

UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

COMPARAÇÃO DE TÉCNICAS DE RENDERIZAÇÃO DE SOMBRAS

TRABALHO DE GRADUAÇÃO

TIAGO BOELTER MIZDAL

Santa Maria, RS, Brasil

2013

COMPARAÇÃO ENTRE TÉCNICAS DE RENDERIZAÇÃO DE SOMBRAS

Tiago Boelter Mizdal

Trabalho de Graduação apresentado ao Curso de Ciência da Computação
da Universidade Federal de Santa Maria (UFSM, RS), como requisito
parcial para a obtenção do grau de
Trabalho de Graduação

Orientador: Prof. Dr. Cesar Tadeu Pozzer

Co-orientadora: Ana Trindade Winck

Nº. 369

Santa Maria, RS, Brasil

AGRADECIMENTOS

Agradeço especialmente ao professor Pozzer, que além de me orientar durante o tempo que estive trabalhando no Laboratório de Computação Aplicada (LaCA), é um grande amigo.

À professora Ana por todo o apoio e ajuda que me deu neste semestre, e mesmo com todos os problemas aceitou ser minha co-orientadora.

À minha família por todo apoio e incentivo que deram a mim nesta etapa.

Aos meus colegas do LaCA, Schardong, Lennon, Schirmer, Netto, Alex e Dudu, pela parceria em trabalhos, projetos e jogatinas.

Universidade Federal de Santa Maria
Centro de Tecnologia
Curso de Ciência da Computação

A Comissão Examinadora, abaixo assinada,
aprova o Trabalho de Graduação

**COMPARAÇÃO ENTRE TÉCNICAS DE RENDERIZAÇÃO DE
SOMBRAS**

elaborado por

Tiago Boelter Mizdal

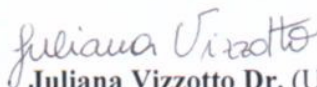
Como requisito parcial para obtenção do grau de
Bacharel em Ciência da Computação

COMISSÃO EXAMINADORA



Ana Trindade Winck, Dr.

(Presidente/Co-Orientadora)



Juliana Vizzotto Dr. (UFSM)



Benhur de Oliveira Stein Dr. (UFSM)

Santa Maria, 22 de Janeiro de 2014

2013

RESUMO

Trabalho de graduação
Curso de Ciência da Computação
Universidade Federal de Santa Maria

COMPARAÇÃO ENTRE TÉCNICAS DE RENDERIZAÇÃO DE SOMBRAS

AUTOR: TIAGO BOELTER MIZDAL

ORIENTADOR: CESAR TADEU POZZER

CO-ORIENTADORA: ANA TRINDADE WINCK

Local da defesa e Data: Santa Maria, 20 de Dezembro de 2013.

As sombras são muito importantes para a percepção humana. Quando um objeto lança uma sombra sobre uma superfície, esta pode nos dar informações sobre o formato do objeto, o formato da superfície que recebe a sombra, e sobre o posicionamento espacial dos objetos. Com o avanço da qualidade gráfica dos jogos, foram desenvolvidas técnicas de geração de sombras que visam tornar os jogos visualmente mais próximos da realidade. Este trabalho tem como o objetivo o estudo de diversas técnicas de renderização de sombras, e desenvolver uma análise comparativa sobre as diferentes técnicas estudadas.

LISTA DE FIGURAS

Figura 1 Apresenta a importância das sombras para a percepção espacial.	8
Figura 2 Representação do funcionamento de um Ray Tracing.....	10
Figura 3 Tipos de Sombra	12
Figura 4 Identificação da umbra e penumbra.....	13
Figura 5 Partes da sombra	14
Figura 6 Exemplo de Hard Shadow (à esquerda) em um jogo de tênis durante o dia, e (à direita) Soft Shadow durante a noite.....	14
Figura 7 A textura (à esquerda) representa a sombra projetada pela estatua (à direita).....	15
Figura 8 Na imagem da esquerda, a cena iluminada. Na imagem do meio o Shadow Map, a imagem da direita tem a cena com sombras.....	17
Figura 9 Representação de shadow volumes e suas respectivas sombras.....	18
Figure 10 Apresentação de um shadow volume utilizando cada plano formado pelos lados de um triângulo em relação a fonte de luz.	18
Figura 11 Apresentação da contagem de shadow maps. Nas imagens de cima a renderização com front-facing, e nas imagens de baixo com back-facing.....	19
Figura 12 Apresentação do processo de contagem utilizando o método "Carmack's Reverse"..	20
Figura 13 Representação do processo de criação de Fake Soft Shadows. A esquerda a imagem renderizada em preto e branco, no meio a imagem da esquerda com aplicação do gaussian blur. E a direita o resultado final.	20
Figura 14 Representação do processo de criação dos penumbra volumes.....	21
Figura 15 Resultado da implementação do Shadow Map.	23
Figure 16 Resultado do Shadow Volumes	24
Figure 17 Resultado das sombras utilizando perspective shadows.....	25

SUMÁRIO

1 INTRODUÇÃO	8
1.1 Objetivos	10
1.1.1 Objetivos gerais	10
1.1.2 Objetivos específicos	10
1.2 Organização do Texto	11
2 Revisão Bibliográfica	12
2.2 Sombras no Mundo Real	12
2.2.1 Tipos de sombra	12
2.2.2 Partes de uma sombra	13
2.2 Técnicas de Projeção de Sombras	15
2.2.1 Perspective Shadows	15
2.2.2 <i>Shadow Maps</i>	16
2.2.3 Shadow Volumes	18
2.2.4 <i>Fake Soft Shadow Maps</i>	20
2.2.5 Soft Shadow Volume	21
3. Implementação	22
3.1 Implementação de <i>Shadow Maps</i>	22
3.2 Implementação de Shadow Volumes	24
3.3 Implementação de <i>Perspective Shadows</i>	25
4. Comparação das Técnicas	26
4.1 Shadow Maps	26
4.1.1 Vantagens	26
4.1.2 Desvantagens	27
4.2 <i>Perspective Shadows</i>	27
4.2.1 Vantagens	28
4.2.2 Desvantagens	28
4.3 <i>Shadow Volumes</i>	29
4.3.1 Vantagens	29
4.3.2 Desvantagens	30
5. Conclusão	30
5.1 Contribuições Alcançadas	30
6.2 Trabalhos Futuros	31
REFERÊNCIAS	32
APÊNDICES	33

1 INTRODUÇÃO

Quando um objeto projeta uma sombra sobre uma superfície, esta pode conter informações sobre o formato do objeto, o formato da superfície e sobre o posicionamento espacial do objeto relativo à superfície (Mamassian, et al., 1998). Destas informações, provavelmente a mais importante é sobre o posicionamento espacial (Figura 1). Leonardo Da Vinci, em 1490, foi a primeira pessoa a analisar as sombras, quando percebeu que ao utilizar sombras e luzes em uma pintura, seria possível evidenciar o espaço tridimensional em suas obras. Da Vinci utilizava sombras estáticas em suas obras, porém, estudos recentes mostram que as sombras em movimento dão à nossa percepção uma dica crucial para o entendimento do posicionamento espacial dos objetos (Mamassian, et al., 1998).

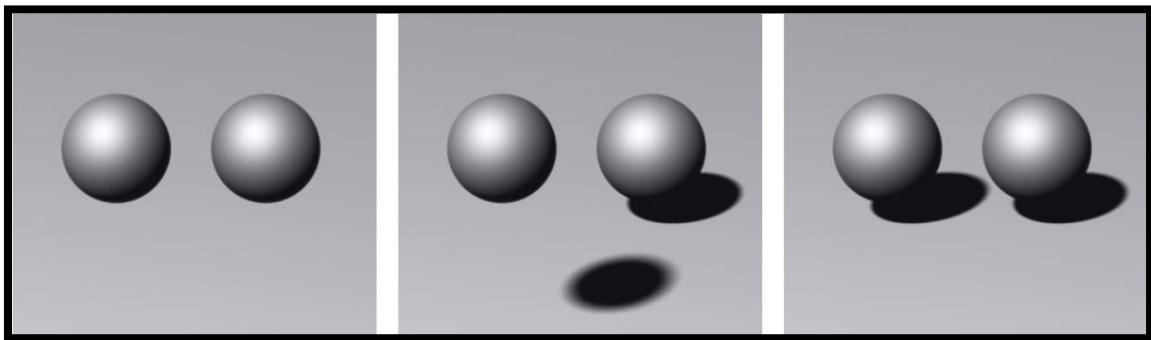


Figura 1 Apresenta a importância das sombras para a percepção espacial.

