Geração de Sombras em Tempo Real Usando Árvores BSP e Buffers Estênceis

Harlen Costa Batagelo
Ilaim Costa Júnior

UNOESC – Universidade do Oeste de Santa Catarina
Campus de Videira
Rua Paese, 198, Bairro das Torres, 89560-000, Videira, SC, Brasil
{harlen, ilaim}@unoescvda.rct-sc.br

Resumo

Este artigo descreve um algoritmo de geração de sombras em tempo real para ambientes poligonais estáticos iluminados por fontes de luzes pontuais móveis. Inicialmente a cena é pré-processada por um algoritmo de particionamento espacial binário, gerando uma estrutura de dados (árvore BSP) que, em tempo de execução, obtém os polígonos geradores de sombras com relação às fontes de luzes. A partir desses polígonos, volumes de sombras são gerados e desenhados no buffer estêncil, formando assim as regiões de sombras numa base pixel por pixel. Outrossim, é utilizado um algoritmo de otimização de modo a reduzir o número de polígonos de sombra redundantes, tornando possível a obtenção de taxas de exibição em tempo real (≅30 Hz) para instâncias de tamanho modesto, de acordo com resultados práticos obtidos de computadores pessoais usando hardware gráfico 3D disponível atualmente.

Palavras-chave: Sombras, Árvores BSP, Buffers Estênceis.

1 Introdução

O cálculo de sombras corresponde à definição de umbras (áreas não-iluminadas) e penumbras (áreas parcialmente iluminadas) formadas da oclusão de fontes de luzes por objetos componentes de um ambiente tridimensional, e tem sido um problema clássico na computação gráfica 3D. A presença de sombras proporciona ao observador uma melhor compreensão do relacionamento espacial entre os objetos, aumentando a sensação de realidade e coerência dada pela imagem. Além disso, sua utilização torna-se imprescindível em diversas aplicações, como em síntese de imagens fotorrealistas, planejamento arquitetural e avaliação de impactos ambientais. Métodos de iluminação global, como o traçado de raios [Whitted (1980)] e a radiosidade [Goral et al. (1984)], têm sido usados com satisfatória eficácia na solução deste problema. Há, entretanto, dificuldade em se encontrar métodos de geração de sombras para aplicações interativas e em tempo real, como realidade virtual e jogos, devido ao alto custo computacional envolvido. Tal dificuldade é especialmente verdadeira quando consideramos uma arquitetura de computadores pessoais. Ademais, os métodos mais utilizados atualmente possuem severas limitações, como a projeção de sombras somente em superfícies planas ou a impossibilidade de manipulação de fontes de luzes omnidirecionais.

Neste trabalho propomos um algoritmo híbrido de geração de sombras em tempo real, através da combinação do método de renderização de sombras volumétricas usando buffers estênceis [Heidmann (1991)][Kilgard (1997)] e um algoritmo de determinação da ordem de visibilidade de polígonos usando árvores BSP [Fuchs et al. (1980)]. Por razões inerentes aos métodos utilizados, o algoritmo sugerido trabalha somente em ambientes poligonais estáticos, mas onde o observador e as fontes de luzes se movem arbitrariamente em tempo real. Com o auxílio de aceleração 3D em hardware disponível atualmente em computadores pessoais, torna-se possível a obtenção de taxas de exibição superiores a 30 quadros por segundo, considerando cenas com um número modesto de polígonos (em torno de 1.000 polígonos).

1.1 Organização

Este trabalho está organizado da seguinte forma: a seção 2 apresenta uma breve taxionomia acerca dos algoritmos de geração de sombras conhecidos atualmente. Na seção 3 é introduzida a idéia de árvores BSP. A definição de buffers estênceis, bem como suas aplicações em renderização de sombras volumétricas, é dada na seção 4. A seguir, nas seções 5, 6 e 7, descrevemos o algoritmo híbrido proposto e os aprimoramentos sugeridos. Os resultados e conclusões dos

testes de desempenho obtidos da implementação do algoritmo são dados nas seções 8, 9 e 10.

2 Algoritmos de Geração de Sombras

Os métodos de geração de sombras foram inicialmente classificados por Franklin Crow [Crow (1977)], que distinguiu três aproximações para o problema conhecido até então.

A primeira classe envolve algoritmos de geração de sombras no momento da varredura da imagem. Os limites dos polígonos geradores de sombra são projetados sobre o polígono que está sendo processado, e a cor do pixel é alterada sempre que há uma transição entres tais limites [Appel (1968)][Bouknigh—Kelley (1970)].

A segunda classe envolve algoritmos que efetuam duas passagens através de um algoritmo de verificação de superfícies ocultas. O primeiro passo consiste em obter todas as superfícies visíveis com relação à posição da fonte de luz, dividindo em fragmentos os polígonos que se encontram parcialmente visíveis. As cores são designadas para cada polígono ou fragmento, e um segundo passo é efetuado, mostrando a cena do ponto de vista do observador [Atherton et al. (1978)].

A terceira classe introduz o conceito de volumes de sombras: para cada aresta de cada polígono encarando a fonte de luz, cria-se um polígono de sombra. Juntos, esses polígonos formam volumes de sombras, os quais são adicionados ao ambiente como conjuntos de polígonos invisíveis, marcando a transição entre áreas iluminadas e não-iluminadas durante o processo de renderização.

Em análise ulterior, Bergeron [Bergeron (1986)] e Woo et al.[Woo et al. (1990)] estenderam o conceito de volumes de sombras de Crow, introduzindo outros mais, como a utilização de mapas de sombras com *z-buffers* [Willians (1978)], o traçado de raios [Whitted (1980)][Appel (1968)] e a radiosidade [Goral et al. (1984)].

O algoritmo de mapas de sombras com *z-buffers*, introduzido por Willians [Willians (1978)], é um dos métodos de geração de sombras mais utilizados atualmente em aplicações interativas. O algoritmo faz uso de buffers de profundidade conhecidos como *z-buffers*, os quais são designados para cada fonte de luz, de modo que cada ponto visível à uma fonte de luz corresponde a outro ponto no *z-buffer* correspondente. Durante a renderização, o *z-buffer* de cada fonte de luz é utilizado como um mapa de texturas, de sorte que a sombra é obtida através da comparação do valor de profundidade deste mapa com o valor de profundidade do pixel que está sendo desenhado.

