpõe entre o objeto e o plano que vai receber a sombra (Figura 2.3).
- Outro problema que este algoritmo apresenta é que as sombras resultantes da sua aplicação podem prolongar-se infinitamente ao longo do plano onde se encontram.

Isto porque o próprio plano se estende infinitamente. Este aspeto pode ser um problema quando for necessário restringir a sombra a uma parte do plano, como por exemplo uma face.

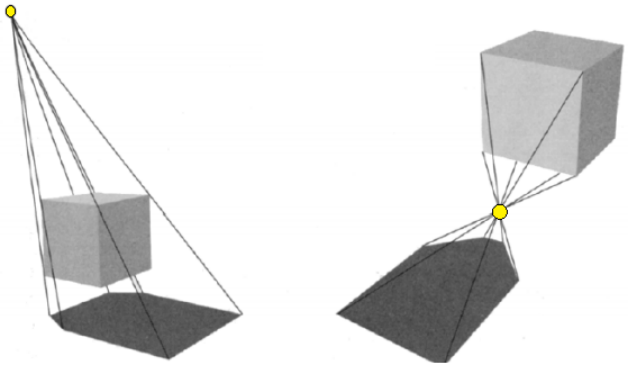


Figura 2.3: Resultado da projeção de acordo com a fonte de luz [3].

2.2.3 Algoritmos Baseados em Sombras Planares

O algoritmo de sombras planares original de Blinn [2] permite gerar unicamente sombras vincadas. No entanto, a partir do método proposto por Heckbert e Herf [8], é também possível gerar sombras esbatidas. Isto consegue-se através da simulação de uma área emissora de luz através de várias luzes pontuais. O método referido permite ainda armazenar a sombra numa textura e desenhá-la múltiplas vezes.

Outro método que se baseia no algoritmo de sombras planares para obter sombras esbatidas é o proposto por Gooch [9]. Neste caso, o efeito de sombras esbatidas é conseguido através do afastamento e aproximação do plano onde vai ser projetada a sombra, simulando assim o efeito de umbra e penumbra.

Em ambos os métodos referidos de geração de sombras esbatidos referidos anteriormente, é necessário projetar cada objeto tantas vezes quanto o número de faces de um objeto, o que resulta num decréscimo do frame rate.

2.3 Mapa de Sombras

O mapa de sombras (*shadow maps*) é uma técnica que permite gerar sombras em tempo real para ambientes tridimensionais. Foi proposta por Lance Williams [5] em 1978 e permite a projeção de sombras diretamente em superfícies curvas.

É um algoritmo que garante a obtenção de sombras de forma bastante célere, pois baseia-se na comparação de valores de profundidade obtidos através de *z-buffering*, ou seja, obtidas diretamente a partir do hardware gráfico.

2.3.1 Algoritmo

O algoritmo baseado em mapa de sombras pode dividir-se em dois passos. No primeiro passo, renderiza-se a cena vista da posição da fonte de luz e armazenados numa textura os valores de profundidade das áreas visíveis. Estes valores são obtidos com recurso a um algoritmo embutido no hardware gráfico, z-buffer, e que representam a distância entre a fonte de luz e as áreas visíveis (Figura 2.5).

Figura 2.4: Mapa de sombras

No segundo passo é novamente renderizada a cena, mas vista da posição do observador. A posição de cada pixel é convertida para o sistema de coordenadas da fonte de luz e comparado o valor de profundidade atual com o valor de profundidade armazenado na textura no passo anterior.

Uma determinada porção da cena encontra-se na sombra se o valor de profundidade obtido no primeiro passo for maior que o valor de profundidade obtido no segundo. Caso contrário, a porção da cena é iluminada.

2.3.2 Problemas do algoritmo

O principal problema desta técnica está relacionado essencialmente com o *aliasing* [10], mais concretamente o *perspective aliasing* e *projection aliasing*.

O primeiro problema resulta do primeiro passo do algoritmo, quando a cena é vista em perspetiva da posição da fonte de luz. Neste caso, os objetos mais próximos da fonte de luz apresentam maior dimensão que os objetos mais distantes. Isto origina a necessidade de diminuir os objetos mais próximos (provocando subamostragem) e aumentar os objetos mais distantes (dando origem a sobreamostragem).

O segundo, *projection aliasing*, embora mais raro resulta em artefactos na sombra em superfícies quase paralelas à direção da fonte de luz.

Figura 2.5: *Perspective Aliasing*

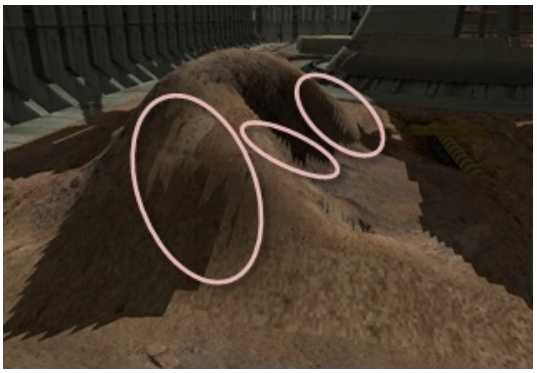


Figura 2.6: *Projective Aliasing*

Outros problemas existentes nos algoritmos baseados em mapa de sombras devem-se à limitação (a nível de hardware) da resolução máxima da textura a ser utilizada, bem como à possibilidade de ocorrência de erros aquando da comparação dos valores de profundidade obtidos em ambos os passos do algoritmo, que podem resultar em sombras incorretas (*shadow acne*).

Devido aos problemas existentes, o mapa de sombras serviu de base a inúmeros algoritmos de simulação de sombras que se focaram essencialmente em corrigir os problemas/falhas nele presentes.

2.3.3 Algoritmos Baseados no Mapa de Sombras

Os principais algoritmos que derivam do algoritmo de mapa de sombras e que podem ser vistos como revisões do método original são o Perspective Shadow Maps (PSM),

Trapezoidal Shadow Maps (TSM), Parallel-Split Shadow Maps (PSSM).

No algoritmo PSM [11], o mapa de sombras só é criado após o conteúdo da pirâmide truncada (*view frustum*) ser convertido para um cubo cuja gama de valores está entre -1 e 1. Após esta conversão, os objetos vão apresentar as dimensões finais com que vão ser representados. Esta conversão consiste em atribuir o efeito de profundidade à cena, onde os objetos mais próximos do observador vão parecer maiores do que os objetos mais distantes, mesmo que ambos tenham as mesmas dimensões. Reduz-se assim o *perspective aliasing* com a criação do mapa de sombras após a referida conversão.

O TSM [12] é semelhante ao PSM. No entanto, a conversão da pirâmide truncada (*view frustum*) é feita com base nos oito vértices que a definem, em vez da conversão direta do seu conteúdo. Neste método é então calculado um trapezóide com base nos oito vértices da pirâmide truncada que define a região visível. Em seguida são aplicadas ao trapezóide operações de rotação, translação, alinhamento, redimensionamento e normalização. Por fim é obtido o mapa de sombras que por sua vez resulta em sombras livres de *perspective aliasing*.

O PSSM [13] consiste em obter, para cada fonte de luz, vários mapas de sombra de forma hierárquica. É feita a divisão de pirâmide truncada que define a região visível (*view frustum*) com recurso a planos paralelos ao plano de projeção. Em seguida são obtidos os mapas de sombra de cada região, que por serem relativos a áreas mais reduzidas vão resultar na obtenção de sombras com maior detalhe. Esta versão do algoritmo baseado em mapa de sombras permite assim tratar o problema do *perspective aliasing*.

2.4 Volume de Sombras

O volume de sombras (*shadow volumes*) é outra técnica utilizada em ambientes gráficos tridimensionais para adicionar sombras a cenas renderizadas. Foi proposta por Franklin Crow [6] como sendo a geometria que descreve a forma tridimensional da região não iluminada pela fonte de luz. Neste algoritmo, um "volume de sombra" é utilizado para separar zonas que estão na sombra e zonas que são iluminadas.

2.4.1 Algoritmo

O algoritmo baseado em volume de sombras consiste em determinar a silhueta formada pela parte dos objetos visível da posição do observador e em seguida formar uma pirâmide com origem na posição da fonte de luz, que passe pelas arestas da silhueta obtida e se estenda infinitamente.

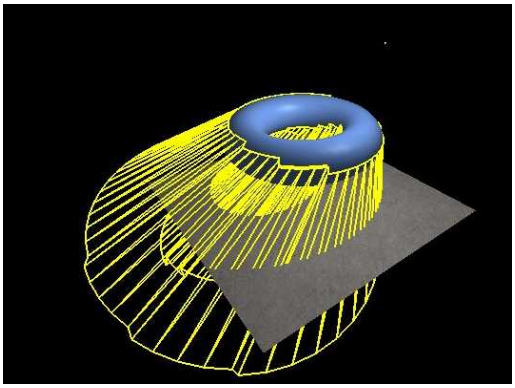


Figura 2.7: Representação da extrusão da silhueta de um objeto.

O volume que representa a zona não iluminada é formado pela extrusão da silhueta, com a mesma direção da fonte de luz, até atingir outro objeto (Figura 2.7). O conceito aqui presente é que todos os objetos contidos neste volume estão na sombra, enquanto objetos fora do mesmo são iluminados [7].

Com base na proposta original do algoritmo baseado no volume de sombras, Tim Heidmann implementou o mesmo algoritmo tirando partido do uso do *stencil buffer*. Isto permitiu efetuar a renderização de sombras de forma mais célere e possibilitou a sua utilização em aplicações gráficas em tempo real. Existem três versões do algoritmo baseado no volume de sombras que utilizam o *stencil buffer*: depth pass, depth fail e exclusive-or. O que as distingue é a forma como é utilizado o *stencil buffer*, que funciona como filtro na renderização da parte relevante da cena. Esta seleção da parte relevante da cena é feita com base nos valores de profundidade da cena.

Os aspetos positivos apresentados neste algoritmo devem-se ao facto de ser totalmente genérico e poder ser aplicado ao desejado número de fontes de luz e objetos. Permite também gerar sombras sobre qualquer tipo de geometria e sobre o próprio objeto que dá origem à sombra (*self-shadow*) [14].

2.4.2 Problemas do algoritmo

No que diz respeito a aspetos negativos neste algoritmo, refira-se que este algoritmo apenas gera sombras vincadas e consume bastantes recursos computacionais. Este último aspeto deve-se ao facto do algoritmo ser essencialmente geométrico [14], sendo que a redução do desempenho tende a ser mais notório quando aumenta o número de polígonos para o qual são geradas sombras.