Quando uma sombra está se movimentando, existem duas possíveis possibilidades para explicar o movimento. Uma possibilidade é quando a fonte de luz que está criando a sombra está em movimento e o objeto que a projeta está imóvel. A outra possibilidade é quando o objeto está se movimentando e a fonte de luz está estática. A percepção visual humana tem como prioridade acreditar que a fonte de luz está parada. A explicação disso se dá ao fato de que com uma fonte de luz estática é possível inferir com mais certeza o movimento, o formato, o tamanho e a posição no espaço de um objeto (Mamassian, et al., 1998).

Para se produzir aplicações gráficas, que possam passar ao usuário a impressão de que estão em um ambiente tridimensional, é muito importante fazer o uso de sombras. Existem três técnicas básicas para a projeção de sombras, *projection shadows*, *shadow*

maps e shadow volumes (Eisemann, et al., 2012). Destas três técnicas básicas, surgiram inúmeras variações que melhoram a qualidade das sombras projetadas e a velocidade com que elas são geradas.

Projection shadows é uma técnica que funciona apenas quando a sombra de um objeto é projetada em um plano. Ela funciona calculando-se a projeção de um objeto em um certo ângulo em um plano, e então desenhando a projeção como um objeto preto no plano.

Shadow maps é provavelmente a técnica mais utilizada para se gerar sombras (Stamminger, et al., 2002). A técnica consiste em renderizar os objetos a partir dos pontos de luz, e armazenar apenas a profundidade de cada pixel perante à fonte de luz no *depth buffer*. Posteriormente, renderizando a cena a partir da câmera, a cada vez que se desenha um pixel, verificar no *depth buffer* se o pixel não está atrás de um outro pixel, caso esteja, desenhá-lo escurecido.

Shadow volumes é uma técnica que aborda a projeção de sombras de uma maneira diferente. A técnica foi introduzida pela primeira vez em 1977 (Crow, 1977), porém apenas em 1991 ela foi capaz de ser implementada em GPU, antes disto as placas de vídeo não possuíam processamento suficiente para que o algoritmo funcionasse em tempo real (Heidmman, 1991). A técnica consiste em criar um volume sombreado para cada triângulo, relativamente a cada fonte de luz, e ao rasterizar a cena, utilizar o *stencil buffer* para verificar quais os pixels que estão inclusos em um volume, e assim desenhá-los mais escuros.

Existem outras técnicas para a geração de sombras, como o *ray tracing* (Whitted, 1980), e o *mental ray* (Nvidia, 2012), ambas as técnicas geram imagens traçando o caminho da luz por meio de raios através dos pixels da câmera, e simulando os efeitos da interação da luz com os objetos (Figura 2). Estas técnicas são principalmente focadas na geração de imagens ultra realistas, porém, nenhuma destas técnicas é capaz de operar em tempo real com o hardware atual.

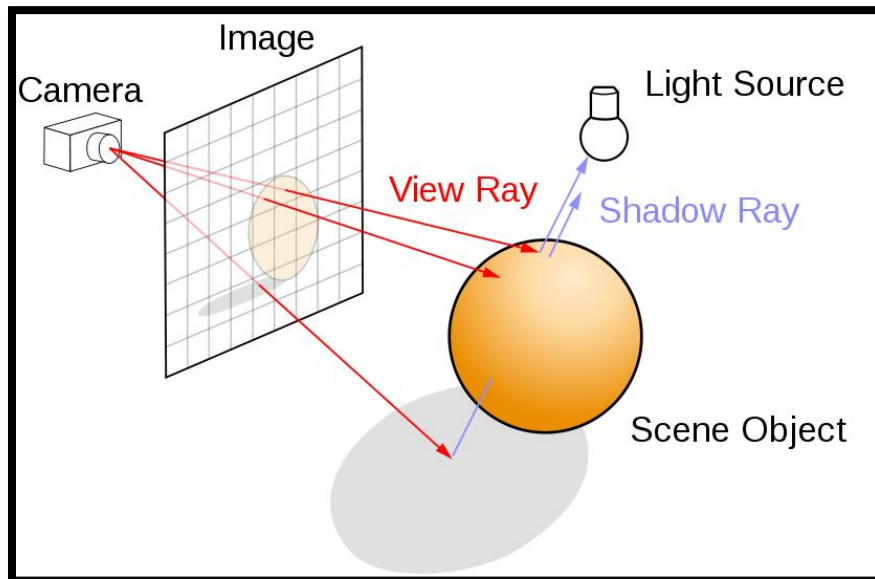


Figura 2 Representação o funcionamento do algoritmo Ray Tracing.

1.1 Objetivos

1.1.1 Objetivos gerais

Esse trabalho tem como objetivo o estudo das técnicas básicas de projeção de sombras e de algumas variações para que se possa comparar as técnicas em relação a qualidade gráfica e desempenho, identificando situações onde cada técnica tenha seu melhor e pior desempenho, para que se possa entender os pontos positivos e negativos de cada técnica.

Para alcançar os objetivos desse trabalho pretende-se implementar as técnicas usando a API Direct3D 11 (Microsoft, 2010).

1.1.2 Objetivos específicos

Define-se como objetivos específicos de desenvolvimento:

- Construção modularizada de um software, para que possa ser facilmente incrementado e modificado;
- Estudo, compreensão e implementação dos algoritmos *projection shadows*, *shadow mapping* e *shadow volume*;

- Realizar uma análise dos algoritmos básicos de geração de sombras a partir da implementação inicial;
- Comparação dos métodos implementados, para ressaltar pontos fortes e fracos, e identificar situações de melhor e pior caso.

1.2 Organização do Texto

O presente trabalho está organizado da seguinte forma: o capítulo 2 contém uma revisão bibliográfica abordando os tópicos de interesse do trabalho, como o funcionamento das sombras, e a descrição das técnicas. No capítulo 3 é detalhada a implementação do trabalho.

No capítulo 3 são apresentados detalhes da implementação. O capítulo 4 contém o cronograma de execução deste trabalho. No capítulo 5 serão apresentados resultados de comparações entre os métodos de projeção de sombras.

2 Revisão Bibliográfica

2.2 Sombras no Mundo Real

No mundo real, quando uma fonte de luz incide sobre uma superfície, a superfície fica iluminada, porém, quando um objeto está posicionado entre a fonte de luz e a superfície, parte da luz é bloqueada pelo objeto e forma-se uma área menos iluminada. Esta área é chamada de sombra (Hasenfratz, et al., 2003).

2.2.1 Tipos de sombra

A medida que a luz interage com o objeto, é possível distinguir dois tipos diferentes de sombras: as sombras próprias, ou *attached shadows*, e as sombras projetadas, ou *cast shadows* (Figura 3).

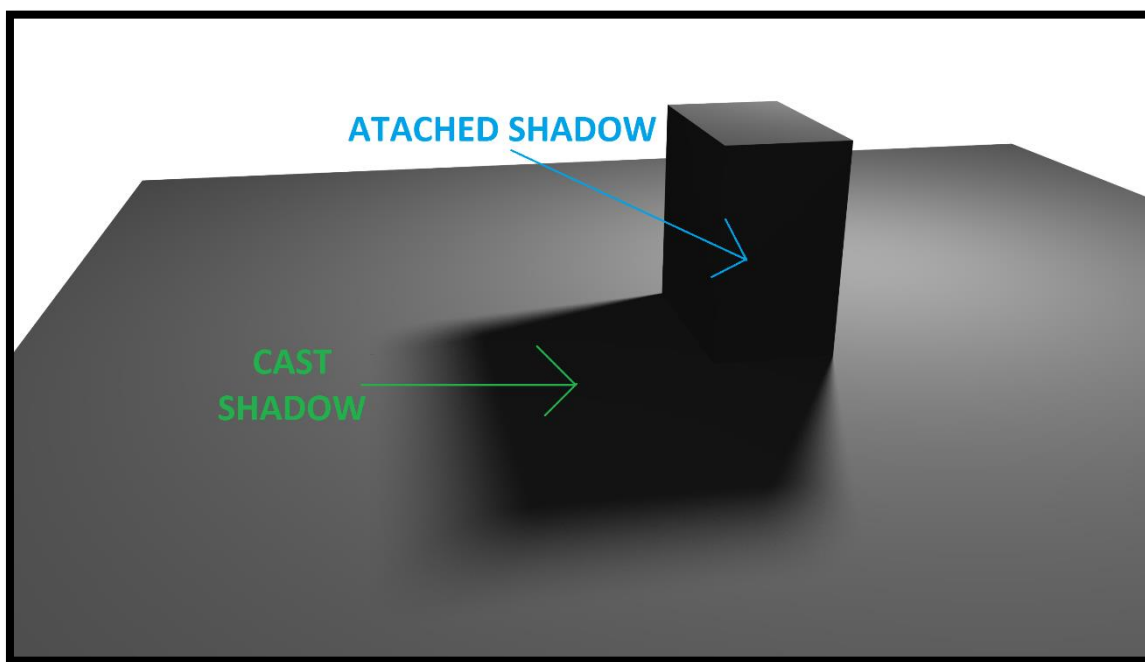


Figura 3 Tipos de Sombra

As sombras próprias são formadas quando uma parte de um objeto bloqueia a luz, e então a outra parte do objeto não recebe a mesma quantidade de iluminação, ficando com uma cor mais escura. São sombras formadas por um objeto nele mesmo.