A qualidade das sombras é diretamente proporcional à resolução do *z-buffer*, significando que a disponibilidade de memória é um fator importante a ser considerado. Mesmo assim este algoritmo tem sido extensivamente utilizado na prática, em virtude da facilidade de implementação e obtenção de resultados satisfatórios. Por outro lado, este método não é verdadeiramente um algoritmo de geração de sombras em tempo real, porquanto os mapas não são calculados em tempo real, mas somente renderizados. Além disso, há uma certa dificuldade em se manipular fontes de luzes omnidirecionais, pois neste caso são necessários vários mapas de sombras.

Considerando os métodos de geração de sombras citados acima, distinguimos dois grupos principais: os modelos de iluminação global e iluminação local. O primeiro envolve algoritmos com a capacidade de calcular inter-reflexões especulares e/ou difusas das luzes entre as superfícies do ambiente, resultando na simulação correta das propriedades físicas da luz sobre os objetos, como efeitos atmosféricos e iluminação indireta. Tais métodos têm a desvantagem de serem extremamente custosos em termos de eficiência, não sendo possível sua utilização em tempo real. Encontram-se dentro deste modelo o traçado de raios e a radiosidade, usados na síntese de imagens fotorrealistas. Já o modelo de iluminação local envolve algoritmos onde o cálculo do sombreamento de uma superfície independe do sombreamento das superfícies vizinhas. Modelos de iluminação local não representam sombras fisicamente corretas, mas têm como vantagem a eficiência, sendo extensivamente utilizados em aplicações em tempo real. Freqüentemente estes também são chamados de métodos de sombras ríspidas (do inglês *hard shadows*), pois normalmente se utilizam de luzes que não produzem penumbra, como fontes de luzes pontuais ou direcionais.

O algoritmo híbrido sugerido neste trabalho está incluído no modelo de iluminação local, e pode ser classificado dentro do modelo de Crow como uma combinação do algoritmo de volumes de sombras com o método de duas passagens através de um algoritmo de verificação de superfícies ocultas.

3 Árvores BSP

O algoritmo da árvore de particionamento espacial binário, ou simplesmente árvore BSP (do inglês *Binary Space Partitioning*), foi idealizado por Fuchs, Kedem e Naylor [Fuchs et al. (1980)], baseado no trabalho de Schumacker [Schumacker (1969)]. Árvores BSP fornecem um meio elegante e eficiente de determinar a ordem da visibilidade de grupos estáticos de polígonos vistos de um ponto de vista arbitrário. São extensivamente utilizadas na prática, graças a sua capacidade de ordenar em tempo linear os polígonos necessários para a varredura da imagem, além de remover superfícies ocultas e lidar com casos que não seriam possíveis usando somente o algoritmo do pintor [Foley et al. (1990)], como polígonos circulares sobrepostos. Além disso, são freqüentemente empregadas em aplicações de CSG, aceleração de traçado de raios e detecção de colisão.

Árvores BSP representam subdivisões recursivas de espaços convexos n-dimensionais por "hiperplanos". Define-

se hiperplano como um objeto de n-1 dimensões, utilizado para dividir o espaço em duas partes. Por exemplo, considerando um espaço tridimensional, o hiperplano será um plano. Em duas dimensões o hiperplano será uma linha.

A árvore BSP é normalmente construída durante um estágio de pré-processamento, visto que, dependendo da cena processada e da heurística utilizada, sua construção pode levar muito tempo (vários minutos em um computador serial atual). A estrutura resultante, em forma de árvore binária, pode ser percorrida numa inordem (ordem simétrica) especial, estabelecendo em tempo linear a ordem correta da visibilidade dos polígonos sob um ponto de vista arbitrário. Destarte, árvores BSP tornam-se ideais para aplicações onde os objetos são todos estáticos mas o ponto de vista se move arbitrariamente em tempo real.

3.1 Algoritmo

Inicialmente escolhe-se um hiperplano para atuar como a raiz da árvore e dividir o espaço em dois grupos. Em 3D, este plano de partição é obtido da equação do plano de um polígono qualquer da cena, ou de um polígono escolhido de acordo com uma heurística. Os polígonos restantes são classificados com relação ao vetor normal deste plano. Um grupo conterá todos os polígonos que estão no lado da frente do plano de partição, e o outro grupo conterá os que estão atrás. Polígonos que se encontram em ambos os lados, isto é, que estão interceptando o plano de partição, são divididos por este, tendo os seus fragmentos adicionados aos grupos correlatos. Em seguida, cada grupo escolhe um novo plano de partição, dividindo o espaço da mesma maneira, recursivamente, até cada grupo não conter mais nenhum polígono adicional. Com este procedimento fica definida a estrutura em forma de árvore binária que reflete o relacionamento espacial dos polígonos do ambiente. Tal estrutura contém informações suficientes para, por exemplo, determinar os polígonos visíveis em relação a uma posição qualquer, além da prioridade desta visibilidade. Obviamente, a árvore BSP deve ser reconstruída sempre que houver alterações no relacionamento espacial entre os objetos da cena. A figura 1 ilustra uma árvore BSP construída a partir de um cenário bidimensional.

Os planos de partição podem ser selecionados por diferentes critérios de otimização. Em grande parte das aplicações é desejável a obtenção de uma árvore balanceada, isto é, com um número aproximadamente igual de polígonos em cada ramificação. Uma árvore perfeitamente balanceada pode ser construída com o auxílio de planos de partição fictícios, adicionados à árvore como polígonos invisíveis que dividem o espaço em duas metades iguais. A desvantagem deste método reside no fato do plano escolhido poder provocar a divisão de muitos outros polígonos, aumentando o número de nós da árvore de uma maneira indesejada. Por outro lado, a escolha exclusiva de planos que não dividam nenhum polígono pode acarretar na geração de uma árvore excessivamente não-balanceada. Um exemplo típico deste problema ocorre no tratamento de poliedros convexos, quando então a árvore se degenera em uma mera lista encadeada. De fato, a construção de árvores BSP ótimas é considerada um problema NP-completo. Por isso, na prática, há sempre uma troca entre a minimização do número de divisões de polígonos e o balanceamento da árvore, de acordo com o tipo de aplicação.