Outro aspeto negativo deste algoritmo está relacionado com a dificuldade na identificação das arestas que compõem a silhueta do objeto que vai causar a sombra, o que pode originar sombras com falhas ou sombras em locais indesejados.

2.5 Notas Suplementares

Existem outros dois aspetos a ter em conta no que diz respeito à geração de sombras através de algoritmos: iluminação global e sombras em GPU.

Os algoritmos de iluminação global, como a traçagem de raios e radiosidade, têm como finalidade a simulação realista da luz em cenas tridimensionais. Como a sombra é uma consequência da existência de iluminação, os algoritmos referidos contemplam também a geração de sombras nas cenas onde são aplicados.

Ambos os métodos de iluminação global permitem gerar sombras esbatidas com bastante realismo. No entanto são métodos com grande custo computacional. Devido a este facto, começaram por ser mais utilizadas na renderização de imagem estática. No entanto, em circunstâncias particulares, já são usadas na renderização em tempo real, com a sua inclusão nalguns motores de jogo bastante conhecidos (CryENGINE® 3 e Frostbite® 2).

Os algoritmos de geração de sombras abordados nas secções anteriores apresentam todos algumas limitações, no entanto através da utilização da GPU é possível pelo menos atenuar algumas delas.

Dos algoritmos abordados, o algoritmo de sombras planares é o que menos foi explorado e daí não existirem abordagens do mesmo em GPU. Por se tratar de um algoritmo baseado em geometria, seria possível obter resultados bastante mais expeditos, até para cenas mais complexas em GPU.

O algoritmo original baseado em mapa de sombras, é em termos de desempenho bastante rápido e capaz de gerar sombras para cenas complexas sem haver necessidade de utilizar paralelismo. No entanto, o paralelismo em GPU pode ser uma mais valia para solucionar os problemas de *aliasing*, nomeadamente com a aplicação da técnica Percentage Close Filtering [15]. Esta técnica permite atenuar zonas de *aliasing* através da sua suavização, efetuando para cada pixel o cálculo da média dos valores de profundidade dos vizinhos, obviamente isto causa problemas de desempenho, resolvidos através do paralelismo em GPU.

Os algoritmos baseados em volumes de sombras são essencialmente geométricos pelo que tendem a sobrecarregar a CPU na obtenção das arestas que formam a silhueta dos objetos que provocam sombra na cena. Através do paralelismo este passo do algoritmo pode ser feito na GPU, tornando assim o algoritmo mais rápido na obtenção de sombras para cenas de grandes dimensões [16].

Em suma, dos três algoritmos de sombra abordados, a utilização do paralelismo em GPU torna-se útil para acelerar cálculos geométricos na geração de sombras planares e volumes de sombra, bem como para remover o *aliasing* do mapa de sombras.

Capítulo 3

Geração de Sombras por Projeção Direta

Neste capítulo é exposto um algoritmo de sombras baseadas em projeções diretas ("direct shadow projection", do inglês, ou DSP). Este algoritmo resolve os problemas fundamentais do algoritmo original de Blinn, nomeadamente: a impossibilidade de ser aplicado a superfícies não planas, a existência de sombras inversas resultantes do posicionamento da fonte de luz entre o objeto e o plano de projeção, a possibilidade da sombra se estender infinitamente sobre o plano de projeção e a ausência de auto-sombras.

3.1 Descrição do Algoritmo

O algoritmo de sombras baseadas em projeções permite gerar sombras de objetos recorrendo apenas à projeção desse objeto no plano. Tem por isso como base o método de sombras planares [2]. Trata-se de um algoritmo que não é difícil de implementar visto que se baseia apenas em projeções, pelo que não há lugar ao surgimento de *aliasing* nas sombras. No entanto, a técnica de sombras planares tem sido utilizada com alguma reserva devido às limitações resultantes de apenas funcionar em superfícies planas e de ignorar as sombras que os objetos fazem sobre si próprios.

Dito de outra forma, a grande limitação da técnica de sombras planares está então relacionada com a impossibilidade de gerar sombras sobre superfícies não planas. No entanto esta limitação pode ser mitigada, visto que uma superfície curva pode

ser aproximada por várias facetas. Na verdade, uma superfície pode ser suavizada aumentando a quantidade e/ou diminuindo a dimensão das facetas que aproximam a superfície, como se ilustra na Figura 3.1. Com isto é então possível utilizar a técnica de sombras planares para obter sombras sobre vários tipos de geometrias.

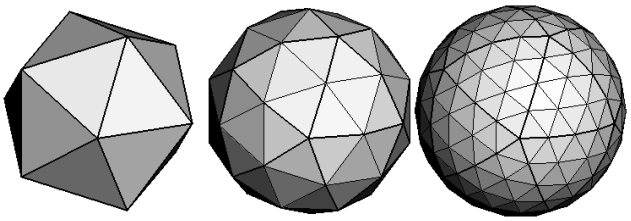


Figura 3.1: Sólidos Platónicos [4].

Para obter uma cena com sombras realistas é essencial ter também em conta a sombra que um objeto provoca sobre si próprio. Assim, para cada face de um objeto, torna-se necessário renderizar:

- A própria face;
- A sombra que a face faz sobre os restantes objetos (faces) da cena;
- A sombra que os restantes objetos (faces) fazem sobre aquela face.

Outro aspeto a ter em conta é a utilização de triângulos como elemento base dos objetos da cena, o que significa que cada face de um objeto irá ser representada por um triângulo.

As sombras inversas resultantes do posicionamento da fonte de luz entre o objeto e o plano de projeção podem ser removidas bastando para isso ignorar os objetos que se encontram atrás do plano de projeção. Visto que estes não são relevantes para a sombra a ser projetada sobre o plano.

Como os objetos representados numa cena são constituídos por facetas e não por planos, existe a necessidade de evitar que a sombra se estenda infinitamente ao longo do plano de projeção, o que pode ser feito através do uso do stencil buffer.

Em termos genéricos, os triângulos da cena são divididos em dois grupos, os que estão totalmente na sombra e os que estão parcialmente na sombra. Para determinar se um triângulo se encontra totalmente ou parcialmente na sombra, basta fazer o produto

escalar entre o vetor normal do triângulo e o vetor que define a direção da fonte de luz, o que não é mais do que a essência do algoritmo de *backface culling*.

Caso se verifique que o triângulo está totalmente na sombra, é desenhada a sombra sobre a totalidade do triângulo, não havendo por isso lugar a cálculos adicionais. Caso contrário, é necessário desenhar o triângulo e de seguida projetar sobre ele todos os restantes triângulos da cena que estejam à sua frente. Repetindo este processo para cada triângulo da cena, obtém-se a sombra final desejada, como se ilustra no Algoritmo 1.

Algoritmo 1 Geração de sombras via DSP

```
1:  for ∀T do
2:      if T ← visible then
3:          if T ← totally in shadow then
4:              Shadow T fully
5:          else
6:              Draw T
7:              Find set S of triangles in front of T
8:              Project triangles of S on T
9:          end if
10:     end if
11: end for
```

Explica-se de seguida, com mais pormenor, os passos do Algoritmo 1:

1. Carregar modelo OBJ em memória.
2. Ativar iluminação e *stencil test*.
3. Para cada um dos triângulos (**T**) existentes na cena:
   (a) Fazer *backface culling* do triângulo T relativamente à posição da câmara, para determinar se é visível ou está escondido por outros triângulos. Isto vai permitir acelerar o processo visto que o triângulo **T** pode logo à partida ser ignorado se estiver oculto. Esta verificação refere-se à linha 2 do Algoritmo 1.
      i. Caso não seja visível não há necessidade de efetuar os passos seguintes. Deve por isso ser tratado o próximo triângulo.

(b) Definir o tipo da função de teste *stencil* como *"GL_ALWAYS"*. O que vai permitir ao *stencil buffer* armazenar o estado de todos os pixeis que forem desenhados a seguir. Veja-se a secção 3.3 para mais detalhes sobre o *stencil buffer*.

(c) Fazer *backface culling* da posição fonte de luz em relação ao triângulo **T**, o que vai permitir tratar de diferente forma os triângulos totalmente na sombra e os triângulos parcialmente na sombra ou completamente iluminados. Esta verificação refere-se à linha 3 do Algoritmo 1.

i. Caso o triângulo **T** não seja visível da posição da fonte de luz, pode dizer-se que está totalmente na sombra. Este caso diz respeito a linha 4 do Algoritmo 1 e deve por isso ser tratado da seguinte forma:

A. Ativar funcionalidade "blend", de modo a poder alterar a cor definida para que se assemelhe a uma zona de sombra.

B. Desativar iluminação para que esta não interfira na cor obtida com a funcionalidade anterior.

C. Aplicar cor definida no OBJ, com valor de 0,5 no parâmetro referente ao "blend".

D. Aplicar textura ao triângulo caso exista.

E. Desenhar o triângulo **T**.

F. Ativar iluminação.

G. Desativar "BLEND".

ii. No caso do triângulo **T** ser visível da posição da fonte de luz, este pode estar parcialmente na sombra ou sem qualquer sombra. Dizem respeito a este caso as linhas 6,7 e 8 do Algoritmo 1 e deve ser feito o seguinte:

A. Aplicar textura ao triângulo **T**, caso exista.

B. Desenhar triângulo **T**.

C. Alterar a função de teste *stencil* para *"GL_EQUAL"*. De modo a que apenas seja possível modificar os pixeis relativos ao triângulo **T** já desenhado. Com esta alteração feita à função de teste *stencil* é possível fazer o recorte da sombra, evitando que esta se estenda para zonas não desejadas.

D. Guardar a normal do plano onde está contido o triângulo **T** e um dos seus vértices. Isto para poder construir a matriz de projeção no

passo seguinte.

E. Fazer *"PushMatrix"* de modo a poder repor mais tarde o estado atual da pilha de matrizes.

F. Alterar a matriz *modelview*, através da multiplicação da matriz M (Equação 3.1) pela matriz *modelview* definida. Isto vai permitir projetar automaticamente tudo o que for desenhado, sobre o triângulo **T** desenhado anteriormente. Ver secção 3.2 para mais detalhe na alteração da matriz *modelview*.

G. Ativar funcionalidade "blend", de modo a poder alterar a cor definida para que se assemelhe a uma zona de sombreada.

H. Desativar iluminação para não interferir na cor a ser atribuída à sombra.

I. Definir a cor preta com valor 0,5 no parâmetro referente ao "blend", para que os triângulos desenhados em seguida se assemelhem a uma sombra.