As sombras projetadas ocorrem quando um objeto bloqueia a luz de uma fonte luminosa, porém o resto da luz é recebido por outras superfícies, assim criando uma silhueta da luz bloqueada em outras superfícies. São sombras projetadas de um objeto em outros.

2.2.2 Partes de uma sombra

A fonte luminosa não é apenas formada por um único ponto no espaço, é formada por uma área que produz uma certa luminosidade. De acordo com a quantidade de iluminação que cada parte da sombra recebe, é possível identificar duas partes diferentes da sombra: a umbra e a penumbra (Figura 4).

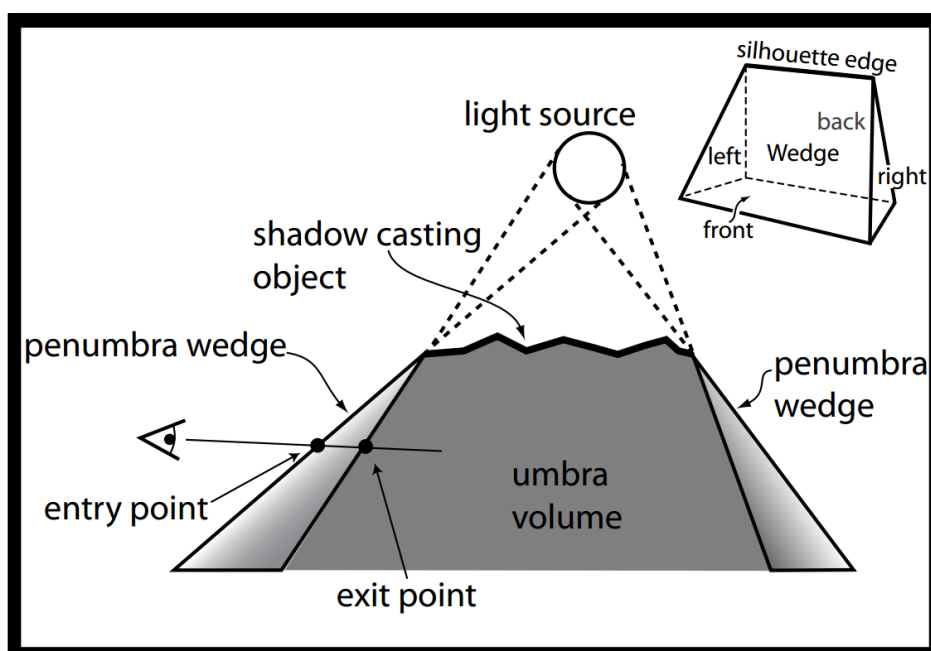


Figura 4 Identificação da umbra e penumbra.

A penumbra acontece quando apenas uma parte da luz é bloqueada pelo objeto e a outra parte ainda incide sobre a superfície, produzindo assim uma sombra menos intensa. A umbra é a parte da sombra que não recebe nenhuma luminosidade diretamente da fonte de luz, e assim tem uma cor mais escura (Figura 5).

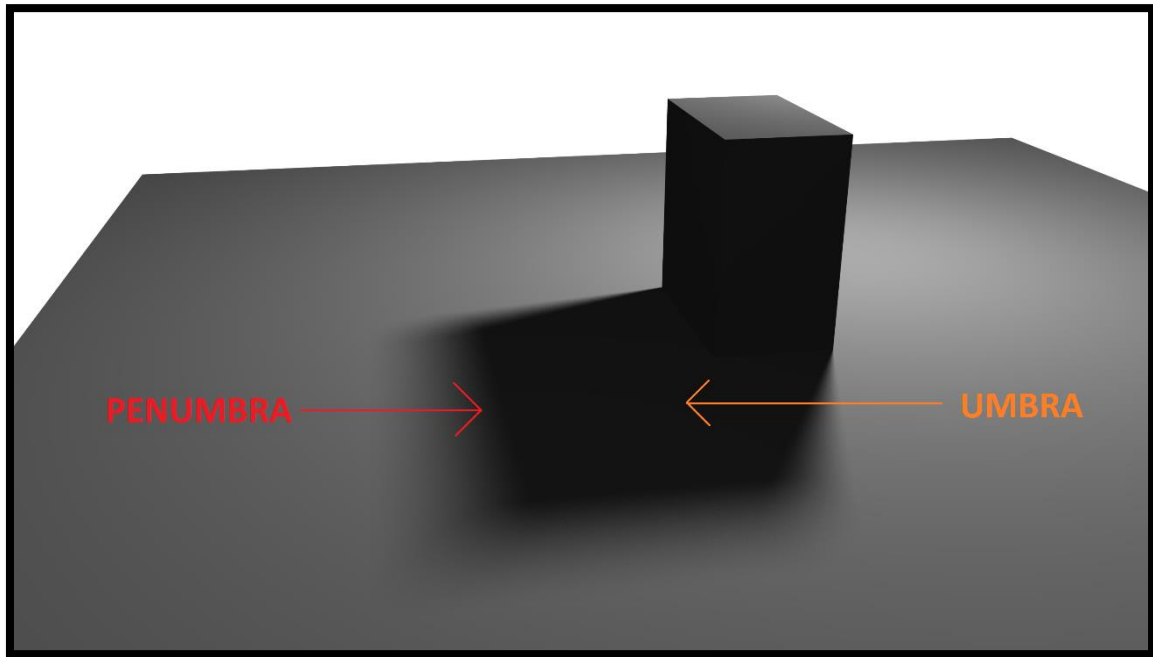


Figura 5 Partes da sombra

Algumas vezes, a fonte luminosa está tão distante dos objetos, que ela pode ser considerada como um único ponto no espaço. Percebe-se que as sombras projetadas por essa fonte de luz não possuem penumbra, ou esta é tão pequena que não pode ser percebida, essas são chamadas de Hard Shadows (Figura 6). Quando a luz projeta uma penumbra, as sombras são chamadas de Soft Shadows (Figura 6).



Figura 6 Exemplo de *Hard Shadow* (à esquerda) em um jogo de tênis durante o dia, e (à direita) *Soft Shadow* durante a noite.

2.2 Técnicas de Projeção de Sombras

A computação gráfica sempre tenta tornar as aplicações mais realistas, e para isso, é necessário que as sombras também sejam realistas, assim tornando possível ao usuário entender melhor o espaço tridimensional apresentado. Com essa finalidade, surgiram diversos algoritmos que visam projetar sombras em uma aplicação gráfica. Alguns algoritmos são capazes de projetar sombras de maneiras ultra realistas, como o *ray tracing* e o *mental ray* mas que não funcionam em tempo real, e assim são ideais para imagens e filmes. Outros algoritmos são capazes de funcionar em tempo real, porém não são tão próximos da realidade. Nesta parte do trabalho será apresentado os algoritmos de projeção de sombra mais comuns utilizados para aplicações em tempo real.

2.2.1 Perspective Shadows

Perspective shadows é um método que se baseia no mapeamento de texturas. Foi introduzido pela primeira vez em 1992 (Segal, et al., 1992). O método funciona como um projetor de slides, a luz passa por um objeto, e então o objeto é projetado em um plano. Ao projetar o objeto no plano, é criada uma textura com a silhueta do objeto, que então é aplicada ao plano (Figura 7).



Figura 7 A textura (à esquerda) representa a sombra projetada pela estatua (à direita).