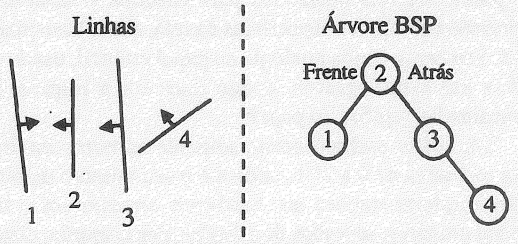


Figura 1 – Grafo de uma árvore BSP construída a partir de um cenário 2D. As setas representam a direção do vetor normal, que por sua vez indica o lado da frente de cada linha.

Depois de construída, a árvore pode ser percorrida de acordo com um procedimento recursivo de modo a obter a ordem correta de visibilidade dos polígonos com relação a um ponto de vista arbitrário. Como no algoritmo do pintor, os polígonos mais distantes podem ser exibidos primeiro, numa ordem de trás para frente. Isso é feito da seguinte maneira: no caso do observador estar na frente do plano de partição da raiz da árvore, o algoritmo mostra primeiro todos os polígonos que estão no grupo de trás, depois o polígono-raiz, seguido pelos polígonos do grupo da frente. Se o observador está atrás do plano de partição, os polígonos do grupo da frente são mostrados primeiro, seguido pelo polígono-raiz, e então pelos polígonos de trás do plano. Este mesmo procedimento é executado recursivamente dentro de cada grupo, até se chegar às folhas da árvore. Invertendo os passos descritos, os polígonos são exibidos na ordem contrária, isto é, de frente para trás, o que pode ser desejável em determinadas aplicações.

As superfícies ocultas, isto é, aquelas na qual o vetor normal forma um ângulo maior que 90° com o observador, podem ser removidas apenas deixando de mostrar os polígonos nos casos em que o observador está em um grupo de polígonos de trás. Pseudocódigos e extensões do algoritmo da árvore BSP podem ser encontrados no BSP FAQ [Wade (1996)], disponível na Internet.

3.2 Eficiência

Considerando uma árvore construída para a tarefa de determinação da ordem de visibilidade de polígonos e/ou remoção de superfícies ocultas, a complexidade assintótica

superior, tanto de tempo como de espaço, é de $O(n^2)$ para n polígonos originais. Entretanto isso ocorre somente em casos excepcionais como, por exemplo, quando todos os polígonos estão sendo interceptados por planos de partição. Assim, na maioria dos casos práticos, espera-se uma eficiência de $O(n)$.

4 Buffers Estênceis

Buffers estênceis, inicialmente implementados na API OpenGL, são planos de bits utilizados para habilitar e desabilitar a impressão em regiões específicas da tela, numa base pixel por pixel. Seu nome é uma analogia ao papel estêncil, usado como matriz em mimeógrafos, e que mascara a impressão de determinadas regiões do papel.

Um buffer estêncil aceita somente números inteiros na faixa inclusiva de 0 à 2^n-1, onde n é o seu número de bits. As atuais implementações em hardware armazenam o buffer estêncil em bits reservados do *z-buffer*. Por exemplo, dispondo 15 bits para o *z-buffer* e 1 bit para o buffer estêncil (totalizando 16 bits), ou 24 bits para o *z-buffer* e 8 bits para o buffer estêncil (totalizando 32 bits).

As primitivas (pontos, linhas e polígonos) são aplicadas no buffer estêncil da mesma maneira que normalmente desenhamos na tela. Por outro lado, o buffer estêncil não armazena os valores RGB dos pixels, mas valores designados de acordo com operações aritméticas simples definidas previamente, tais como "incrementar o valor do buffer estêncil", "decrementar", "inverter bit a bit", "trocar por uma constante (um valor de referência)", entre outras. A operação escolhida é aplicada de acordo com a avaliação de uma condição que deve existir entre o valor atual do pixel do buffer estêncil e um valor de referência. Tal condição é semelhante à do teste de um *z-buffer*, porém de maneira mais extensa, podendo determinar ações como, por exemplo, "aplicar a operação definida se o valor de referência é maior do que o valor do buffer estêncil", ou "se maior ou igual", "se menor", "menor ou igual", etc.

Como visto, a condição do teste do buffer estêncil é executada usando um valor de referência e o valor atual existente no buffer estêncil. Contudo, antes da avaliação, ambos os valores deverão ser filtrados por uma máscara de bits. Esta máscara contém um valor designado pelo usuário, usado para efetuar uma operação de interseção booleana (*AND*) com o valor de referência e com o valor atual do buffer. Em pseudocódigo, isto pode ser escrito como:

```
If ((ref AND mask) condition (stencil AND mask))
    Apply_Operation(stencil);
```

Onde *ref* é o valor de referência, *stencil* é o valor atual do pixel do buffer estêncil, e *mask* a máscara de bits. Se a condição passar, a operação definida é aplicada ao valor atual do buffer estêncil, e ignorada em caso contrário. Como podemos observar, o buffer estêncil aceita uma vasta gama de operações, de sorte que podemos formular ações como "trocar o valor atual do buffer estêncil pelo valor de referência se este for maior do que aquele" ou mesmo "incrementar o valor atual do buffer estêncil se o primeiro bit do valor de referência é igual ao primeiro bit do valor atual do buffer estêncil".

Após a aplicação das primitivas usando as condição definidas, o buffer estêncil pode ser utilizado para mascarar a impressão de áreas específicas do *back buffer*, como, por exemplo, habilitar a impressão de pixels somente se o valor correspondente no buffer estêncil for igual a um valor dado.

4.1 Volumes de Sombras com Buffers Estênceis

Efeitos especiais de dissolução de imagens, desaparecimento gradual e detecção de bordas podem ser facilmente obtidos usando buffers estênceis. Entretanto, um efeito particularmente interessante é o de renderização de sombras usando volumes de sombras, ou simplesmente renderização de sombras volumétricas, idealizada pela Silicon Graphics [Heidmann (1991)][Kilgard (1997)].