J. Calcular as sombras a serem projetadas sobre o triângulo **T**. Para isso é necessário recorrer à equação (3.4) para determinar exatamente quais dos restantes triângulos (**T'**) se encontram à frente do triângulo **T**. O que pode dar origem a uma das seguintes situações:

   - O triângulo **T'** encontra-se à frente do triângulo **T** e é desenhado como constituinte da sombra. Este caso encontra-se explicado de forma mais detalhada na subsecção 3.4.1.
   - O triângulo **T'** interseta o plano que contem o triângulo **T**, onde dois vértices estão à frente do plano e um atrás. Isto vai resultar em dois triângulos a serem desenhados como parte da sombra a ser projetada sobre **T**. Na subsecção 3.4.2 é dada uma explicado de forma mais detalhada sobre este caso.
   - O triângulo **T'** interseta o plano onde está contido o triângulo **T** e apenas um dos vértices de **T'** se encontra à frente do plano. O que vai resultar num novo triângulo a ser desenhado como sombra sobre o triângulo **T**. É explicado este caso com mais detalhe na subsecção 3.4.3.
   - Por fim, o triângulo **T'** encontra-se atrás do triângulo **T** e não vai

fazer sombra sobre o triângulo T.

K. Fazer "*PopMatrix*", de modo a repor a pilha de matrizes de volta ao estado inicial. Isto vai permitir repor a matriz *modelview* alterada anteriormente. Caso contrario a tudo o que fosse desenhado seria projetado para o ultimo triângulo tratado.

(d) Ativar iluminação para terminar a geração das sombras.

3.2 Projeção de Sombras no Plano

O algoritmo DSP baseia-se na projeção de toda a cena sobre cada uma das faces dos objetos existentes. Para isso é utilizada a matriz M (veja-se (3.1)), que vai ser combinada com a matriz *modelview* definida no OpenGL de modo a permitir fazer a projeção dos objetos desenhados de forma automática para o plano pretendido [17] [3]. Esta combinação de matrizes não é mais do que a multiplicação da matriz M pela matriz *modelview* definida no OpenGL, da qual resulta a seguinte matriz de projeção direta M.

$$M = \begin{bmatrix} l_x n_x + c & n_y l_x & n_z l_x & -l_x c - l_x d \\ n_x l_y & l_y n_y + c & n_z l_y & -l_y c - l_y d \\ n_x l_z & n_y l_z & l_z n_z + c & -l_z c - l_z d \\ n_x & n_y & n_z & -d \end{bmatrix} \tag{3.1}$$

Em que d e c dizem respeito aos valores produzidos pelas equações (3.2) e (3.3), respetivamente. Mais concretamente, d representa o produto interno dos vetores $\overrightarrow{l}$ e $\overrightarrow{n}$, em que $\overrightarrow{l}$, na matriz e equações referidas, diz respeito à posição da fonte de luz, $\overrightarrow{n}$ ao vetor normal do plano para onde vai ser efetuada a projeção, sendo $\mathbf{e} = (e_x, e_y, e_z)$ um ponto pertencente ao mesmo plano.

$$d = n_x l_x + n_y l_y + n_z l_z \tag{3.2}$$

$$c = e_x l_x + e_y l_y + e_z l_z - d \tag{3.3}$$

Como a matriz M é formada com base no plano que contem o triângulo para onde se deseja fazer a projeção, a aplicação da matriz M necessita ser feita para cada um dos triângulos que se encontram parcialmente na sombra.

3.3 Stencil Buffer

Para além dos *buffers* mais conhecidos como é o caso do *buffer* de cor e do *buffer* de profundidade, existe um outro que também é suportado pelo hardware gráfico, o stencil buffer. O stencil buffer é utilizado para guardar informação dos pixeis, o que permite efetuar a comparação entre diferentes estados dos mesmos pixeis. Isto permite desenhar apenas os pixeis que passem no teste "stencil" o que resulta na possibilidade de confinar a área a ser renderizada.

No algoritmo exposto na secção anterior o *stencil buffer* é utilizado para delimitar a sombra projetada sobre a face. Isto é necessário porque a projeção conseguida através da matriz (3.1) é efetuada para o plano que contém a face, logo o objeto projetado (sombra) pode estender-se ao longo de todo o plano. No entanto a parte relevante do mesmo é apenas a que se encontra sobre a face e não sobre a totalidade do plano que a contem.

3.4 Comparação de Triângulos

No algoritmo DSP, é necessário comparar entre si todos os triângulos existentes na cena de modo a ignorar os que não fazem sombra sobre nenhum dos restantes triângulos. Se de um triângulo for obtida a equação do plano que o contém e de outro triângulo se obtiverem os vértices que o definem, é possível determinar se cada um dos vértices se encontra à frente ou atrás do plano. Foi para isso utilizada a equação geral do plano, dada por:

$$ax + by + cz + d = 0 \tag{3.4}$$

onde, a, b e c representam as componentes do vetor normal ao plano, e em que $d = -ax - by - cz$.

Após ter sido obtida a equação geral do plano, esta é utilizada para determinar a posição de um ponto em relação ao plano por ela definido, ou seja, se um ponto se encontra atrás ou à frente do plano, bastando para isso substituir na equação (3.4) os valores a, b e c pelas coordenadas do ponto que se quer testar. Caso o resultado dê zero o ponto está sobre o plano, se o resultado tiver sinal positivo o ponto está à frente do plano e se o resultado tiver sinal negativo o ponto encontra-se atrás do plano.

Sabendo a posição de um ponto em relação ao plano, é possível aplicar o mesmo processo na comparação de dois triângulos e determinar se um triângulo está à frente, atrás ou interseta outro triângulo. Isto é conseguido, verificando a posição de cada um dos vértices de um triângulo em relação ao plano onde está contido o outro triângulo.

No contexto do algoritmo isto vai permitir saber o número máximo de triângulos que pode fazer sombra sobre cada um dos triângulos da cena. Sendo para isso necessário repetir o processo n^2 vezes.

Esta comparação pode resultar em três cenários distintos: os três vértices estão à frente ou atrás do plano, dois vértices à frente do plano e apenas um atrás ou só um vértice à frente do plano e dois atrás.

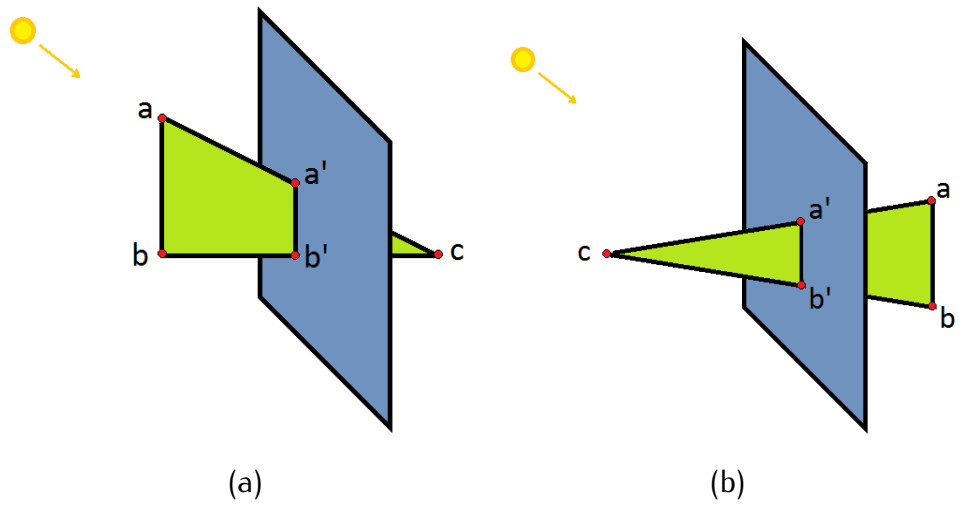


Figura 3.2: Exemplo de interseção entre um triângulo e um plano.

3.4.1 Três vértices à frente ou atrás do plano

Este é o caso mais comum e ocorre essencialmente quando se comparam faces de diferentes objetos. Caso o resultado da aplicação da equação (3.4) para os três vértices for positivo significa que todos os vértices estão à frente do plano, logo vai ser desenhada a sombra correspondente sobre a face em causa.

Caso o resultado da equação para os três vértices seja negativo, não irá ser desenhada nenhuma sombra sobre a referida face.

3.4.2 Dois vértices à frente do plano e um atrás

Um dos casos particulares é a existência de uma interseção entre uma face e o plano que contém a outra face a ser comparada. Isto vai resultar em dois dos vértices estarem à frente do plano e um dos vértices estar atrás do plano.

Tendo em conta que cada face é representada por um triângulo, a interseção vai ocorrer em dois pontos apenas. Isto porque só duas das arestas do triângulo vão intersetar o plano, como se exemplifica na Figura 3.2(a).

Com esta interseção, a parte do triângulo que se encontra à frente do plano vai ser um quadrilátero constituído pelos vértices *a, b, a' e b'* representados da Figura 3.2(a). Como já referido anteriormente as faces tratadas no algoritmo DSP são apenas triângulos, logo o quadrilátero obtido na interseção anterior vai ter que ser transformado em dois triângulos e de seguida desenhada a sombra causada pelos mesmos.

3.4.3 Um vértice à frente do plano e dois atrás

Outro caso particular é quando apenas um dos vértices da face (triângulo) está à frente do plano e os restantes dois vértices se encontram atrás do plano. Este caso é bastante semelhante ao explicado anteriormente, pois a interseção vai originar dois novos pontos. Porém neste caso a parte do triângulo que se encontra à frente do plano vai ser igualmente um triângulo, como se exemplifica Figura 3.2(b).

Assim o triângulo a causar a sombra no plano passa a ser formado pelos vértices *c, a' e b'*. Com este ajuste é possível tratar apenas a parte da face que está à frente do plano, evitando o surgimento de projeções inversas que resultariam em sombras erradas.

3.5 Bibliotecas Auxiliares

Na implementação do algoritmo DSP foram utilizadas algumas bibliotecas auxiliares de modo a adicionar funcionalidades que contribuíssem para obter um resultado final que fosse visualmente realístico. Para além do algoritmo que permita obter sombras corretamente, existe também a possibilidade de carregar cenas realistas com alguma complexidade.

Como o algoritmo é independente dos objetos aos quais é aplicado, foi utilizada uma

biblioteca auxiliar para carregar modelos OBJ de modo a poder testar várias cenas sem a necessidade de as definir manualmente. A biblioteca utilizada foi uma versão melhorada da biblioteca GLM originalmente desenvolvida por Nate Robbins [18] [19]. Com esta biblioteca é possível carregar modelos .OBJ complexos e realistas, havendo ainda a possibilidade de aplicar texturas às cenas definidas.

Para possibilitar a utilização de texturas com extensão .JPEG e .PNG é necessário utilizar as bibliotecas auxiliares "JpegLib"e "Pnglib", respetivamente. Apenas estes dois formatos são usados na texturização pois são os mais comuns, permitindo ainda a existência de transparência (.PNG).