Para a implementação deste método, deve-se: transformar as coordenadas dos objetos em coordenadas relativas a fonte de luz; transformar as coordenadas que estão na

variação da câmera $[-1, 1]$ em coordenadas de textura $[0, 1]$ e calcular as coordenadas da textura da silhueta do objeto no plano que recebe a sombra. Para isto, deve-se utilizar a seguinte equação:

$$T\alpha = \begin{bmatrix} \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & \frac{1}{2} \end{bmatrix} P V M$$

onde $T\alpha$ são as coordenadas de textura, T é uma matriz 4×4 que representa o plano, M é uma matriz que representa a *model matrix*, V representa a *view matrix* e P a *projection matrix* da fonte de luz. A matriz apresentada na equação serve para transformar coordenadas de câmera em coordenadas de textura.

Então ao renderizar a cena a partir da câmera, para cada fragmento visível do objeto que projeta a sombra, deve se calcular as respectivas coordenadas de textura no plano e desenhar a textura em preto.

2.2.2 Shadow Maps

Shadow mapping é o algoritmo mais utilizado para a projeção de sombras. Este método foi introduzido em 1978 por Lance Williams (Williams, 1978). O método consiste em renderizar a cena a partir da fonte de luz, e armazenar apenas a profundidade de cada pixel. Isto pode ser feito utilizando o *depth buffer*. Esta imagem é chamada de *shadow map*, e ela representa todos os pontos do espaço que são iluminados, ou seja, representa todos os pontos no espaço que projetam uma sombra. Após este passo, renderiza-se a cena através da câmera, e para cada pixel, verificar se este pixel está presente no *shadow map* ou não. Caso esteja o pixel é iluminado, caso contrário, é escurecido. (Figura 8).

Depth buffer, ou Z Buffer, é utilizado no gerenciamento das coordenadas tridimensionais da imagem. Foi criado para solucionar problemas de visibilidade, decidindo quais são os elementos da imagem que estão visíveis e quais os elementos que estão escondidos. O depth buffer armazena, para cada pixel, a distância do pixel ao observador. Caso um outro pixel será desenhado sobre ele, ele verifica as distâncias, e

armazena somente a mais próxima. A distância é dada apenas pela coordenada Z do ponto.

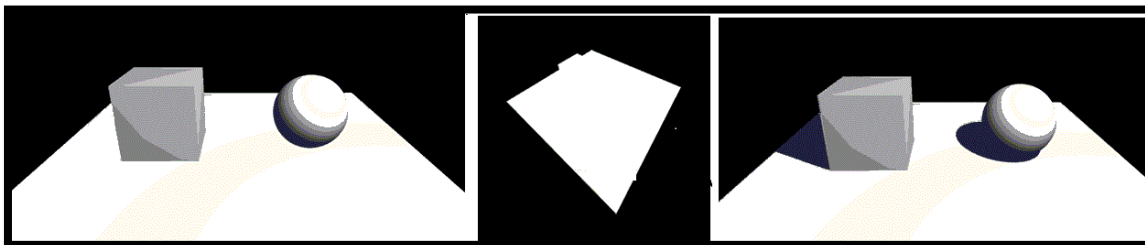


Figure 8 Na imagem da esquerda, a cena iluminada. Na imagem do meio o Shadow Map, a imagem da direita tem a cena com sombras.

O método *shadow map* necessita que a cena seja renderizada duas vezes para funcionar, o que prejudica o desempenho. Em virtude disso, a resolução utilizada no *shadow map* é geralmente menor do que a resolução da aplicação. Isto faz com que a sombra raramente represente a sombra que realmente deveria existir. Isto pode causar dois problemas: as sombras podem acabar ficando “pixeladas” e surge a necessidade de utilizar um erro para comparar os pixels de ambas as imagens, pois como as resoluções são diferentes, é necessário ter um erro para poder mapear pixels de uma imagem na outra. Ao se utilizar um erro muito grande podem ocorrer falhas nas sombras, e existir sombras onde não deveriam, e ao se fazer uso de um erro muito pequeno, podem ocorrer casos onde as sombras de um objeto são projetadas sobre ele mesmo. Ao aumentar a resolução do *shadow map*, pode-se diminuir o erro, porém prejudica-se o desempenho.

Como o método do *shadow map* compara apenas os pixels para ver quais estão mais perto da luz, sendo assim uma operação binária, só podem existir áreas iluminadas, e áreas que não são iluminadas, fazendo assim com que as apenas *hard shadows* possam ser projetadas. Se existir mais de uma fonte luminosa, é necessário renderizar a cena para cada fonte de luz, para calcular o *shadow map* de cada uma, e então comparar todos os *shadow maps* com a câmera para que sejam definidas as áreas sombreadas. Isso pode causar grandes problemas de desempenho.

2.2.3 Shadow Volumes

O método *shadow volumes* foi criado por Tim Crow em 1977, porém, apenas em 1991 que as placas de vídeo tiveram processamento suficiente para processar este algoritmo. O método consiste em, para cada triângulo, criar um volume que representa um espaço em sombras, ou seja, um espaço onde uma certa fonte de luz não ilumina. Estes volumes são chamados de *shadow volumes* (Figura 9).

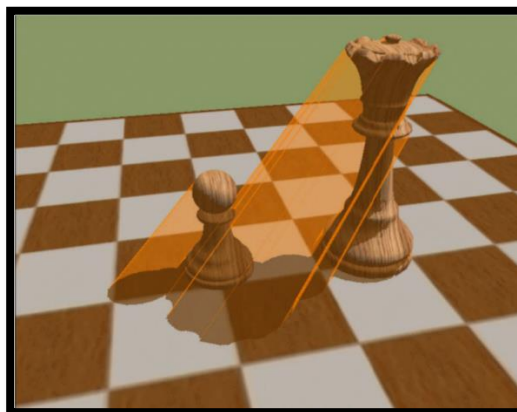


Figura 9 Representação de *shadow volumes* e suas respectivas sombras.

Para criar os *shadow volumes*, é preciso descobrir qual o vetor direção de cada ponto de cada triângulo da cena em relação à fonte de luz. Este vetor é possível de ser calculado subtraindo da posição da fonte de luz à posição do ponto do triângulo. Então estendendo este ponto na direção deste vetor ao infinito, cria-se três planos para cada triângulo, estes três planos definem um shadow volume (Figura 10).

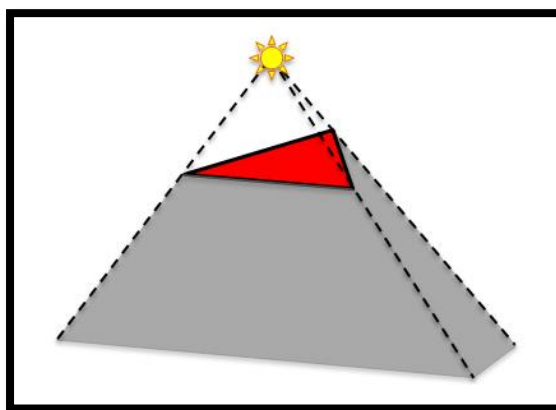


Figure 10 Apresentação de um *shadow volume* utilizando cada plano formado pelos lados de um triângulo em relação a fonte de luz.

Após a criação dos volumes, é necessário definir se um pixel está dentro ou fora de um *shadow volume*. Para isso a cena é renderizada duas vezes, utilizando os *shadow*

volumes e apenas o *stencil buffer*, que é um buffer que pode ser utilizado para limitar a área da imagem que será desenhada. Então cada pixel que será desenhado na tela, é renderizado uma vez com somente o *front-facing* habilitado, e para cada face encontrada, soma-se o valor um ao respectivo pixel no *stencil buffer*. Então os pixels são renderizados novamente, mas com apenas o *back-facing* habilitado, e para cada face encontrada é subtraído o valor um do pixel no *stencil buffer*. Isto irá fazer com que o *stencil buffer* contenha a quantidade de *shadow maps* presentes em cada pixel (Figura 11).