Um volume de sombra é um poliedro formado por polígonos invisíveis semi-infinitos, projetados de cada lado de cada polígono gerador de sombra. Tais polígonos semi-infinitos são quadriláteros conhecidos como polígonos de sombra. Dois de seus vértices correspondem aos vértices pertencentes ao polígono gerador de sombra. Os restantes apontam para o infinito, na direção da sombra projetada, mas tornando-se finitos após serem cerceados pelos limites do volume de visão em coordenadas homogêneas.

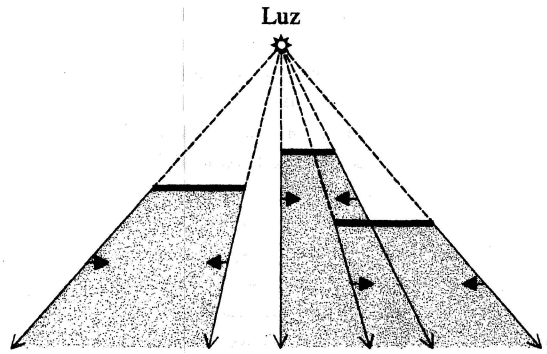


Figura 2 – Volumes de sombras representados em 2D. As setas sólidas indicam a direção do vetor normal de cada polígono de sombra.

Os volumes de sombras usam um princípio de paridade em que os seus polígonos de sombra, com *z-buffering* ativado,

podem ser empregados para demarcar a transição entre áreas iluminadas e não-iluminadas da imagem, usando o vetor normal para indicar o lado da sombra. A figura 2 ilustra volumes de sombras em duas dimensões.

Com *z-buffering* ativado, a cena original (sem sombras) é renderizada. Em seguida, os polígonos de sombra que não estão encarando o observador são utilizados para incrementar (adicionar em uma unidade) o buffer estêncil, que inicialmente tem todos os seus pixels zerados. Os polígonos de sombra restantes, isto é, aqueles que estão encarando o observador, são aplicados da mesma forma, mas agora usando a operação de decrementar (subtraindo em uma unidade). Como resultado, as regiões do buffer estêncil com pixels diferentes de zero corresponderão aos pixels sombreados na imagem original.

O algoritmo de renderização de sombras volumétricas pode ser resumido nos seguintes passos:

1. Construir volumes de sombras para os polígonos que estão encarando a fonte de luz;
2. Utilizando *z-buffering*, renderizar a cena original, isto é, sem sombras;
3. Desenhar no buffer estêncil os polígonos de sombra que não estão encarando o observador, usando a operação de incrementar;
4. Desenhar no buffer estêncil os polígonos de sombra que estão encarando o observador, usando a operação de decrementar;
5. Preencher o *back buffer* com a cor da sombra, usando o buffer estêncil como máscara. Os pixels do buffer estêncil contendo valores diferentes de zero correspondem aos pixels do *back buffer* que estão na sombra.

Uma aspecto interessante deste algoritmo reside na propriedade de conservação da geometria da cena, visto que as sombras são desenhadas somente após a renderização da cena original. Isso também possibilita a interação das sombras com objetos adicionados posteriormente ao processo de geração dos volumes de sombras.

4.2 Modos de Renderização de Sombras

Uma vez pronto o buffer estêncil para a renderização, todos os pixels da imagem original que correspondam a pixels do buffer estêncil com valores diferentes de zero, deverão ser modificados com a cor da sombra. Este procedimento pode ser executado através da aplicação de um grande polígono encobrindo toda a janela de visão, usando o buffer estêncil como máscara, e com a cor deste polígono determinando a cor da sombra. Se a iluminação ambiente for nula e somente uma fonte de luz estiver sendo usada, um polígono de cor preta será suficiente. Caso contrário, deverão ser utilizados polígonos translucentes com a devida cor ambiente. Tais polígonos proporcionam melhores resultados visuais, além de possibilitarem a composição de sombras, isto é, a exibição da combinação de diferentes intensidades de sombras geradas por várias fontes de luzes. Por outro lado, o nível de translucência (componente alfa de uma superfície RGBA) é normalmente escolhido de maneira empírica, o que pode resultar em iluminação inconsistente. Isto pode ser percebido, por exemplo, quando um objeto que reflete luz especular está sendo encoberto por um volume de sombra.

5 Algoritmo Híbrido

O algoritmo sugerido combina os métodos apresentados nas seções 3 e 4. Inicialmente constrói-se a árvore BSP da cena num estágio de pré-processamento intensivo. Em tempo de execução, a árvore BSP é percorrida, obtendo o conjunto de polígonos visíveis com relação à fonte de luz. Tais polígonos se tornam os geradores dos volumes de sombras [Chin—Feiner (1989)][Slater (1992)]. Este estágio pode ainda ser substituído ou combinado com outros algoritmos que possibilitem a determinação da visibilidade dos polígonos em tempo de execução. Existe, todavia, uma restrição quanto ao modo de apresentação das informações de tais polígonos, porquanto estes devem estar representados como índices de uma *array* de vértices. Felizmente esta é a representação padrão de cenas em programas de modelagem 3D, e é aqui necessária para preservar as relações de adjacência entre os polígonos.

Considerando os polígonos geradores de sombra obtidos da árvore BSP, um algoritmo de otimização (explicado a seguir na seção 6) é empregado na detecção das arestas adjacentes, marcando-as como "invisíveis". Polígonos de sombra são gerados para cada aresta – tanto interna como externa – e aqueles correspondentes às arestas externas são desenhados no buffer estêncil, usando a técnica já descrita na seção 4.

O algoritmo também trata de casos especiais, como a perda da paridade do buffer estêncil, ocasionada quando o observador se encontra dentro de um volume de sombra ou ao interceptá-lo com seu volume de visão. Este problema é resolvido através do cálculo da interseção de todos os volumes de sombras com relação ao plano de cerceamento frontal do volume de visão. Os vértices resultantes formam polígonos de correção, os quais são adicionados ao buffer estêncil de acordo com o método descrito adiante na seção 7. Em suma, o algoritmo é composto dos seguintes passos:

Pré-processamento:

1. Construir a árvore BSP da cena.

Tempo de execução:

1. Percorrer a árvore, obtendo o grupo de polígonos visíveis

com relação à fonte de luz;
2. Identificar as arestas internas do grupo de polígonos obtidos;
3. Gerar polígonos de sombras para todas as arestas (tanto internas como externas). Aqueles correspondentes às arestas internas são marcados como invisíveis;
4. Desenhar no buffer estêncil os polígonos de sombra marcados como visíveis, conforme o procedimento explicado na seção 4;
5. Recortar cada volume de sombra com relação ao plano de cerceamento frontal do volume de visão, armazenando os vértices gerados;
6. Construir polígonos de correção usando os vértices gerados da etapa anterior, aplicando-os no buffer estêncil conforme o procedimento explicado na seção 7.