3.6 Paralelização do Algoritmo

O algoritmo foi inicialmente desenvolvido de forma sequencial. No entanto, devido ao elevado número de triângulos e projeções necessárias para obter uma cena com sombras foi necessário tornar o algoritmo mais eficiente. Para tal foi desenvolvida uma versão paralela em GPU de modo a tirar partido da placa gráfica para fazer todos os cálculos necessários. A versão paralela foi desenvolvida em CUDA [20], que é uma plataforma de computação paralela e de modelos de programação que permitem obter um aumento significativo a nível de desempenho através do uso da placa gráfica.

No algoritmo DSP, como já referido anteriormente, é necessário comparar cada uma das faces com todas as restantes de modo a isolar a parte da sombra relevante. Esta comparação pode tornar o algoritmo demasiado moroso pois existem várias equações e matrizes envolvidas nos cálculos geométricos. Com a versão paralela do algoritmo pretende-se acelerar o mesmo, efetuando em cada *thread* a comparação de apenas dois triângulos.

3.6.1 Modelo de Programação CUDA

A paralelização em GPU baseia-se na existência em simultâneo de inúmeras *threads*, umas em execução e outras à espera da sua vez de serem executadas. Uma *thread* é a entidade ativa de uma GPU, pois é nela que vai ser executado o código paralelizado. Apesar do nome ser igual, as *threads* executadas na GPU diferem bastante das executadas no contexto do processador (CPU). A principal característica que distingue

uma *thread* GPU de uma *thread* CPU é que as *threads* no contexto de GPU executam todas o mesmo código, variando apenas os valores/dados por elas tratados. Esta característica denomina-se *Single Instruction Multiple Threads* (SIMT). No contexto CPU o código executado por cada *thread* é independente das restantes.

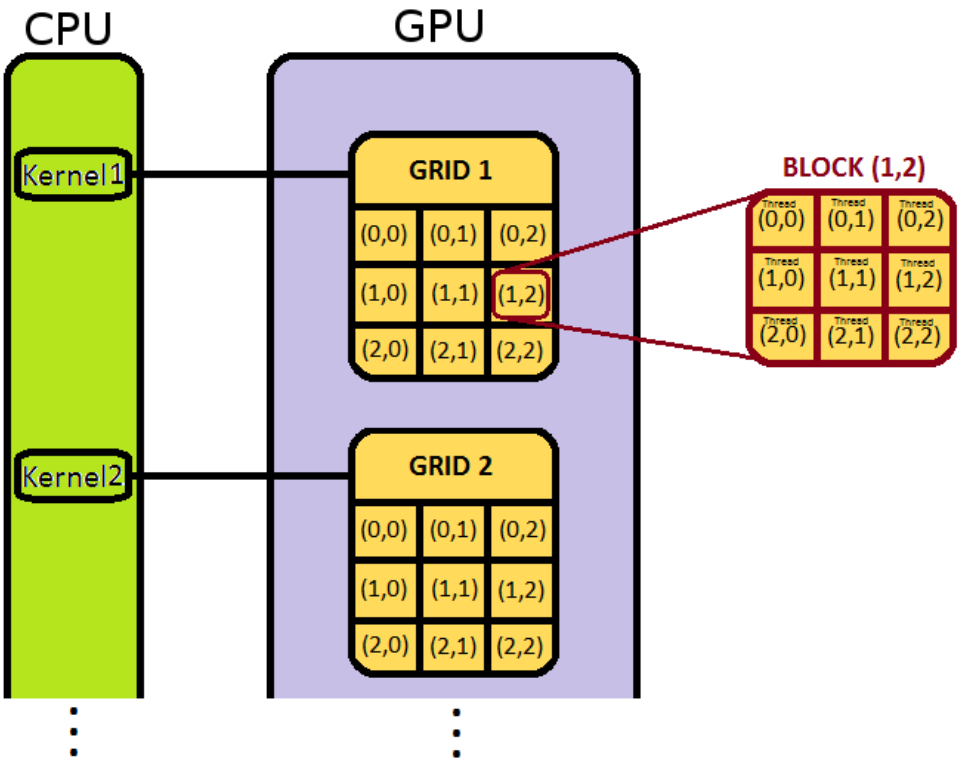


Figura 3.3: Esquema de funcionamento do CUDA

O código executado numa *thread* pela GPU é designado por *kernel*. Em termos de código, um *kernel* aparenta-se com uma função, à exceção de ser precedido pela palavra-chave *__global__* e de não permitir qualquer tipo de valor de retorno.

Como representado na Figura 3.3, GPU ou *device*, como é também denominado neste contexto, é constituída por *grids*. As *grids* por sua vez são formadas por um conjunto de *blocks*. E é nestes últimos que se encontram as *threads* onde vai ser executado o código do *kernel*.

3.6.2 Redesign do Algoritmo

Antes de proceder à paralelização do algoritmo é necessário efetuar algumas alterações na forma como o algoritmo se encontra estruturado na versão sequencial. Alterações

essas que consistem na utilização de Vertex Buffer Objects (VBOs) para renderização da cena, em vez do modo imediato utilizado na versão sequencial. A utilização de VBOs na versão paralela do algoritmo DSP deve-se principalmente à sua fácil integração com o CUDA, para além de proporcionar o armazenamento dos dados diretamente na memória gráfica.

A utilização de VBOs para renderização da cena, em detrimento do modo imediato, é necessário serem feitas algumas mudanças na ordem dos passos do algoritmo. Nomeadamente o passo em que os triângulos da cena são comparados entre si. Na versão sequencial era feita a comparação de cada triângulo com todos os restantes e desenhadas as sombras sobre esse triângulo. Com a utilização de VBOs é necessário, em primeiro lugar, comparar todos os triângulos entre si e só depois desenhar as sombras resultantes da comparação efetuada.

No entanto, para fazer uma correta utilização dos VBOs na renderização das sombras da cena, é necessário saber o tamanho exato do VBO aquando da sua criação. Isto porque, não é possível alterar o tamanho do VBO e preservar os dados nele contidos. No caso do algoritmo DSP, o VBO é utilizado apenas para armazenar os vértices dos triângulos que compõem as sombras da cena. Como cada triângulo pode ficar na sombra de todos ou de nenhum dos restantes triângulos da cena, é necessário determinar exatamente que triângulos fazem sombra sobre que triângulos. Caso contrário iria ser criado um VBO excessivamente grande com posições que não seriam utilizadas. Para resolver este problema é feito um cálculo prévio onde é determinado o tamanho exato do VBO e só em seguida é efetuada a sua criação.

Após as alterações feitas ao modo de renderização utilizado, foi necessário identificar os passos mais morosos do algoritmo. Foi relativamente fácil de verificar que se tratava do passo onde é feita a comparação de cada triângulo da cena com todos os restantes. Este passo é relativamente simples em termos de equações envolvidas, mas torna-se complexo devido ao número de vezes que é repetido.

Com a paralelização do algoritmo é possível fazer de uma só vez n comparações, o que significa que pode ser feita em simultâneo a comparação de cada face com todas as restantes. A nível de complexidade vai passar de n por n para apenas 1 por 1. Isto porque em cada *thread* vai ser feita a comparação de apenas dois triângulos, o que

resulta na comparação em simultâneo de todos os triângulos com todos os triângulos.

$$X = blockIdx.x \times blockDim.x + threadIdx.x; \tag{3.5}$$

$$Y = blockIdx.y \times blockDim.y + threadIdx.y; \tag{3.6}$$

Os *kernels* foram elaborados com base nos ciclos utilizados para efetuar a comparação de triângulos na versão sequencial do algoritmo DSP (passo 3(c)iiJ). Isto porque num *kernel* não existe necessidade de ciclos, pois através das equações (3.5) e (3.6) é possível determinar o identificador único de cada *thread* (threadId).

Através do quadrado do número total de triângulos existentes na cena é possível determinar o número exato de *threads* necessárias. Isto faz com que, por exemplo, na *thread(2,1)* vá ser comparada a face n^o 2 com a face n^o 1, para assim se verificar se a face n^o 1 faz sombra sobre a face n^o 2.

Foram então codificados dois *kernels*, denominados *kernelAuxiliar* e o *kernelPrincipal*. O *kernelPrincipal* faz a comparação de triângulos (secção 3.4). No que respeita ao *kernelAuxiliar* é efetuado um cálculo de modo a saber exatamente qual o número de faces que fazem sombra sobre cada uma das restantes faces da cena. Isto torna-se necessário pois permite reduzir o tamanho do VBO, removendo assim posições vazias.

3.6.3 Algoritmo DSP Paralelizado

Com as alterações referidas anteriormente, a paralelização do Algoritmo 1 resultou nos seguintes passos:

1. Carregar modelo OBJ em memória.
2. Alocar memória na GPU para poder armazenar num array apenas o número de triângulos que faz sombra sobre cada um dos triângulos da cena.
3. Lançar o *kernelAuxiliar* de modo a determinar tamanho do VBO, que corresponde ao número total de triângulos que está à frente de cada um dos triângulos da cena. Para isso é necessária a utilização da equação (3.4), através da qual vai ser comparado cada triângulo (plano) com cada um dos vértices dos restantes triângulos existentes.

- Caso se conclua que os três vértices comparados se encontrem à frente do triângulo, são incrementadas 3 unidades na posição do array correspondente ao número do triângulo.
- Se da comparação dos três vértices com o triângulo se concluir que dois deles estão à frente do triângulo e apenas um está atrás do mesmo, são incrementadas 6 unidades na posição do array correspondente ao triângulo tratado.
- Se se concluir que apenas um dos vértices se encontra à frente do triângulo e os restantes dois atrás do mesmo, são incrementadas 3 unidades na posição do array correspondente ao triângulo em causa.
- Se os três vértices comparados se encontrarem todos atrás do triângulo não é incrementado qualquer valor.

4. Somar todos os valores obtidos no *kernelAuxiliar*, relativos ao número de triângulos que fazem sombra sobre cada triângulo da cena.

5. Criar VBO com tamanho igual ao resultado do somatório realizado no passo anterior.

6. Lançar o *kernelPrincipal* onde o VBO vai ser preenchido apenas com os vértices dos triângulos que formam as sombras. Para isso é necessário aplicar novamente a equação 3.4 para cada plano (**P**) que contenha um triângulo e testar os vértices dos restantes triângulos (**T**), de modo a saber quantos vértices estão à frente e atrás do plano.