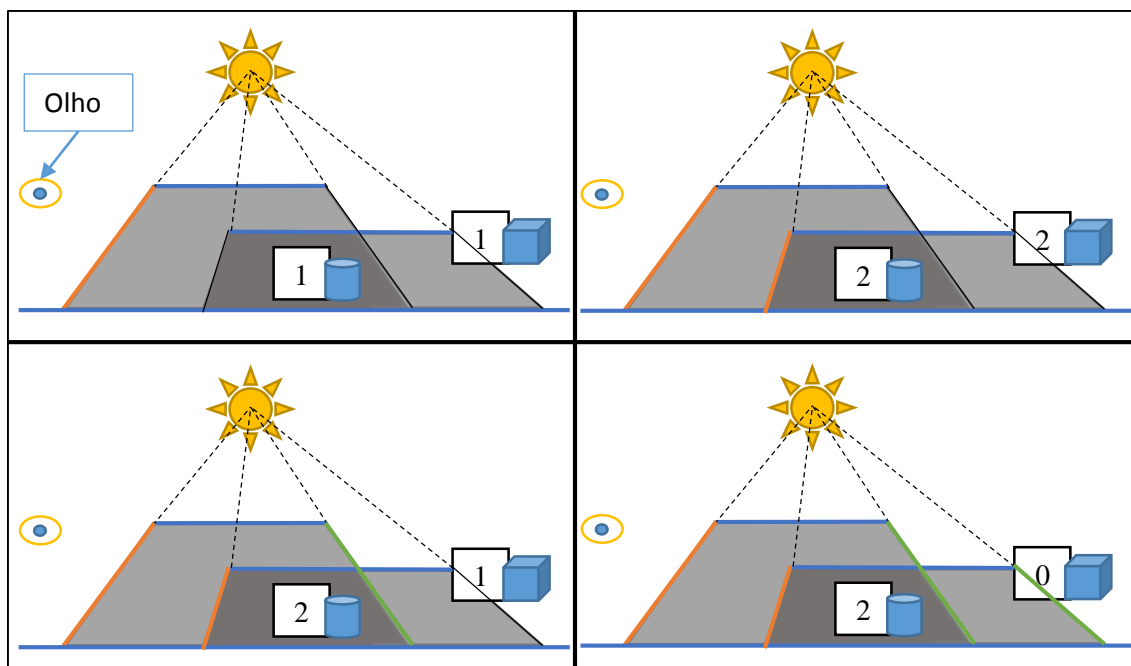


Figura 11 Apresentação da contagem de *shadow maps*. Nas imagens de cima a renderização com *front-facing*, e nas imagens de baixo com *back-facing*.

Após a contagem, a cena é renderizada normalmente, sem utilizar os *shadow volumes*, e para cada pixel, verifica-se qual o número que está no *stencil buffer*. Se for zero, a iluminação desta fonte de luz é aplicada, caso o número seja maior que zero, ele está em uma sombra. Quando houver várias fontes de luz, é possível definir diferentes intensidades para as sombras de acordo com o número definido no *stencil buffer*.

Este método contém um erro, que acontece quando a câmera está dentro de um *shadow volume*. O que ocorre é que quando a contagem for feita, uma ou mais faces podem deixar de serem contadas e assim a sombra não será feita corretamente. Uma maneira de contornar este erro é utilizando o método “Carmack’s Reverse” (Carmack, 2000), que foi criado em 2000 por John Carmack. O método consiste em reverter o processo de contagem, começando a renderizar com o *back-facing* habilitado e somando o valor um no *stencil buffer* e então habilitando somente o *front-facing* e subtraindo o valor um (Figura 12).

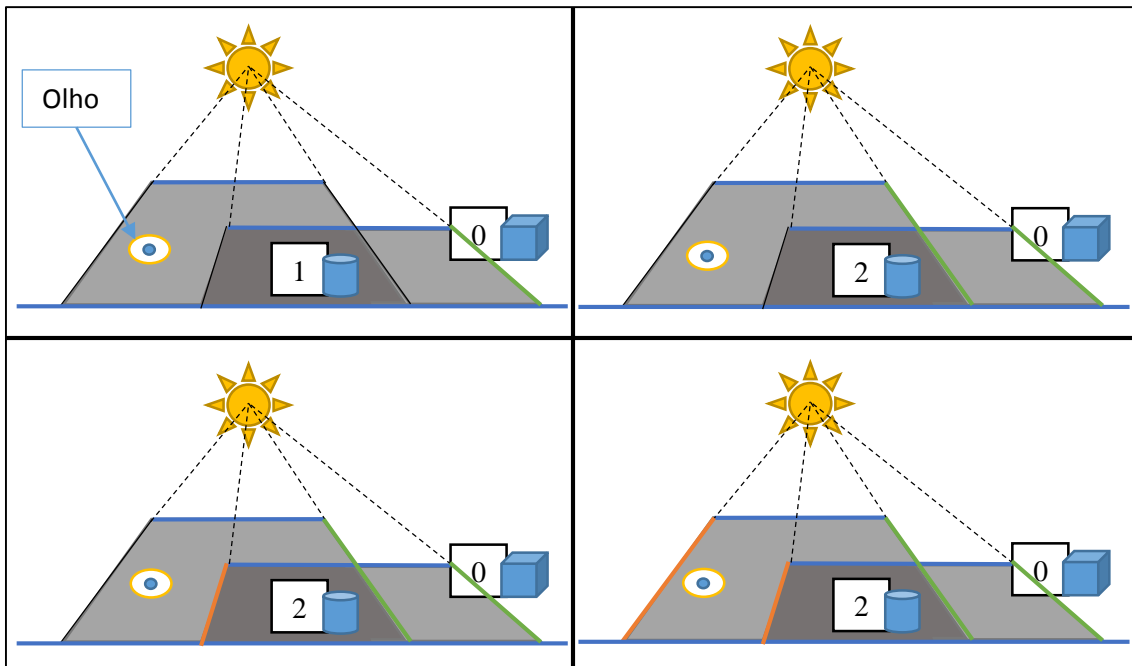


Figura 12 Apresentação do processo de contagem utilizando o método "Carmack's Reverse".

2.2.4 Fake Soft Shadow Maps

É possível converter um *shadow maps* que projetou uma *hard shadow* em um *soft shadow*. Existem inúmeros métodos que fazem isso, o mais comum é utilizar *Gaussian Blur* (Zheng, et al., 2011). Ao renderizar a cena com as sombras, utilizando o *shadow map*, renderiza-se as partes iluminadas em branco, e as sombras em preto. Então aplica-se um *gaussian blur* nesta imagem. Então renderiza-se novamente a cena, com cores, e multiplica-se pela imagem com o *blur*, criando a imagem final, com *soft shadows* (Figura 13).

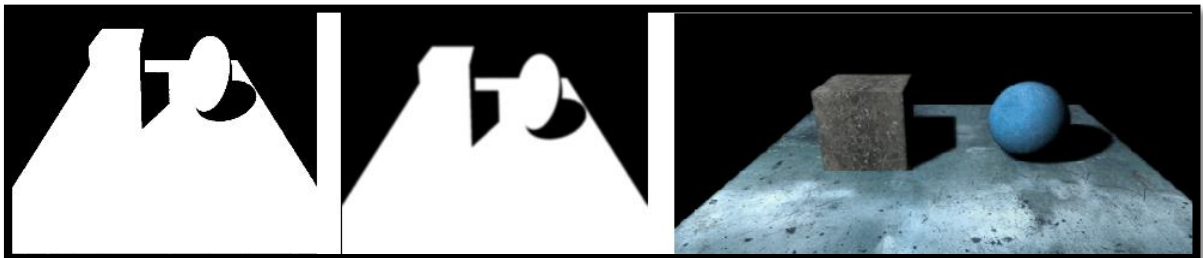


Figura 13 Representação do processo de criação de *Fake Soft Shadows*. A esquerda a imagem renderizada em preto e branco, no meio a imagem da esquerda com aplicação do *gaussian blur*. E a direita o resultado final.

2.2.5 Soft Shadow Volume

Este método funciona baseado no método básico de *shadow volume* (Assarsson, et al., 2003). Primeiramente a umbra da sombra projetada pelo objeto é calculada utilizando *shadow volumes*. Então é calculada as dos triângulos do objeto utilizando um algoritmo de força bruta. Com cada borda, uma nova aresta é criada na direção da antiga borda, porém com tamanho maior que a antiga, relativo à distância entre a fonte luminosa e o objeto. Após isso os *penumbra volumes* são calculados, da mesma maneira que os *shadow volumes*, porém utilizando as novas arestas (Figura 14), assim se tem a área em que a penumbra irá ocorrer.