Se, por alguma razão, não houver situações em que o observador possa estar dentro ou parcialmente dentro de um volume de sombra (ex., aplicações com visão em terceira pessoa), o algoritmo pode ser ligeiramente simplificado. Neste caso, os passos 5 e 6 anteriores poderão ser removidos, e os polígonos de sombra necessitarão ser gerados somente a partir das arestas externas. De fato, a geração de polígonos de sombra para arestas internas é necessária somente para a composição de eventuais polígonos de correção de paridade do buffer estêncil.

5.1 Extensões

As extensões deste algoritmo híbrido correspondem às mesmas extensões possíveis para o algoritmo de renderização de sombras volumétricas.

Fontes de luzes múltiplas podem ser adicionadas facilmente através da repetição, para cada fonte de luz adicional, de todos os passos descritos anteriormente, exceto a reconstrução da árvore. As novas sombras geradas deverão ser combinadas com as sombras existentes através do uso de polígonos translucentes. O algoritmo também pode ser estendido para a simulação de sombras suaves (penumbra), usando um buffer de acumulação (disponível somente na API OpenGL). Este efeito pode ser feito renderizando as sombras múltiplas vezes no buffer de acumulação, movendo ligeiramente a posição da fonte de luz em cada vez.

6 Identificação de Arestas Adjacentes

Considerando as arestas de um grupo de polígonos convexos encarando a fonte de luz, as mesmas são ditas internas ou adjacentes se estão sendo compartilhadas por dois ou mais polígonos de orientação idêntica. Para o problema de geração de volumes de sombras, tais arestas podem ser descartadas, já que os polígonos de sombras gerados a partir das arestas externas já contém informações suficientes para a definição dos limites dos volumes de sombras. A redundância obtida pela geração de polígonos de sombra para os lados internos é seriamente custosa para o processo de renderização das sombras, visto que, para cada aresta interna, dois polígonos de sombra adicionais são desenhados no buffer estêncil.

Como solução, utilizamos um algoritmo de otimização que trabalha examinando cada lado de cada polígono, armazenando os índices dos vértices e identificando as redundâncias. O algoritmo exige, como entrada, um grupo de polígonos representados como índices de uma *array* de vértices. Este grupo refere-se aos polígonos da cena original que estão encarando a fonte de luz. Como saída, o algoritmo retorna um conjunto de listas contendo os lados identificados.

A principal desvantagem deste algoritmo consiste na sua ineficácia com relação ao tratamento de vértices formadores de junções em "T" originárias da construção da árvore BSP. Felizmente tal problema pode ser corrigido ainda durante a fase de pré-processamento.

6.1 Algoritmo

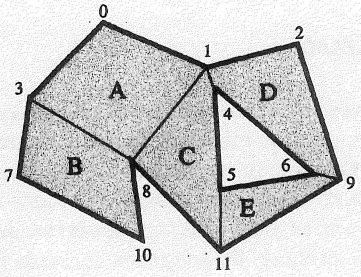
Primeiramente cria-se uma lista de n listas vazias, onde n é o número de vértices utilizados pelo grupo de polígonos. Apesar de parecer claro o uso de listas encadeadas, elas não são vantajosas. Ao contrário, na prática a eficiência decai devido ao elevado número de acessos à memória dinâmica. Listas estáticas, por outro lado, igualmente não são ideais devido à esparsidade das informações geradas, porquanto é desconhecido o número de posições necessárias em cada lista. Na prática isso não chega a constituir um problema, exceto no tratamento de instâncias contendo vários milhares de polígonos, onde o consumo de memória se eleva consideravelmente. Sugerimos, neste último caso, o uso de uma tabela de *hashing*.

```
Find_Adjacent_Edges(polygon set P, list set L)
{
  for each polygon p of P
    for each side n of p
    {
      i = first vertex of n
      j = second vertex of n
      if L[j] contains i
        delete i from L[j]
      else
        add j to L[i]
    }
  return L
}
```

Figura 3 – Pseudocódigo do algoritmo de identificação de arestas adjacentes.

Para cada lado de cada polígono, obtém-se o índice dos dois vértices do lado atual. Se a lista equivalente ao índice do segundo vértice contiver o índice do primeiro vértice, uma aresta adjacente foi encontrada. Neste caso, marca-se o elemento encontrado como "invisível". Caso contrário, o índice do segundo vértice é adicionado à lista de número equivalente ao índice do primeiro vértice. A figura 3 contém o pseudocódigo do algoritmo descrito.

No final do processo, cada lista conterá os índices que, juntamente com os índices correspondentes ao número de cada lista, formarão as arestas externas. Por exemplo, supondo que a lista de número 2 contenha os elementos 1, 3 e 5, as arestas externas ficam formadas pelos vértices (2,1), (2,3) e (2,5). A figura 4 ilustra um exemplo, mostrando o conteúdo das listas resultantes.



Polígonos de entrada:

A = 0,1,8,3 B = 3,8,10,7 C = 1,4,5,11,8
D = 1,2,9,6,4 E = 5,6,9,11

Saída:

Lista	Elementos		
0	1		
1	**8**	**4**	2
2	9		
3	0		
4	5		
5	**11**	6	
6	4		
7	3		
8	**3**	10	
9	**6**	11	
10	7		
11	8		

Figura 4 - Exemplo de identificação de arestas adjacentes. Os números hachurados da tabela correspondem aos elementos marcados como invisíveis, os quais são índices dos vértices que formam arestas adjacentes.

6.2 Eficiência

A eficiência é limitada superiormente em $O(n^2)$, tanto para a complexidade de tempo como de espaço. Na prática, a eficiência esperada é de $O(n)$.