   - Caso os 3 vértices de **T** estejam à frente de **P**, são os três adicionados ao VBO.
   - No caso de dois vértices de **T** estarem à frente de **P** e apenas um atrás de **P**, significa que vão ser calculados os pontos de interseção das arestas de **T** com **P**. Com estes pontos de interseção e os vértices que estão à frente de **P** é possível formar dois novos triângulos, dos quais os vértices vão ser adicionados ao VBO.
   - Caso apenas um dos vértices de **T** esteja à frente de **P** e os dois restantes estejam atrás de **P**, vão ser calculados os pontos de interseção entre **P** e as arestas de **T**. Os vértices do triângulo, formado pelo vértice de **T** à frente de **P** e os dois pontos de interseção encontrados, são adicionados ao VBO.

- Caso nenhum dos vértices de **T** se encontre à frente de **P**, não é adicionado qualquer vértice ao VBO.

7. Libertar a memória reservada em GPU, exceto VBO, visto que os cálculos necessários já se encontram realizados e o VBO preenchido.

Após serem calculadas as sombras de todos os triângulos da cena, são então desenhados cada um dos triângulos visíveis da mesma maneira que na versão sequencial do algoritmo, com a exceção do passo 3(c)iiJ, onde são desenhadas as sombras projetadas com o recurso ao VBO. Mais concretamente, este passo consiste no seguinte:

- Desenhar sombras sobre o triângulo desenhado anteriormente. Isto é conseguido desenhando de uma só vez todos os triângulos do VBO que correspondem ao triângulo em causa. Como a matriz de projeção resulta da combinação da *model-view* com a matriz M (cf. 3.1), tudo o que for desenhado está automaticamente projetado sobre o triângulo já desenhado.

3.7 Resultados Experimentais

Nesta secção são apresentados os resultados produzidos pelo algoritmo descrito anteriormente neste capítulo. Na obtenção destes resultados foram tidos em conta várias cenas 3D e configurações de hardware.

3.7.1 Resultados Visuais

De modo a testar o algoritmo de sombras proposto neste capítulo foi elaborada uma aplicação gráfica com recurso à linguagem de programação C++ e às bibliotecas gráficas do OpenGL. Para a elaboração da versão paralela do algoritmo foi também utilizado o CUDA Toolkit 5.0. O IDE escolhido para elaboração da aplicação referida foi o Visual Studio 2012 e o sistema operativo Windows 8 Pro de 64 bits.

A aplicação gráfica acima referida permite o controlo da fonte de luz por parte do utilizador de modo a possibilitar a alteração interativa da sombra da cena em tempo real. O utilizador pode também remover interativamente as sombras de modo a verificar o impacto que as mesmas causam na cena. O mesmo sucede com as texturas, caso existam.

Para poder observar o comportamento do algoritmo, foram utilizados vários modelos de cenas 3D com diferentes características. O nome dos modelos e o seu número de triângulos encontram-se na Tabela 3.1.

Nome do modelo	Número de triângulos	Figura
Pillar	44	3.4(a)
Geometric1	62	3.4 (b)
Rock	194	3.4(c) e 3.5
Geometric2	288	3.4(d)
Beacon	1044	3.6
WoodBench	2130	3.7
Water World	2230	3.8
Piano	2490	3.9
Wagen	4420	3.10
Building	4660	3.11
Hut	5062	3.12

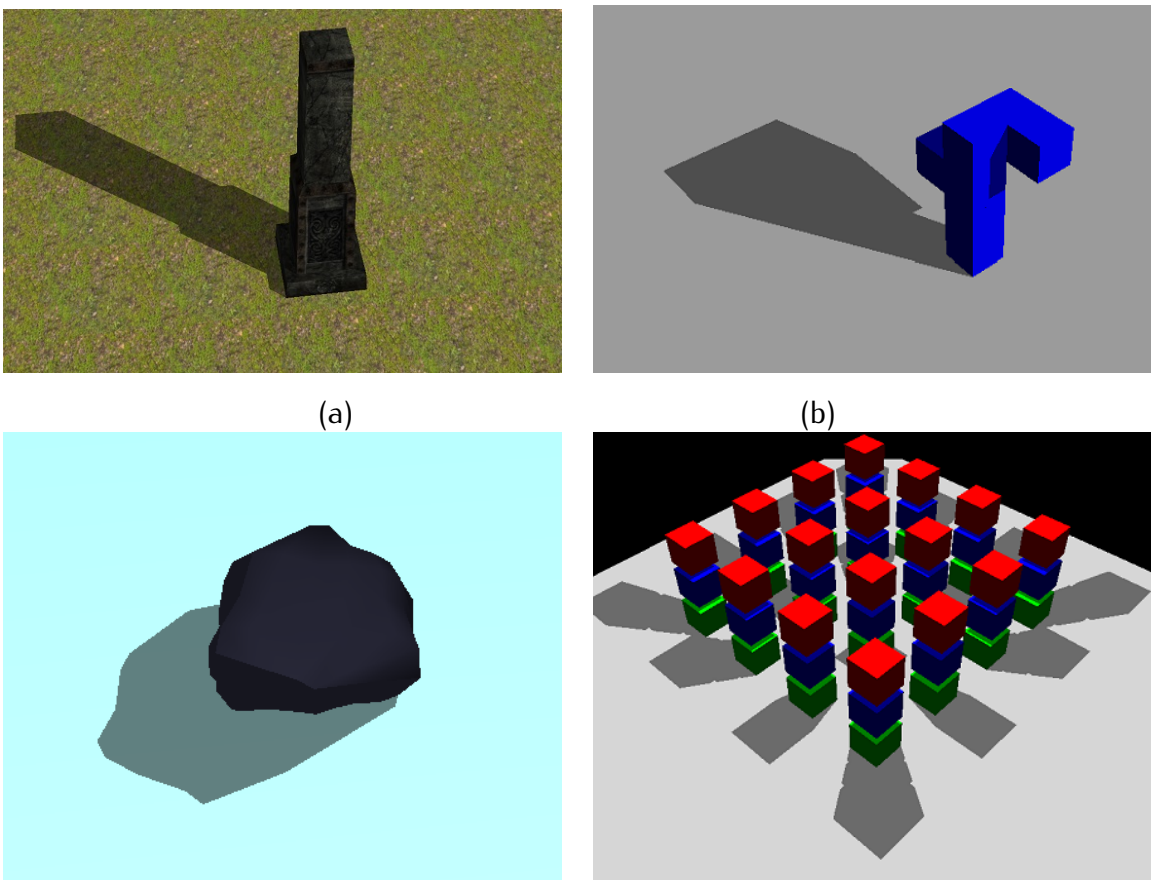
Tabela 3.1: Modelos utilizados na obtenção de resultados.

Como já foi referido anteriormente, foram desenvolvidas duas versões do mesmo algoritmo, sequencial e paralela. Logo os resultados visuais obtidos e apresentados em seguida vão ser agrupados em diferentes subsecções. Em primeiro lugar serão apresentados os resultados da versão sequencial e de seguida os resultados da versão paralela.

3.7.1.1 Versão Sequencial

Na versão sequencial do algoritmo de sombras baseadas em projeção foram utilizadas cenas de reduzida complexidade e, na sua maioria, sem a presença de texturas. Isto provoca alguma falta de realismo à cena. No entanto, nesta versão, o principal objetivo passava essencialmente por verificar o correto funcionamento do algoritmo.

Exemplo disso são as cenas com os modelos Pillar, Geometric1, Rock e Geometric2 representadas na Figura 3.4. Em particular, é visualmente percetível a existência de auto-sombras (Figura 3.4(b)) e a ausência de *aliasing*.



(a) (b)

(c) (d)

Figura 3.4: Modelos utilizados na versão sequencial do algoritmo SBP: (a) Pillar com 44 triângulos; (b) Geometri1 com 62 triângulos; (c) Rock com 194 triângulos; (d) Geometric2 com 288 triângulos

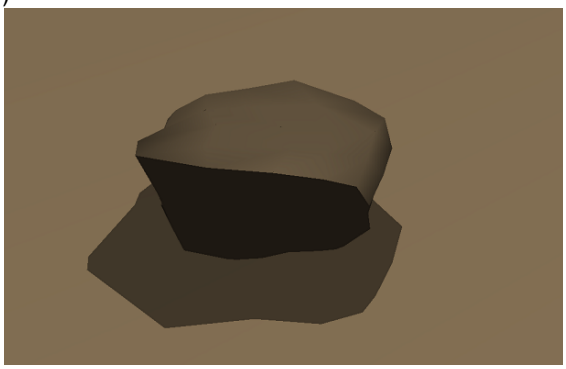
3.7.1.2 Versão Paralela

Nesta versão do algoritmo DSP foram utilizadas cenas bastante mais complexas do que as utilizadas na versão sequencial, quer no número de triângulos que as constituem, quer na complexidade das formas apresentadas.

À semelhança do que acontece na versão sequencial, é possível observar que o algoritmo lida bem com sombras, auto-sombras e *aliasing*. Estas vantagens do algoritmo tornaram-se evidentes quer em modelos essencialmente planos, quer naqueles que apresentam superfícies visualmente curvas, como se ilustra nas Figuras 3.5 - 3.12.

(a)

(b)



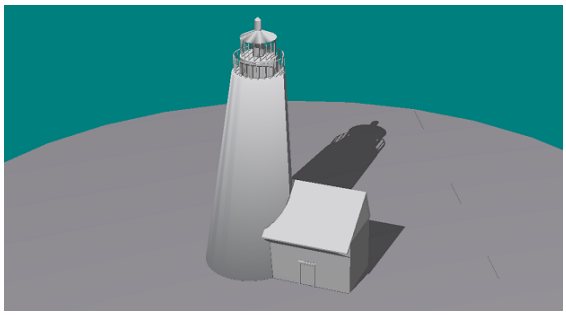
(c)

(d)

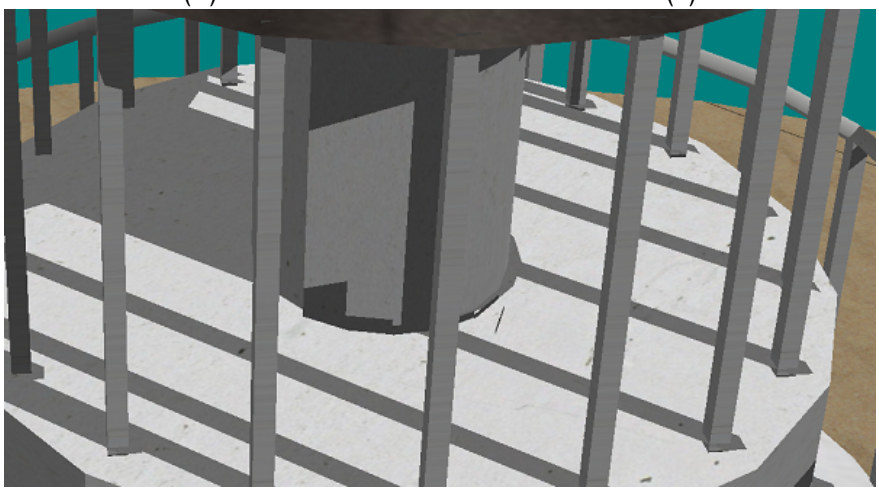
Figura 3.5: Resultado obtidos com o modelo "Rock" na versão paralela.