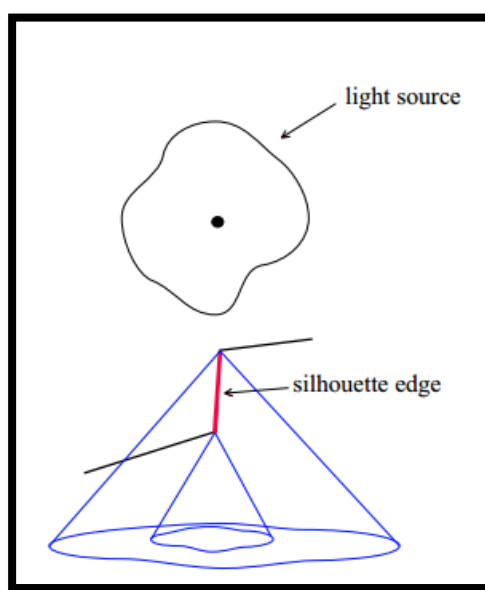


Figura 14 Representação do processo de criação dos penumbra volumes.

Após o cálculo dos *penumbra volumes*, a intensidade da sombra na penumbra é dada em relação a distância entre a borda dos *shadow volumes* que representam a umbra e a borda dos *penumbra volumes*. Quanto mais perto da umbra a sombra fica mais intensa, e quanto mais longe menos.

3. Implementação

O software foi implementado na linguagem C++, fazendo uso da API Direct3D 11. Esta API foi liberada juntamente com o Windows 7 em 2008, onde tinha algumas novidades como suporte a tessellation, renderização por múltiplas threads, e compute shaders. A API Direct3D faz parte do conjunto DirectX da Microsoft, que dá acesso a certas funcionalidades do hardware, como som, imagem ou entrada de mouse e teclado. Em particular, esta API fornece um conjunto de funções que nos permitem interagir com uma placa de vídeo, para que seja possível renderizar cenas utilizando um hardware especializado. Para utilizar a API Direct3D 11 é necessário fazer download do Microsoft DirectX SDK (Microsoft, 2013).

Para que a aplicação fosse fácil de ser modificada e incrementada, foi necessário modularizá-la. Foi criado um módulo que inicializa o Direct3D, um módulo para gerenciar as entradas de mouse e teclado, um módulo que gerencia as partes gráficas, um módulo que faz o carregamento e gerenciamento de modelos 3D, e um módulo que faz o gerenciamento de todos os outros módulos.

O módulo que gerencia a parte gráfica do programa realiza tarefas como: atualização da tela, renderização da cena, definição dos atributos da janela, carregamento e gerenciamento de shaders, entre outras. E é neste módulo que são implementados os métodos de projeção de sombras.

3.1 Implementação de *Shadow Maps*

Como foi explicado no capítulo 2, a projeção de sombras utilizando shadow maps exige que a cena seja renderizada a partir das fontes de luz. Para isso, foram criados dois shaders, um vertex shader e um pixel shader. Shaders são códigos que após compilados são executados em uma placa de vídeo, assim possuindo um desempenho muito superior em relação ao mesmo código sendo executado em um processador. Eles podem possuir dados globais e dados próprios, que após processados podem ser enviados à outros shaders. Para isso foram criadas algumas estruturas de dados, que armazenam as informações necessárias e que facilitam a comunicação de dados entre shaders, que são:

```
struct VertexInputType {
```

```
struct PixelInputType {
```

```

float4 position : POSITION;
float2 tex : TEXCOORD0;
float3 normal : NORMAL;
};

float4 position : SV_POSITION;
float2 tex : TEXCOORD0;
float3 normal : NORMAL;
float4 lightViewPosition : TEXCOORD1;
float3 lightPos : TEXCOORD2;
};

```

Ambas as estruturas recebem a posição do vértice, ou do pixel, as coordenadas de textura, e o vetor normal. A estrutura do pixel shader recebe a posição da fonte de luz e a direção dela.

O vertex shader é executado uma vez para cada vértice que é dado à placa de vídeo. Neste shader são calculadas as posições de cada vértice como se vistos pela fonte de luz. Após o cálculo, os resultados são enviados ao pixel shader.

O pixel shader é executado para cada pixel a ser desenhado na tela, caso mais de um pixel estiver no mesmo lugar, apenas o mais próximo é desenhado, e a distância deste pixel à fonte de luz é salva no depth buffer. Após calcular as distâncias dos pixels mais próximos à fonte luminosa, é necessário calcular a distância do pixel a ser desenhado em relação à fonte de luz. Isto pode ser calculado pela diferença entre coordenada Z do pixel e a coordenada Z da fonte luminosa. Então é verificado se a distância do pixel a ser desenhado ou a distância presente no shadow map é a menor. Caso a distância do pixel for menor, significa que a luz está atrás dele, então ele faz parte de uma sombra, caso contrário ele é iluminado.

A figura 15 apresenta as sombras resultantes deste algoritmo em uma cena. O vertex shader e o pixel shader podem ser vistos nos apêndices A.1 e A.2, respectivamente.

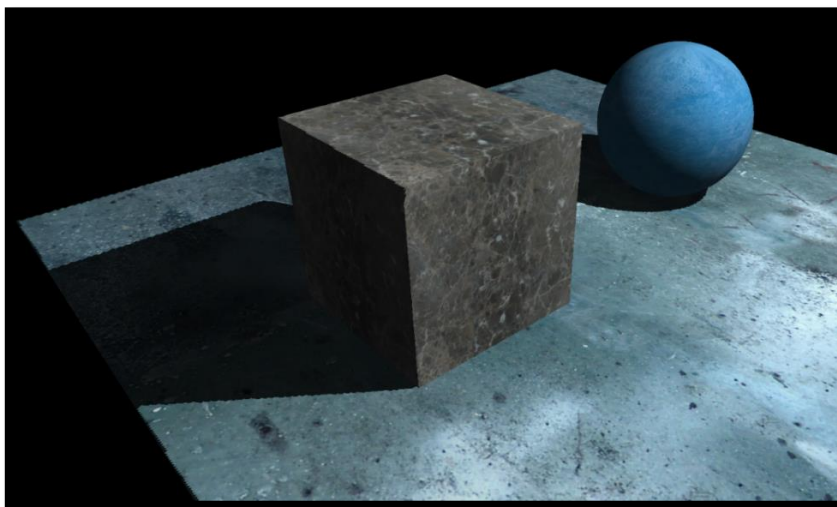


Figura 15 Resultado da implementação do *Shadow Map*.

3.2 Implementação de Shadow Volumes

Para a implementação de shadow volumes, foi criado um pixel shader e dois vertex shaders e foram utilizadas as mesmas estruturas que foram criadas para a implementação de shadow maps.

O primeiro vertex shader calcula o vetor direção entre a luz e o ponto do triângulo, e então estende este ponto ao infinito. Este shader pode ser visto no apêndice B.1. O segundo vertex shader recebe os pontos dos planos calculados pelo primeiro shader, e então renderiza a cena primeiramente utilizando somente back-facing e depois utilizando apenas front-facing. Estas renderizações definem os valores no StencilBuffer, este shader está presente no apêndice B.2.

Após os vertex shader processarem, o pixel shader somente multiplica a cor do pixel pelo valor definido no stencil buffer multiplicado por um décimo. Assim quanto maior o valor definido no stencil buffer, mais escuro será o pixel (Figura 16). O código deste pixel shader é apresentado no apêndice B.3.

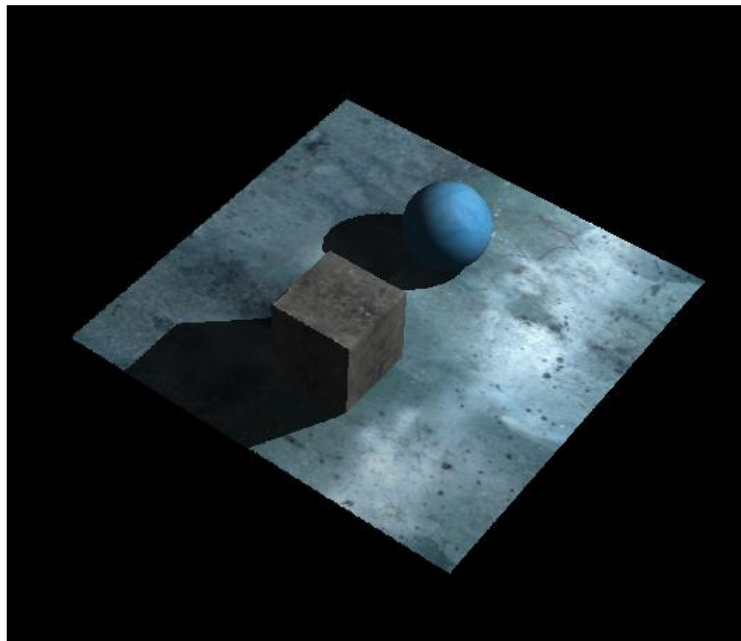


Figure 16 Resultado do *Shadow Volumes*

3.3 Implementação de *Perspective Shadows*

Para implementar perspective shadows, foi criado um pixel shader que simplesmente escurece um pixel, os pixels que são retornados por este shader são utilizados na textura. Também foi criado um vertex shader que realiza os cálculos necessários para transformar as coordenadas da câmera em coordenadas de textura e então calcula as coordenadas da textura no plano.