7 Capping

Freqüentemente ocorre situações em que os polígonos de sombra estão sendo interceptados pelo volume de visão do observador. Em alguns casos, como por exemplo, quando o observador está dentro de um volume de sombra, o estágio de cerceamento (*clipping*) pode descartar fragmentos de polígonos de sombra que contenham importantes informações de paridade, gerando inversões de paridade no buffer estêncil e, desse modo, ocasionando o surgimento de sombras invertidas.

A solução típica [Crow (1977)] consiste em inverter de maneira global a paridade do buffer estêncil sempre que o observador se encontrar dentro de um volume de sombra. A figura 5 ilustra um exemplo onde esta solução pode ser adotada. Infelizmente, tal método não funciona nos casos em que o plano da frente do volume de visão está interceptando somente algumas partes dos volumes de sombras. Por exemplo, se o observador estiver olhando em direção à fonte de luz e alguns polígonos entre eles se encontrarem lançando sombras, a paridade poderá ser invertida somente localmente. Um exemplo é ilustrado na figura 6.

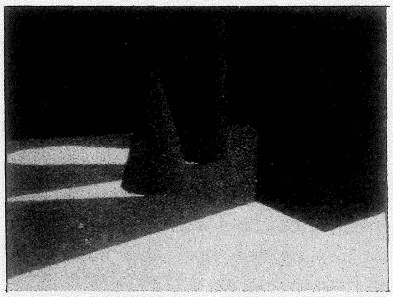


Figura 5 – Informação de paridade perdida globalmente.

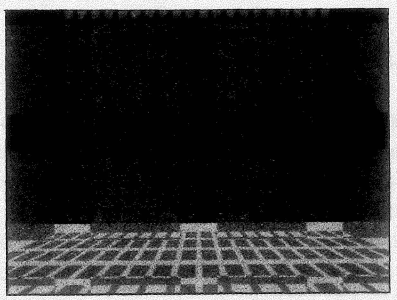


Figura 6 – Informação de paridade perdida localmente.

A solução adotada neste trabalho foi deduzida observando que, através da armazenagem dos vértices gerados do cerceamento de polígonos com o plano da frente do volume de visão, pode-se construir polígonos de correção, também conhecidos como *cap polygons* [McCool (1998)] (em português, "polígonos-tampa", pois funcionam como "tampas" dos volumes) usados para reverter a paridade do buffer estêncil para a situação desejada. A interseção entre cada polígono de sombra de cada volume de sombra (incluindo aqueles formados pelas arestas marcadas como invisíveis) e o plano da frente é calculada. Para cada volume de sombra, os novos vértices gerados são ordenados em ordem polar, criando então um *cap polygon* (figura 7). Esses mesmos polígonos são desenhados no buffer estêncil usando a operação "decrementar", corrigindo assim as inversões de paridade tanto locais como globais. A imagem resultante da correção da figura 6 é mostrada na figura 8.

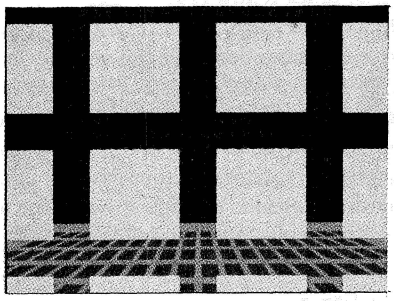


Figura 7 – *Cap polygons* (em branco).

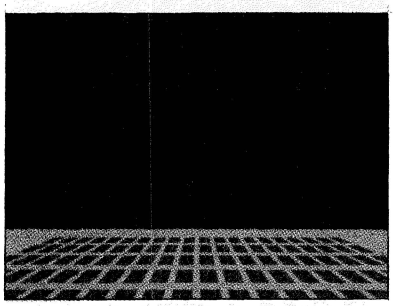


Figura 8 – Imagem corrigida pelo *capping*.

8 Implementação

O algoritmo proposto foi implementado em C++ usando o compilador MS Visual C++ 6.0. A API MS DirectX 6.0 foi utilizada na renderização das primitivas usando aceleração 3D sempre que disponível.

Os testes de desempenho foram realizados em dois computadores pessoais: um Pentium II de 400 Mhz, equipado com uma placa de aceleração 3D (chipset ATI Rage 128) e um Pentium II de 266 Mhz sem aceleração 3D. Neste último caso foi utilizado o modo de emulação de hardware do Directx.

Quatro cenas com diferentes números de polígonos foram empregadas para os testes de desempenho: uma esfera geodésica (figura 9), alguns sólidos simples (figura 10), um tetraedro recursivo (figura 11) e uma formação de cubos (figura 12). Todas as cenas foram renderizadas com uma resolução de 400x300 pixels, 32 bits de cores RGBA, sombreamento *Gouraud* e buffer estêncil de 8 bits. Um buffer estêncil de 1 bit foi utilizado no computador sem aceleração 3D.

Os testes foram computados sobre dez execuções de cada cena, cada execução com uma fonte de luz posicionada num lugar distinto.

O pré-processador da árvore BSP foi configurado de modo a obter o menor número possível de quebra de polígonos. Por outro lado, observamos que diferentes configurações do algoritmo não influenciaram a eficiência da geração de sombras de maneira significativa.

9 Resultados

A tabela 1 mostra o número de polígonos presentes em cada cena testada. As duas primeiras cenas representam os casos mais comuns em aplicações práticas, com um pequeno número de polígonos de sombra por quadro. Por outro lado, as duas últimas refletem as cenas mais complexas testadas (os "piores casos"), com centenas de polígonos de sombra gerados em cada quadro.

	Número de polígonos		
Cena	**nBSP**	**NShadow**	**Total**
Esfera geodésica	256	27	283
Sólidos	94	34	128
Tetraedro recursivo	752	596	1348
Cubos	1502	754	2256

Tabela 1 – Número de polígonos em cada cena, onde nBSP é o número de polígonos da árvore BSP e nShadow o número médio de polígonos de sombra gerados em cada quadro. Todas as cenas foram originalmente feitas de triângulos.