(a)

(b)



(c)



(d)

Figura 3.6: Resultado obtido com o modelo "Beacon" na versão paralela.

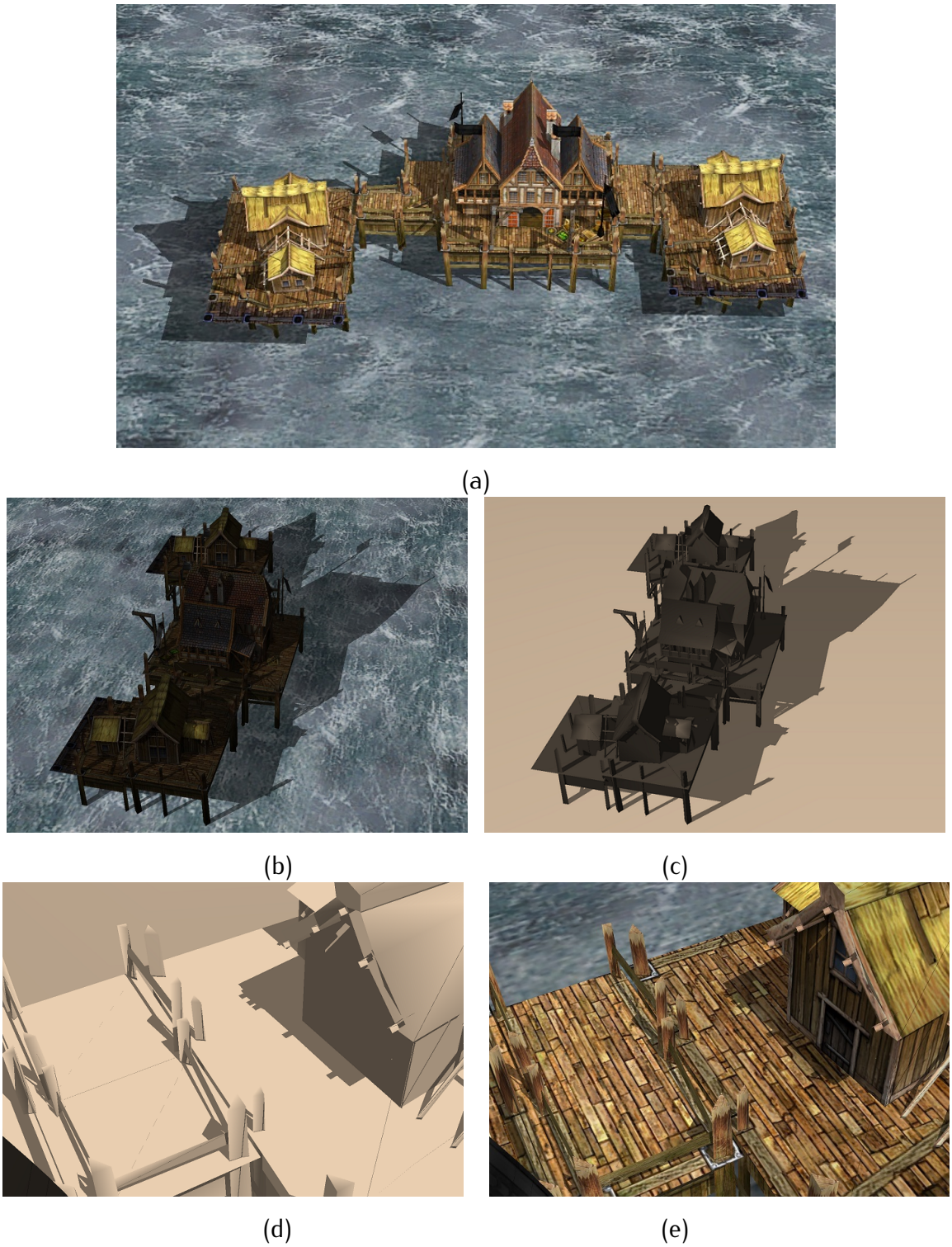
(a)

(b)

(c)

(d)

Figura 3.7: Resultado obtido com o modelo "Woodbench" na versão paralela.

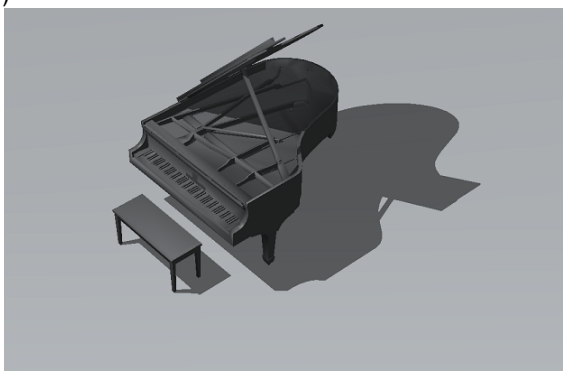


(a) (b) (c) (d) (e)

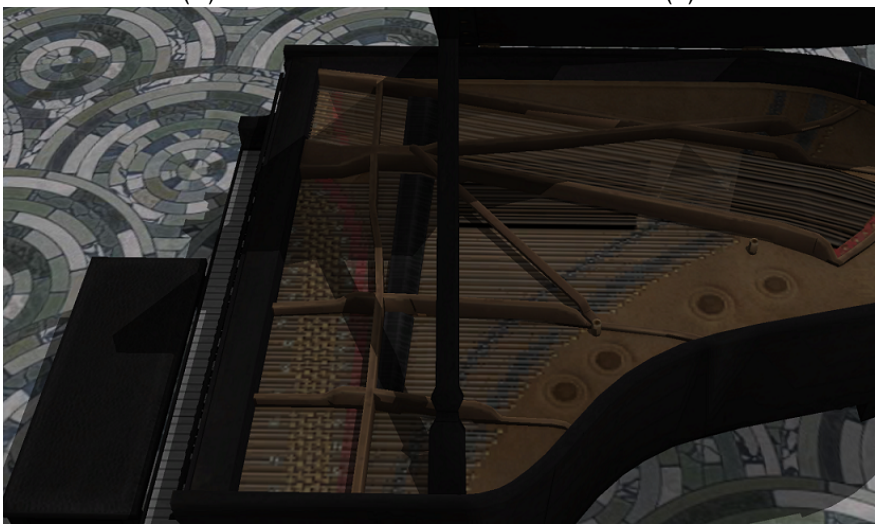
Figura 3.8: Resultado obtido com o modelo "WaterWorld" na versão paralela.

(a)

(b)

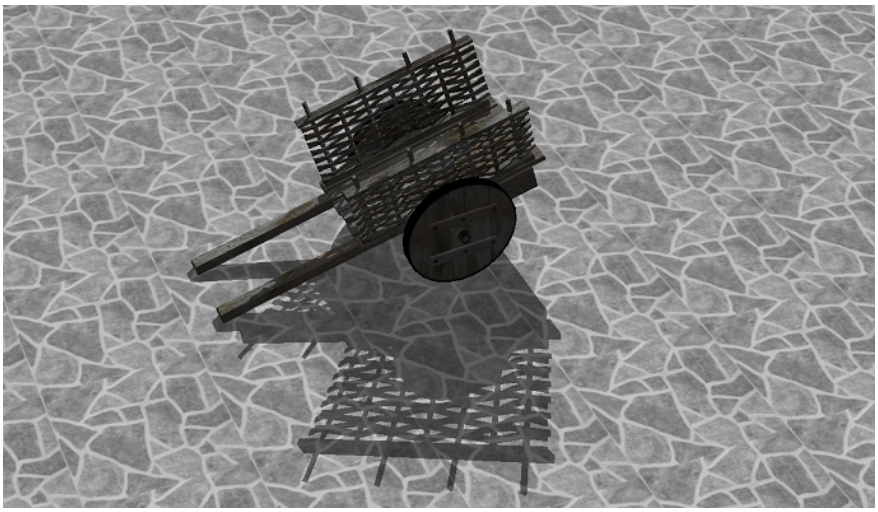


(c)

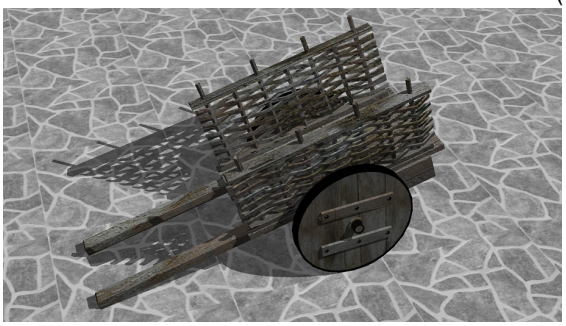


(d)

Figura 3.9: Resultado obtido com o modelo "Piano" na versão paralela.