Ambos os shaders podem ser vistos nos apêndices C.1 e C.2, respectivamente. O resultado do processamento destes shaders pode ser visto na Figura 17.

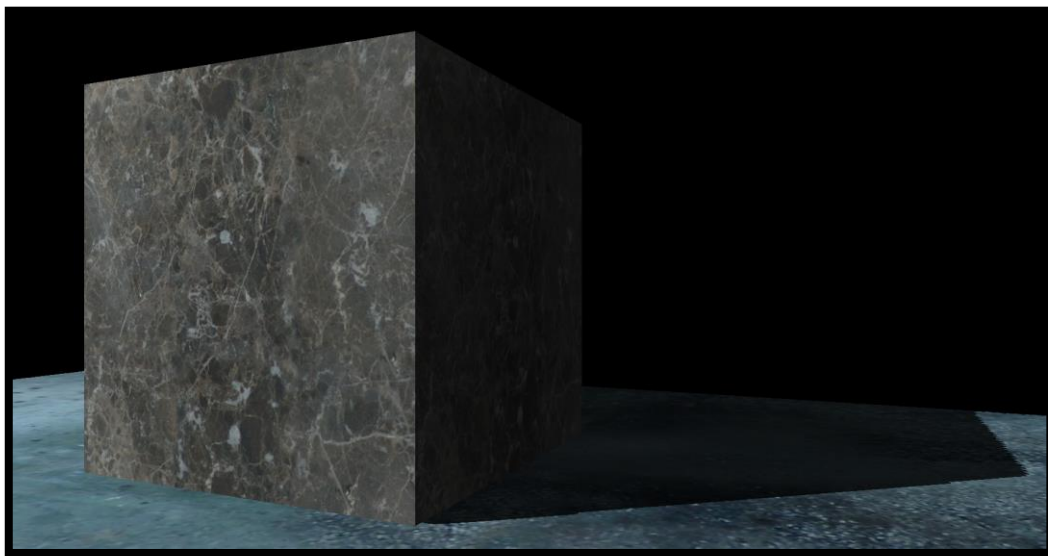


Figure 17 Resultado das sombras utilizando *perspective shadows*.

4. Comparação das Técnicas

Para realizar a comparação das técnicas serão demonstradas algumas situações onde cada técnica tem seu melhor e pior desempenho. Também serão apresentadas algumas vantagens e desvantagens de cada uma, levando em consideração alguns aspectos que todas possuem em comum, que são: complexidade, usabilidade, velocidade de processamento e qualidade gráfica.

4.1 Shadow Maps

Como foi explicado anteriormente, ao se fazer uso de shadow maps é necessário renderizar a cena uma vez para cada fonte de luz presente. Assim, quanto maior for a quantidade de fontes luminosas, pior será o desempenho da aplicação. Sendo assim, uma aplicação que não necessite uma grande qualidade gráfica e que possua poucas fontes de luz é um cenário ideal para a utilização de shadow maps.

4.1.1 Desempenho

A velocidade de processamento da projeção de sombras utilizando shadow maps foi calculada em relação ao FPS (*frames per second*) que é a quantidade de vezes que a tela é capaz de ser renderizada em um segundo. A tabela a seguir ilustra uma média de FPS em uma aplicação que utiliza shadow maps com uma resolução de 800x600, onde a primeira linha mostra o número de luzes utilizado, e a primeira coluna o número de triângulos.

	1	2	3	4	5
10	730	655	563	452	350
100	728	642	560	445	338
1000	725	625	554	439	333
10000	697	617	514	385	305
100000	513	415	328	235	112

4.1.2 Vantagens

- Complexidade: *shadow maps* é uma técnica fácil de entender e fácil de implementar, exige conhecimento de moderado álgebra linear e um básico sobre o funcionamento de shaders.

- **Velocidade de processamento:** é a técnica que possui um desempenho muito bom, porém o desempenho é influenciado por uma série de fatores. Pode-se observar na tabela acima que ao aumentar a quantidade de objetos na cena o desempenho sofre uma perda mínima, porém ao utilizar uma grande quantidade de fontes luminosas o desempenho é bastante prejudicado. A velocidade de processamento também é influenciada pela resolução das texturas utilizadas na aplicação, pois ao se utilizar uma resolução maior para criar o *shadow map*, este possui uma quantidade maior de pixels a serem processados.

4.1.3 Desvantagens

- **Usabilidade:** como é necessário que a fonte luminosa tenha um ponto específico, e que tenha uma direção, esta técnica só pode ser utilizada com fontes de luz do tipo *spot light* ou *directional light*.
- **Qualidade gráfica:** esta técnica tem a sua qualidade gráfica muito limitada a resolução do *shadow map*. Ao se utilizar uma resolução menor que a resolução da aplicação, as sombras ficarão “pixeladas”, porém ao se aumentar a resolução do *shadow map*, o desempenho é prejudicado.

4.2 *Perspective Shadows*

Esta técnica possui um desempenho ótimo, porém só pode ser utilizada em situações onde as sombras serão projetadas em um plano. Uma aplicação que possua paredes e chão planos, onde os objetos não projetam sombras uns nos outros é um cenário ideal para a utilização desta técnica. Caso os objetos projetem sombras uns nos outros, pode se utilizar outra técnica para renderizar estas sombras, e fazer uso de *perspective shadows* para renderizar as sombras nas paredes e chão.

4.2.1 Desempenho

O desempenho da técnica *Perspective Shadow* foi verificado utilizando a mesma abordagem para o cálculo do FPS do *shadow maps*. A tabela a seguir ilustra os dados encontrados.

	1	2	3	4	5
10	780	662	608	442	306
100	768	645	592	421	284
1000	744	630	566	404	265
10000	709	626	520	362	231
100000	580	430	342	207	97

4.2.2 Vantagens

- Complexidade: é uma técnica muito fácil de entender e de implementar, exige um conhecimento básico de álgebra linear e de computação gráfica.
- Velocidade de processamento: dentre as técnicas implementadas, está foi a técnica que teve um desempenho regular em comparação com as outras com o caso mais simples da aplicação. Porém, o desempenho é prejudicado de acordo com a quantidade de objetos na cena, de acordo com a quantidade de fontes de luz e de acordo com a resolução da aplicação.

4.2.3 Desvantagens

- Usabilidade: esta técnica só pode ser utilizada quando se está projetando sombras em planos, assim a grande maioria das aplicações não podem fazer uso de *perspective shadows*.
- Qualidade gráfica: a qualidade gráfica é dependente da qualidade da textura utilizada no plano. A única situação onde as sombras projetadas são próximas da realidade é ao se fazer uso de resoluções muito grande

para a textura do plano, estas resoluções prejudicam bastante o desempenho da aplicação.

4.3 *Shadow Volumes*

Esta técnica é possui uma qualidade excelente, e pode ser facilmente utilizada com uma grande quantidade de fonte luminosas, porém é a técnica com o pior desempenho em velocidade de processamento de todas as técnicas implementadas. Um cenário ideal para esta técnica seria uma aplicação que necessite de uma qualidade gráfica ótima, e que seria utilizada em computadores melhores.

4.3.1 Desempenho

A tabela a seguir mostra a relação da média de FPS ao renderizar sombra utilizando *shadow volumes* em diversas situações.

	1	2	3	4	5
10	1020	904	485	366	256
100	980	883	455	302	222
1000	906	765	364	284	189
10000	842	694	482	238	134
100000	754	607	398	167	75

4.3.2 Vantagens

- Velocidade de processamento: como pode ser visto nas tabelas acima, dentre as técnicas implementadas, esta foi a que teve o pior desempenho. O desempenho desta técnica é bastante prejudicado de acordo com a quantidade de fontes luminosas, e de acordo com o número de triângulos a serem renderizados.
- Usabilidade: esta técnica pode ser utilizada em qualquer tipo de aplicação, porém como possui um desempenho inferior às outras, necessita de um *hardware* mais avançado para que se possa ser bem utilizada.