Comparando as duas primeiras cenas na tabela 1, pode-se observar que o número de polígonos de sombra não é proporcional ao número de polígonos da árvore BSP. Em geral, isso ocorre devido ao fato do número de polígonos de sombra ser mais dependente da posição da fonte de luz e da geometria da cena.

A tabela 2 apresenta os resultados dos testes de desempenho usando aceleração 3D em hardware. As cenas A, B, C e D correspondem às cenas descritas na tabela 1. Examinando os resultados da tabela 2, pode-se notar que a etapa de renderização de sombras é a mais custosa de todas,

enquanto que as etapas de geração dos volumes de sombras e *capping* despendem somente poucos milissegundos por quadro mesmo para as cenas mais complexas.

	Cena			
Tarefa	**A**	**B**	**C**	**D**
Geração de Sombra*	0.86	0.36	3.54	6.56
Capping*	0.18	0.08	0.64	1.42
Renderização*	0.92	1.12	37.48	52.08
Total	1.96	1.56	41.66	60.06
Hz (média)	510	641	24	17

* Tempo em milissegundos.

Tabela 2 – Desempenho de cada etapa usando aceleração 3D em hardware.

	Cena			
Tarefa	**A**	**B**	**C**	**D**
Ger. de Sombra*	1.26	0.54	4.98	9.28
Capping*	0.32	0.12	0.96	1.88
Renderização*	55.34	99.14	611.26	1001.5
Total	56.92	99.8	617.2	1012.66
Hz (média)	18	10	2	1

* Tempo em milissegundos.

Tabela 3 – Desempenho de cada etapa usando modo de emulação.

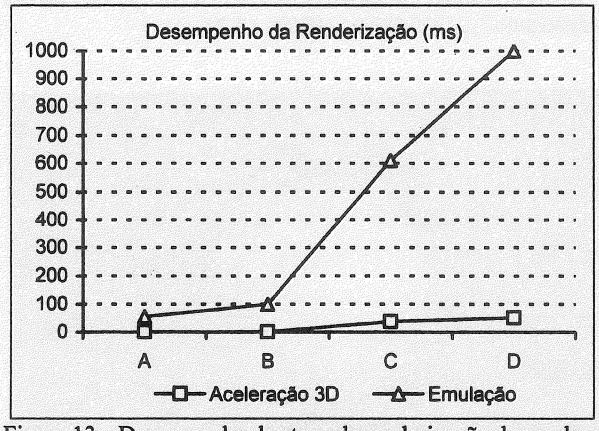


Figura 13 – Desempenho da etapa de renderização de sombras com e sem aceleração 3D.

A tabela 3 contém os valores de desempenho obtidos sem aceleração 3D. Os resultados são notavelmente contrastantes. Em comparação com a tabela 2, o tempo de renderização das sombras aumenta em até 60 vezes (figura 13), enquanto que as etapas de geração de sombras e *capping* continuam quase que com os mesmos desempenhos (figura 14). Já que essas duas últimas etapas são realizadas quase que inteiramente em software, os resultados sugerem que sua distinção é devida em grande parte à diferença entre o *clock* dos processadores. Por outro lado, a etapa de renderização das sombras se mostra fortemente dependente de aceleração de hardware, a ponto de tornar-se indispensável para um aproveitamento satisfatório do algoritmo.

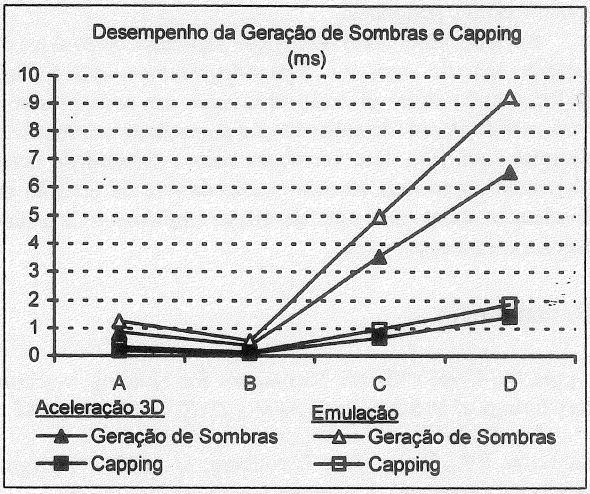


Figura 14 – Desempenho das etapas de geração de sombras e *capping*, com e sem aceleração 3D.

10 Conclusão

Apresentamos um algoritmo híbrido de geração de sombras em tempo real, baseado no método de renderização de volumes de sombras usando buffers estênceis e determinação de visibilidade de polígonos usando árvores BSP.

Os resultados dos testes de desempenho mostraram que a etapa de geração de volumes de sombras é suficientemente rápida à utilização em tempo real, mesmo considerando cenas complexas sem aceleração de hardware. Entretanto, a eficiência da etapa de renderização das sombras revelou-se extremamente dependente de placas de aceleração 3D com recursos de estencilação.

A eficiência do algoritmo pode ser melhorada através do uso de portais [Luebk — Georges (1995)] e algoritmos semelhantes em conjunto com a árvore BSP. Sabendo que a renderização das sombras é a etapa mais custosa de todo o processo, as otimizações devem estar focalizadas na redução do número de polígonos de sombra. Isto pode ser feito, por exemplo, através de um algoritmo de manipulação de nível de detalhes (*LOD management*) para cenas complexas. Outras melhorias podem ser feitas desenvolvendo técnicas de

aceleração do processo de identificação de arestas adjacentes e geração dos volumes de sombras.

Apesar do algoritmo estar limitado a ambientes estáticos e fontes de luzes pontuais, ele se mostra satisfatório para uso em aplicações interativas, como realidade virtual e jogos, devido à rápida popularização das placas aceleradoras 3D com suporte a buffers estênceis. Além disso, o algoritmo pode ser implementado facilmente em aplicações que já utilizem árvores BSP ou algoritmos semelhantes de determinação da ordem de visibilidade de polígonos.

Recentemente observamos que algumas companhias de programação de jogos (ex., Epic MegaGames, Id Software, Rebel Act Studios) já estão utilizando em seus produtos variantes da técnica de renderização de sombras volumétricas descrita aqui. Este fato corrobora com a capacidade deste algoritmo como uma técnica estado-da-arte de geração de sombras em tempo real, e que deverá ser amplamente difundida nos próximos anos.