(a)

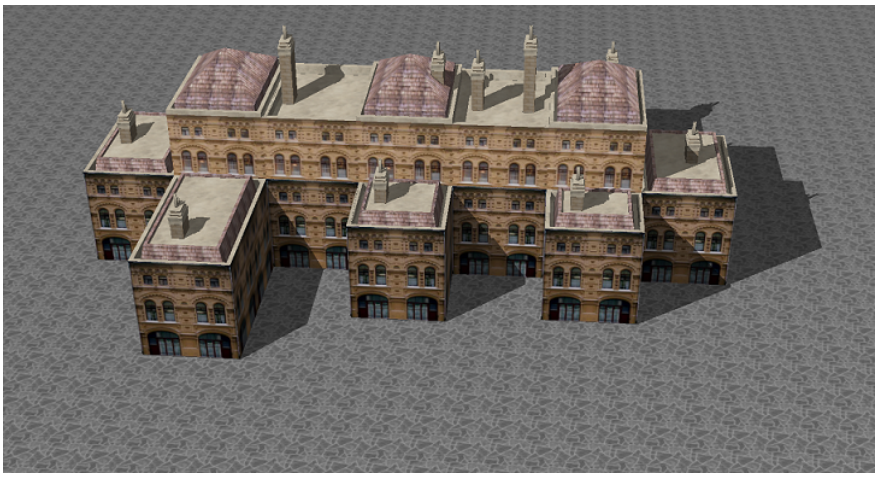


(b)

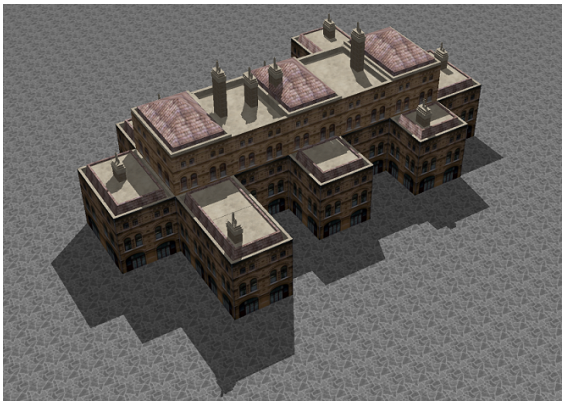
(c)

(d)

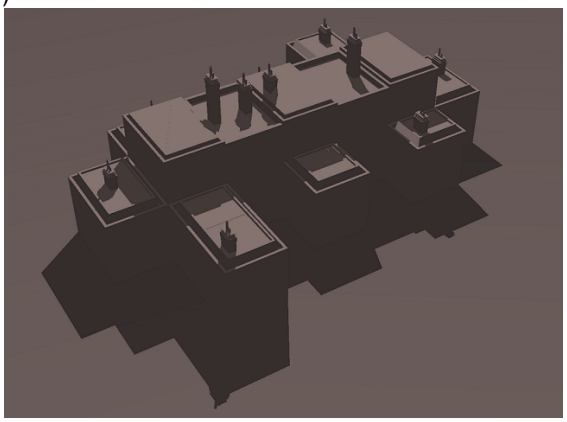
Figura 3.10: Resultado obtido com o modelo "Wagen" na versão paralela.



(a)



(b)



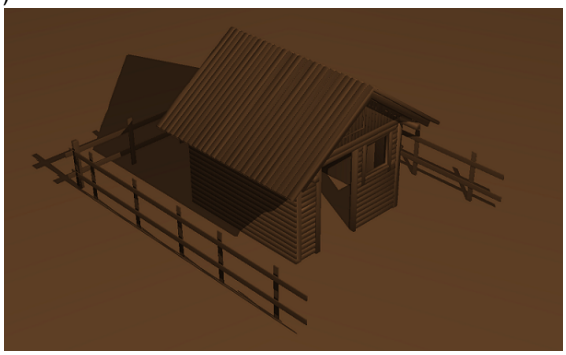
(c)

(d)

Figura 3.11: Resultado obtido com o modelo "Building" na versão paralela.

(a)

(b)



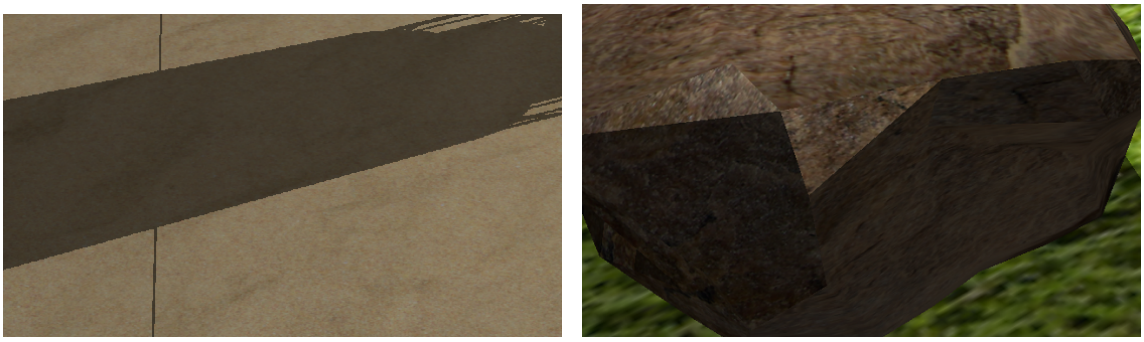
(c)

(d)

Figura 3.12: Resultado obtido com o modelo "Hut" na versão paralela.

3.7.1.3 Artefactos Visuais

Durante a fase de testes do algoritmo, foram identificados alguns "artefactos" nas sombras obtidas. O primeiro caso onde é notório alguma irregularidade nas sombras está relacionado com triângulos pertencentes ao mesmo plano, onde se nota a existência de sombra na fronteira que separa os diferentes triângulos (Figura 3.13(a)). Este caso não devia ocorrer pois, estando os triângulos contidos no mesmo plano têm o mesmo vetor normal, logo não deviam causar sombra uns sobre os outros. No entanto, este problema só é percetível quando o objeto é visto de perto.



(a) (b)

Figura 3.13: Artefactos em sombras.

O segundo caso onde são notórias falhas na sombra obtida com o algoritmo implementado, está relacionado com objetos através de superfícies curvas simuladas com recurso a um baixo número de polígonos. Isto vai provocar uma silhueta não muito realista, formada pela fronteira que separa as zonas sombreada das zonas iluminadas (Figura 3.13(b)). No entanto a utilização de um maior número de triângulos na simulação de uma superfície não plana atenua o problema.

3.7.2 Resultados Quantitativos de Desempenho

Com a finalidade de medir o desempenho do algoritmo implementado, foi utilizado um contador de Frames Per Second (FPS). Assim é possível obter resultados quantitativos em relação ao desempenho do algoritmo na renderização de diferentes cenas. Para além do contador de FPSs, é também possível obter o tempo necessário para calcular apenas as sombras de cada cena.

Refira-se que, o algoritmo foi testado em duas máquinas com características distintas de

modo a verificar a influência do hardware no comportamento do algoritmo. A primeira máquina apresenta um Intel Core 2 Duo T9550 a 2.66 GHz e uma Nvidia GeForce 9700M GT com 512MB de memória dedicada. A segunda tem um processador Intel Core i7 920 a 2.67 GHz, 8GB de RAM, e uma placa gráfica GeForce GTX 295 com 896MB de memória dedicada.

3.7.2.1 Versão Sequencial

Na versão sequencial foram testados quatro modelos, nos quais a complexidade (em número de triângulos) varia de forma gradual (Tabela 3.2). Em seguida são apresentadas as tabelas e os respetivos gráficos dos FPSs obtidos nas duas máquinas onde foi testado o algoritmo.

Nome do modelo	Número de triângulos	FPS máquina 1	FPS máquina 2
Pillar	44	24	22
Geometric1	62	18	21
Rock	194	4	4
Geometric2	218	2	2

Tabela 3.2: Resultados dos modelos testados na versão sequencial do algoritmo DSP.

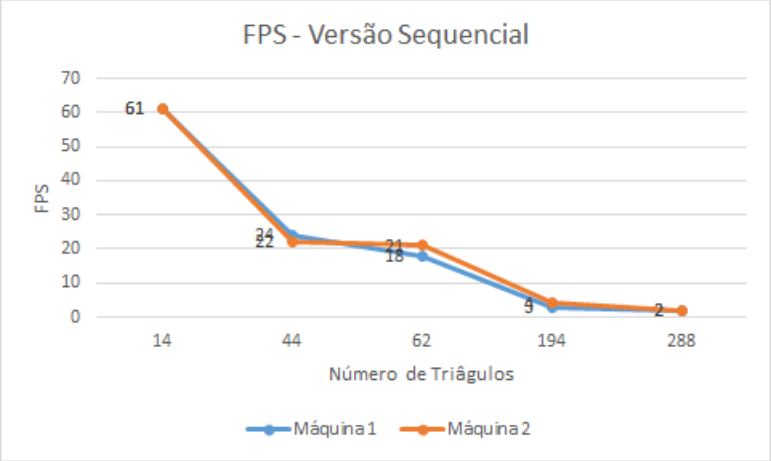


Figura 3.14: Gráfico de desempenho da versão sequencial do algoritmo.

Nome do Modelo	Número de Triângulos	Tempo - Máquina 1	Tempo - Máquina 2
Pillar	44	0.025	0.056
Geometric1	62	0.233	0.07
Rock	194	0.567	0.507
Geometric2	218	1.532	1.302

Tabela 3.3: Tempos do cálculo das sombras na versão sequencial do algoritmo.

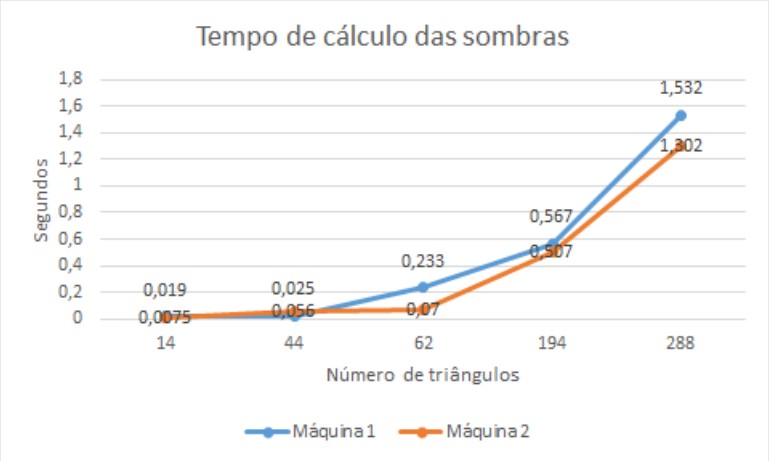


Figura 3.15: Gráfico do tempo necessário para calcular as sombras na versão sequencial.

Na versão sequencial existe uma similaridade no comportamento do algoritmo nas duas máquinas, verificando-se um decréscimo acentuado dos valores de FPSs obtidos.

Do mesmo modo, no que respeita aos tempos necessários para o cálculo das sombras, o comportamento do algoritmo é semelhante em ambas as máquinas onde foi testado. Verifica-se com base nos dados recolhidos, que o tempo necessário para o cálculo das sombras apresenta um comportamento inversamente proporcional ao dos valores de FPSs, aumentando rapidamente (veja-se Tabela 3.3 e Figura 3.15).

Pode constatar-se com base nos valores obtidos nesta versão do algoritmo, que não existe grande dependência das características da máquina utilizada. Pois, tanto nos FPSs obtidos como nos tempos de cálculo das sombras, o comportamento do algoritmo é similar.

3.7.2.2 Versão Paralela

Na versão paralela do algoritmo, foram utilizados modelos de maiores dimensões do que na versão analisada anteriormente. De seguida são apresentados os valores de FPSs obtidos e também os tempos de cálculo dos *kernels* para cada um dos modelos.

Nome do modelo	Número de triângulos	FPS máquina 1	FPS máquina 2
Rock	194	61	61
Beacon	1044	25	32
WoodBench	2130	11	22
Water World	2230	10	20
Piano	2490	-	16
Wagen	4420	-	10
Building	4660	-	9
Hut	5062	-	9

Tabela 3.4: Resultados dos modelos testados na versão paralela do algoritmo DSP.