- Qualidade gráfica: a qualidade gráfica das sombras projetadas utilizando *shadow volumes* é excelente, as sombras são sempre projetadas de acordo com a resolução da aplicação. Ao contrário das outras técnicas implementadas, a qualidade gráfica desta técnica não é prejudicada pela qualidade da resolução das texturas.

4.3.3 Desvantagens

- Complexidade: esta técnica é mais difícil de entender que as outras e é bem complexa de ser implementada. Exige um conhecimento moderado de álgebra linear e um conhecimento avançado sobre *shaders*.

5. Conclusão

Este trabalho teve como objetivo o estudo de diversas técnicas de projeção de sombras e a implementação das técnicas básicas, com a finalidade de realizar uma comparação entre as técnicas e ressaltar as vantagens e desvantagens de cada uma, bem como situações onde cada uma tem seu melhor e pior desempenho.

O objetivo de realizar a implementação dos algoritmos básicos de projeção de sombras foi alcançado. Pode ser realizada uma comparação entre todos os algoritmos implementados e analisados os pontos positivos e negativos de cada um, bem como situações ideais para a sua utilização.

5.1 Contribuições Alcançadas

Neste trabalho foi possível realizar uma comparação a respeito dos algoritmos básicos de projeção de sombras. Para cada técnica implementada foi realizada uma análise dos pontos positivos e negativos, e foram comparados alguns aspectos que todas possuíam em comum. Também foram descritas situações em que cada técnica tem o seu melhor e pior desempenho.

Este trabalho pode servir como ponto de partida para a escolha da melhor técnica de renderização de sombra a ser utilizada em uma aplicação. Pois oferece vários detalhes sobre cada uma e o que cada uma tem de melhor ou pior em relação as outras.

6.2 Trabalhos Futuros

A implementação das variações das técnicas básicas de projeção de sombras, que visam a projeção de *soft shadows* seria um tópico de interesse. Pois seria possível analisar e comparar os pontos fracos e fortes de técnicas que se aproximam mais da realidade.

REFERÊNCIAS

- Assarsson Ult e Akenine-Möller Tomas** A Geometry-based Soft Shadow Volume Algorithm, ACM, 2003.
- Carmack John** Method for rendering shadows using a shadow volume and a stencil buffer [Patente]. - 2000.
- Crow Frank** Shadow Algorithms for Computer Graphics, Siggraph. - 1977.
- Eisemann Elmar [et al.]** Efficient Real-Time Shadows , Siggraph, 2012.
- Hasenfratz J M [et al.]** A Survey of Real-Time Soft Shadow Algorithms , Computer Graphics Forum, 2003. - 4 : Vol. 18.
- Heidmman Tim** Real shadows, real time , Iris Universe. - 1991. - Vol. 18.
- Mamassian Pascal, Knill David C and Kersten Daniel** The perception of cast shadows , Trends in cognitive sciences. - 1998.
- Microsoft** Developing games for windows [Online]. - Microsoft, 2013. - acessado em 14 de 12 de 2013. - <http://msdn.microsoft.com/library/windows/apps/hh452744.aspx>.
- Microsoft** Direct3D 11 Graphics , Microsoft, 2010.
- Nvidia** NVIDIA MENTAL RAY [Online] // NVIDIA. - Nvidia, 2012. - acessado em 14 de 12 de 2013. - <http://www.nvidia-arc.com/products/nvidia-mental-ray/nvidia-mental-ray.html>.
- Segal Mark [et al.]** Fast Shadows and Lighting Effects Using Texture Mapping , Siggraph. - 1992.
- Stamminger Marc e Drettakis George** Perspective Shadow Maps , Siggraph, 2002.
- Whitted Turner** An Improved Illumination Model for Shaded Display ,ACM, 1980.
- Willians Lance** Casting Curved Shadows on Curved Surfaces. - 1978.
- Zheng Zhongxiang e Saito Suguru** Screen Space Anisotropic Blurred Soft Shadows , Siggraph. - 2011.

APÊNDICE A – IMPLEMENTAÇÃO DE SHADOW MAPS

```

PixelInputType ShadowVertexShader(VertexInputType input)
{
    PixelInputType output;
    float4 worldPosition;

    input.position.w = 1.0f;

    //Transforma as posições dos vertices como se vistos pela luz
    output.position = mul(input.position, worldMatrix);
    output.position = mul(output.position, viewMatrix);
    output.position = mul(output.position, projectionMatrix);

    output.lightViewPosition = mul(input.position, worldMatrix);
    output.lightViewPosition = mul(output.lightViewPosition, lightViewMatrix);
    output.lightViewPosition = mul(output.lightViewPosition,
lightProjectionMatrix);

    output.tex = input.tex;

    output.normal = mul(input.normal, (float3x3)worldMatrix);
    output.normal = normalize(output.normal);

    worldPosition = mul(input.position, worldMatrix);

    output.lightPos = lightPosition.xyz - worldPosition.xyz;

    output.lightPos = normalize(output.lightPos);

    return output;
}

```

Código A.1 – Vertex Shader

```

float4 ShadowPixelShader(PixelInputType input) : SV_TARGET {
    float bias = 0.001f; //erro
    float4 color = ambientColor;
    float2 projectTexCoord;
    float depthValue;
    float lightDepthValue;
    float lightIntensity;
    float4 textureColor;

    //calcula a projeção das texturas como se vistas pela luz
    projectTexCoord.x = input.lightViewPosition.x / input.lightViewPosition.w
/ 2.0f + 0.5f;
    projectTexCoord.y = -input.lightViewPosition.y / input.lightViewPosition.w
/ 2.0f + 0.5f;

    //verifica se o pixel está no campo de visão da luz
    if ((saturate(projectTexCoord.x) == projectTexCoord.x) &&
(saturate(projectTexCoord.y) == projectTexCoord.y)) {
        //calcula as distancias do pixel em relacao a luz
        depthValue = depthMapTexture.Sample(SampleTypeClamp,
projectTexCoord).r;
        lightDepthValue = input.lightViewPosition.z /
input.lightViewPosition.w;
        lightDepthValue = lightDepthValue - bias;
    }
}

```

```

        //verifica qual pixel está mais proximo
        if (lightDepthValue < depthValue) {
            lightIntensity = saturate(dot(input.normal, input.lightPos));

            if (lightIntensity > 0.0f) {
                color += (diffuseColor * lightIntensity);

                color = saturate(color);
            }
        }
    }
    textureColor = shaderTexture.Sample(SampleTypeWrap, input.tex);
    color = color * textureColor;

    return color;
}

```

Código A.2 – Pixel Shader

```

VertexInputType CreateTrianglePlane(VertexInputType input)
{
    VertexInputType output;

    float3 LightDir = normalize(input.position - LightPos);
    //calcula o vetor direção
    output.position = g_fExtrudeBias*LightDir;
    //fingir que vai ao infinito

    return output;
}

```

Código B.1 – Vertex Shader

```

TriangleStream<PixelInputType> CalculateShadowVolumes(triangleadj VertexInputType
In[6])
{
    TriangleStream<PixelInputType> ShadowTriangleStream;
    // Calcular a normal do triangulo
    float3 normal = cross(In[2].pos - In[0].pos, In[4].pos - In[0].pos);

    //verificar se a luz pode ser vista
    if (dot(normal, LightDir) > 0.0f)
    {
        PixelInputType p;
        //front-facing
        for (int v = 0; v<6; v += 2)
        {
            p.pos = mul(float4(In[v].position, 1), g_mViewProj);
            ShadowTriangleStream.Append(p);
        }
        ShadowTriangleStream.RestartStrip();

        //back-facing
        for (int v = 4; v >= 0; v -= 2)
        {

```

```

        p.pos = mul(float4(In[v].position, 1), g_mViewProj);
        ShadowTriangleStream.Append(p);
    }
    ShadowTriangleStream.RestartStrip();
}
}

```

Código B.2 – Vertex Shader

```

float4 SVPixelShader(PixelInputType input) : SV_Target
{
    return color * ambientColor * textureColor * float4(ShadowColorBuffer.xyz,
0.1f);
}

```

Código B.3 –Pixel Shader

```

float4 PSPixelShader(PixelInputType input) : SV_Target
{
    return color * ambientColor * textureColor * 0.1f;
}

```

Código C.1 –Pixel Shader

```

PixelInputType PSPixelShader(float4x4 M, float4x4 V, float4x4 P, VertexInputType
T) : SV_Target
{
    PixelInputType output;
    output.position = mul(M * V * P, T);
    return output;
}

```

Código C.2 –Pixel Shader