Referências

Appel, A. (1968) "Some Techniques for Shading Machine Renderings of Solids", *Proc. AFIPS JSCC* 1968, 32, 37-45.

Atherton, P.R., Weler, K. e Greenberg, D. (1978) "Polygon Shadow Generation", *Computer Graphics*, 12, 275-281.

Bergeron, P., (1986) "A General Version of Crow's Shadow Volumes", *IEEE CG&A*, 6(9), 17-28.

Bouknight, W.J., Kelley, K. (1970) "An Algorithm for Producing Half-Tone Computer Graphics Presentations with Shadows and Movable Light Sources", *AFIPS Conf. Proc.*, 36, 1-10.

Chin, N., Feiner, S. (1989) "Near Real-Time Shadow Generation Using BSP Trees", *Computer Graphics* 23(3), 99-106.

Crow, F. (1977) "Shadow Algorithms for Computer Graphics", *Computer Graphics* 11(2) 242-247.

Foley, J., Van Dam, A., Hugues, J. e Feiner, S. (1990), "Computer Graphics: Principles and Practice", Addison Wesley, 2ª Edição.

Fuchs, H., Kedem, Z.M., e Naylor, B.F. (1980), "On Visible Surface Generation by A Priori Tree Structures", *Computer Graphics* 14(3), 124-133.

Goral, C., Torrance, K.E., Greenberg, D. (1984) "Modeling the Interaction of Light Between Diffuse Surfaces", *Computer Graphics*, 18(3), 213-222.

Heidmann, T. (1991), "Real Shadows, Real Time", *Iris Universe*, 18, 23-31, Silicon Graphics Inc.

Kilgard, M. (1997), "OpenGL-based Real-Time Shadows", Silicon Graphics Inc., (http://reality.sgi.com/mjk_asd/tips/rts/).

Luebke, D. e Georges, C. (1995), "Portals and mirrors: Simple, fast evaluation of potentially visible sets", *ACM Interactive 3D Graphics Conference*, Monterey, CA.

McCool, M.D. (1998), "Shadow Volume Reconstruction", Computer Graphics Laboratory, University of Waterloo. (http://www.cgl.uwaterloo.ca/~mccool/)

Schumacker, R., Brand, B., Gilliland, M., Sharp, W., (1969) "Study for Applying Computer-Generated Images to Visual Simulation", Technical Report AFHRL-TR-69-14, NTIS AD700375, U.S. Air Force Human Resources Lab., Air Force Systems Command, Brooks AFB, TX.

Slater, M. (1992) "A Comparison of Three Shadow Volume Algorithms", *The Visual Computer*, 1, 25-38.

Wade, B. (1996) "BSP Tree Frequently Asked Questions (FAQ)", Silicon Graphics Inc., (http://www.sgi.com/bspfaq).

Whitted, T. (1980) "An Improved Illumination Model for Shaded Display", *CACM*, 23(6), 343-349.

Willians, L. (1978) "Casting Curved Shadows on Curved Surfaces", *Computer Graphics*, 12, 270-274.

Woo, A., Poulin, P., Fourier, A. (1990) "A Survey of Shadow Algorithms", *IEEE CG&A*, 10(6), 13-31.

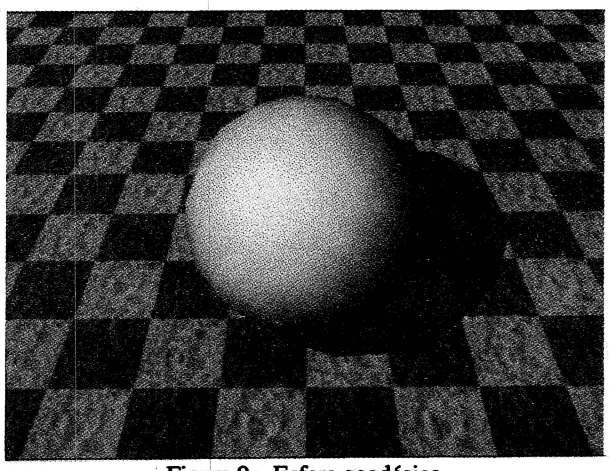


Figura 9 – Esfera geodésica.

Figura 10 – Sólidos.

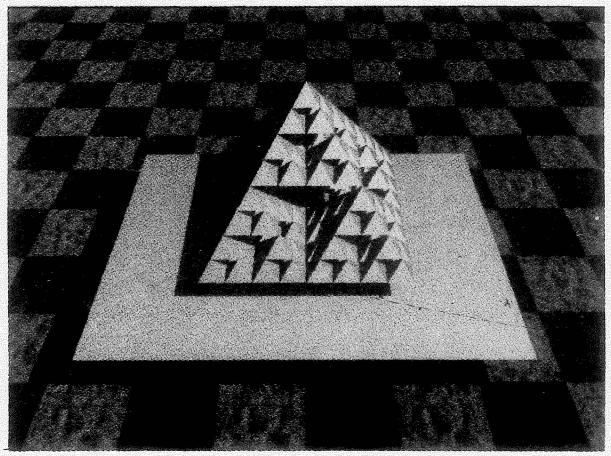


Figura 11 – Tetraedro recursivo

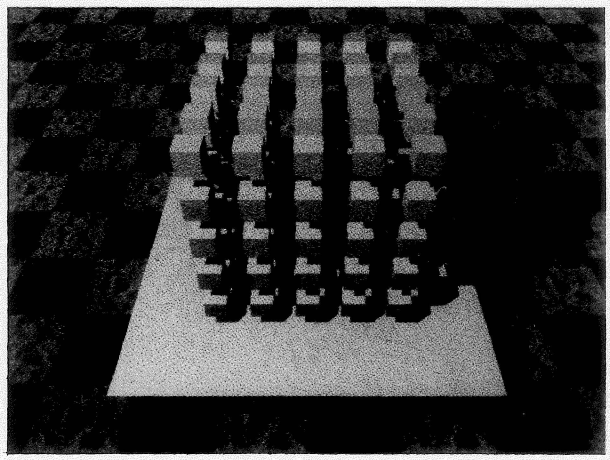


Figura 12 - Cubos