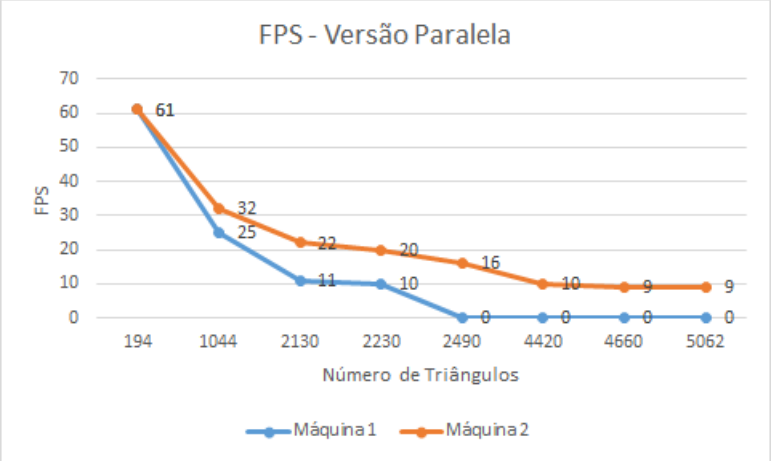


Figura 3.16: Gráfico de desempenho da versão paralela do algoritmo.

A máquina 1 é um computador portátil com alguns anos, pelo que a sua placa gráfica não permite testar os modelos com um maior número de triângulos, o que não acontece com a máquina 2, que é um computador de secretária com uma placa gráfica mais recente.

Através da análise dos dados apresentados na Tabela 3.5 e na Figura 3.17, é possível

Nome do Modelo	Número de Triângulos	Tempo - Máquina 1	Tempo - Máquina 2
Rock	194	0.15	0.015
Beacon	1044	0.135	0.031
WoodBench	2130	0.912	0.203
Water World	2230	1.007	0.208
Piano	2490	1.36	0.281
woodcutter	3471	-	0.499
Wagen	4420	-	0.882
Building	4660	-	0.977
Hut	5062	-	1.151

Tabela 3.5: Tempo do cálculo das sombras na versão paralela.

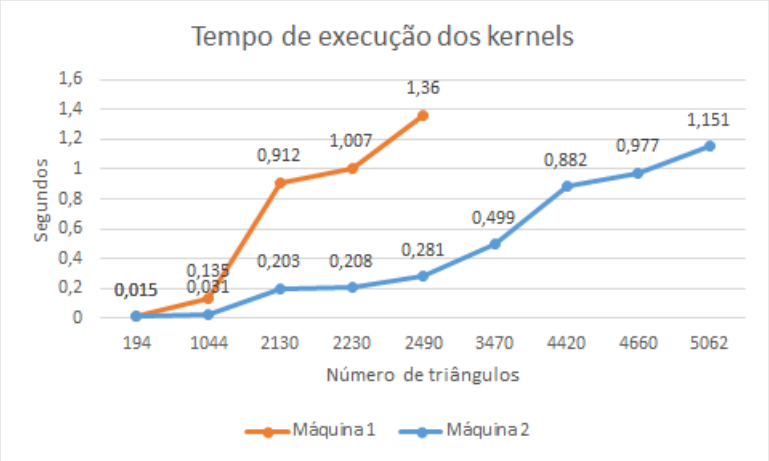


Figura 3.17: Gráfico do tempo necessário para calcular as sombras na versão paralela.

concluir que o desempenho do algoritmo na sua versão paralela decresce de forma mais acentuada na máquina 1 do que na máquina 2. Na máquina 1 o algoritmo atinge os 10 FPS com um modelo de aproximadamente 2000 triângulos, o que apenas se verifica para a máquina 2 com um modelo de 4000 triângulos.

Os valores de FPS apresentados foram obtidos após a sombra ter sido geradas (execução dos kernels) e referem-se por isso ao desempenho do algoritmo na apresentação das sombras obtidas. Nas Tabela 3.5 encontram-se os tempos que demoram a ser calculadas as sombras.

Na versão paralela em GPU, os tempos obtidos são bastante mais baixos na máquina 2

do que na máquina 1. É possível verificar que ambas as máquinas obtêm o mesmo valor para o modelo mais simples, de apenas 194 triângulos, mas que na máquina 1 existe um aumento muito mais acentuado à medida que o número de triângulos aumenta. Enquanto na máquina 2 o aumento dos tempos ocorre de forma mais suave.

3.8 Notas Suplementares

O algoritmo DSP pode ser visto como uma extensão ao algoritmo de Blinn quando aplicado a todas as faces de uma cena de modo a gerar sombras entre si, o que significa que a complexidade do algoritmo sequencial é $\mathcal{O}(n^2)$. A paralelização do algoritmo em CUDA reduz a complexidade de forma significativa, embora o CUDA seja somente usado para o cálculo das interseções entre triângulos.

Capítulo 4

Conclusões e Trabalho Futuro

A investigação efetuada teve como principal intuito reinventar um algoritmo de geração de sombras por projeção direta. Isso requereu um levantamento preliminar das técnicas de geração de sombras existentes na literatura, bem como estudar as suas características.

Tendo isto em conta, foi escolhido o algoritmo de Blinn como ponto de partida para o desenvolvimento de um novo algoritmo para a geração de sombras vincadas que servissem não só para gerar sombras em superfícies planas, como também em superfícies curvas. Havia ainda o propósito de gerar auto-sombras e remover o efeito de escada (ou *aliasing*), que é tão insidioso noutros algoritmos.

Após a implementação do algoritmo na sua versão sequencial, verificou-se que era bastante moroso quando aplicado a cenas relativamente simples, o que se deve ao facto do número de vezes que cada triângulo necessita ser analisado ser igual ao quadrado do número total de triângulos existentes na cena. Ou seja, o aumento do número de triângulos na cena provoca um aumento acentuado nos cálculos necessários à obtenção da sombra desejada. Em termos mais concretos, a complexidade do algoritmo é quadrática. Foi por isso decidido avançar para a elaboração de uma versão paralela em GPU (CUDA).

Apesar da paralelização em GPU atenuar o problema da complexidade computational, poderão ser exploradas no futuro outras abordagens de modo a proporcionar ao algoritmo uma maior escalabilidade. Por exemplo, poderá ser interessante utilizar estruturas de aceleração antes de serem calculadas as sombras. Em alternativa, poderá

proceder-se à alteração da forma como é feita a projeção das sombras sobre os diferentes as diferentes facetas. Em vez da deixar a projeção a cargo do OpenGL e desenhar as sombras de cada de cada triângulos individualmente, poderia ser determinada a projeção dos vértices dos triângulos que compõem a sombra e guardar os triângulos já projetados, o que resultaria na necessidade de fazer a projeção apenas uma vez para cada triângulo, ao contrário do que é feito no algoritmo, onde sempre que a cena é atualizada é dada a instrução ao openGL para fazer a projeção das sombras sobre cada triângulo. Outra alternativa será simplesmente paralelizar completamente o algoritmo na GPU.

Bibliografia

[1] J.-M. Hasenfratz, M. Lapierre, N. Holzschuch, and F. Sillion. A survey of real-time soft shadows algorithms, 2003.

[2] J. Blinn. Me and my (fake) shadow. *IEEE Comput. Graph. Appl.*, 8(1):82–86, January 1988.

[3] Tomas Akenine-Möller, Eric Haines, and Natty Hoffman. *Real-Time Rendering 3rd Edition*. A. K. Peters, Ltd., Natick, MA, USA, 2008.

[4] R. Ananthuni. A web-based visualization and reposition scheme for scientific data. In *The 2006 Intl Conf. on Modeling Simulation and Visualization Methods (MSV'06)*, 2006.

[5] Lance Williams. Casting curved shadows on curved surfaces. In *Proceedings of the 5th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '78, pages 270–274, New York, NY, USA, 1978. ACM.

[6] Franklin C. Crow. Shadow algorithms for computer graphics. *SIGGRAPH Comput. Graph.*, 11(2):242–248, July 1977.

[7] Nicholas Maina Kiguta. Investigation into the feasibility of shadow generation on mobile graphic cards. Master's thesis, University of Iowa, 2010.

[8] Paul S. Heckbert and Michael Herf. Simulating soft shadows with graphics hardware. Technical report, Carnegie Mellon University, 1997.

[9] Bruce Gooch, Peter pike J. Sloan, Amy Gooch, Peter Shirley, and Richard Riesenfeld. Interactive technical illustration. In *In Proceedings of the 1999 symposium on Interactive 3D graphics*, 1999.

[10] João Abrantes. Analysis and proposal of a shadow mapping remix approach. Master's thesis, Universidade do Minho, 2009.

[11] Marc Stamminger and George Drettakis. Perspective shadow maps. In John Hughes, editor, *Proceedings of ACM SIGGRAPH*, Annual Conference Series, pages 557 – 562. ACM Press/ ACM SIGGRAPH, July 2002.

[12] Tobias Martin and Tiow-Seng Tan. Anti-aliasing and continuity with trapezoidal shadow maps. In *Proceedings of the Fifteenth Eurographics conference on Rendering Techniques*, EGSR'04, pages 153–160, Aire-la-Ville, Switzerland, Switzerland, 2004. Eurographics Association.

[13] Fan Zhang, Hanqiu Sun, Leilei Xu, and Lee Kit Lun. Parallel-split shadow maps for large-scale virtual environments. In *Proceedings of the 2006 ACM international conference on Virtual reality continuum and its applications*, VRCIA '06, pages 311–318, New York, NY, USA, 2006. ACM.

[14] John Tsiombikas. Volume shadows tutorial. `http://nuclear.mutantstargoat.com/articles/volume_shadows_tutorial_nuclear.pdf`.

[15] William T. Reeves, David H. Salesin, and Robert L. Cook. Rendering antialiased shadows with depth maps. *SIGGRAPH Comput. Graph.*, 21(4):283–291, August 1987.

[16] Stefan Brabec and Hans peter Seidel. Shadow volumes on programmable graphics hardware. *Computer Graphics Forum (Eurographics*, 2003:433–440, 2003.

[17] D. Romanick. Planar shadows. `http://people.freedesktop.org/~idr/2007-VGP353/20070409-Planarshadows.pdf`.

[18] Jeff Rogers. Glm: an alias wavefront obj file library. `http://devernay.free.fr/hacks/glm/`.

[19] Nate Robbins. Nate robbins. `http://user.xmission.com/~nate/index.html`.

[20] What is cuda. `https://developer.nvidia.com/what-cuda`.