MC409/MO603 – Computação Gráfica

© Jorge Stolfi

Segundo Semestre de 1994

Notas de Aula – Fascículo 8
Algoritmos de visibilidade

8.1 O problema

Após a transformação de perspectiva, e a poda contra o volume da imagem, a etapa seguinte é a eliminação das partes da cena que são ocultas por elementos opacos da própria cena.

Os algoritmos para esta etapa dividem-se em dois tipos principais. O primeiro tipo produz como resultado um *diagrama*: uma conjunto finito finito de *elementos* — pontos, linhas, e regiões — contidos na imagem I, disjuntos dois a dois, tais que cada elemento do diagrama é a projeção de um pedaço visível de um único elemento da cena. Este diagrama pode ser posteriormente transformado numa imagem — uma matriz de píxels — por um *algoritmo de rasterização* separado.

Em contraste, os algoritmos do segundo tipo produzem diretamente uma matriz de píxels, a partir da cena. Deixaremos tais algoritmos para o fascículo seguinte.

Os algoritmos que veremos a seguir pressupõem que a cena é apenas uma coleção de polígonos abertos no espaço $\mathbb{T}_3$. (Se necessário, esses algoritmos podem ser facilmente ampliados para tratar também segmentos e pontos. Entretanto, os melhores algoritmos de rasterização geralmente ignoram segmentos e pontos isolados do diagrama, pois eles não afetam a imagem final.)

Vamos supor também que a cena já foi transformada para o sistema SI, com o observador em $[0, 0, 0, 1]$, e que as partes fora do volume da imagem foram podadas.

8.1.1 Visibilidade entre dois elementos

O cerne de muitos dos algoritmos a seguir é uma *rotina básica de visibilidade*, que, dados dois elementos da cena, E (o elemento a desenhar) e B (um obstáculo potencial), retorna as partes de E que não são ocultas por B.

Observe-se que estas são justamente as partes de E que estão contidas no *volume de sombra* de B. Este volume consiste de todos os pontos do espaço que estão atrás de B, em relação ao observador. Portanto, a rotina básica de visibilade precisa determinar a intersecção do elemento E com o complemento do volume de sombra de B.

Se B é um polígono (ou,em geral, uma figura plana), seu volume de sombra é um prisma semi-infinito, com lados paralelos ao eixo Z, limitado por B na direção de Z positivo, e ilimitado na direção de Z negativo.

Como já observamos no fascículo anterior, dependendo da natureza dos elementos da cena, a rotina básica de visibilidade pode se ver obrigada a subdividir a parte visível do elemento a desenhar E em dois ou mais elementos.

8.1.2 Visibilidade básica para dois triângulos

Os detalhes da rotina básica de visibilidade dependem naturalmente da natureza dos objetos que compõem a cena. Para exemplificar, consideraremos apenas o caso em que os objetos E e B são triângulos; objetos mais complicados, como polígonos, podem ser recortados em (ou aproximados por) coleções de triângulos.

Quando B é um triângulo, seu volume de sombra é um prisma triangular semi-infinito, com lados paralelos ao eixo Z, limitado pelo triângulo B na direção de Z positivo, e ilimitado na direção de Z negativo. Veja a figura 8.1.

Este prisma é a intersecção de quatro semi-espaços, os lados positivos de quatro planos α, β, γ, e δ. Os três primeiros α, β, γ definem os lados do prisma. Cada um destes planos é determinado por dois vértices do triângulo e pelo ponto do observador $o = [0, 0, 0, 1]$, e orientado de tal forma que o terceiro vértice do triângulo está no seu lado positivo. O quarto plano δ é o próprio plano do triângulo, orientado de tal forma que o ponto $[0, 0, 0, -1]$ está no lado positivo.

Portanto, a eliminação das partes de E que são ocultas pelo triângulo B consiste em recortar E sucessivamente pelos quatro planos α, β, γ e δ. Assim como na seção anterior, este processo é convenientemente descrito como uma *pipeline* de quatro estágios, onde cada estágio é um filtro que recorta os triângulos de entrada contra um dos planos α, β, γ, or δ do triângulo B.

Entretanto, no caso presente cada filtro precisa ter duas saídas, uma para as partes que estão no lado positivo de π, outra para as que estão no lado negativo. A saída *negativa* de cada estágio é a entrada do estágio seguinte, enquanto que a união de todas as saídas positivas é a saída final do *pipeline*. Veja a figura 8.2.

8.1.3 Otimização da rotina básica de visibilidade

Observe-se que o processamento de um único triângulo E pela *pipeline* acima pode resultar em zero triângulos, ou mais de um triângulo, na saída da mesma. Na verdade, como já vimos no fascículo anterior, esta implementação baseada em *pipeline* tende a recortar E em muito mais pedaços do que necessário necessário, e portanto a gastar mais ciclos do que necessário — mesmo nos casos em que E é completamente visível ou completamente invisível.

Para remediar este problema, podemos usar as mesmas técnicas que usamos para os algoritmos de poda do fascículo anterior. Por exemplo, uma solução é trabalhar com polígonos arbitrários, em vez de triângulos.

Outra solução parcial é utilizar a técnica de Cohen-Sutherland para detectar antecipadamente os casos em que E é totalmente visível, ou totalmente invisível, antes de recortá-lo. Para tanto, antes de processar um par E, B pela *pipeline* acima, devemos verificar a posição dos três vértices de E em relação a cada um dos quatro planos α, β, γ, e δ que delimitam o volume de sombra de B. Se, para algum plano π dentre estes quatro, os vértices originais de E estiverem todos em π ou no lado negativo de π, concluímos que o triângulo E não é ocultado por B, e pode ser devolvido imediatamente pela rotina, sem passar pela *pipeline*. Simetricamente, se em todos os estágios o triângulo E

estiver no plano correspondente ou no lado positivo do mesmo, então E é totalmente invisível, e pode ser descartado sem mais. Nos demais casos, E é processado pela *pipeline*, como descrito acima.

Na verdade, podemos reforçar esta otimização, observando que o triângulo B só pode ocultar alguma parte do triângulo E se B penetrar o *volume de anti-sombra* de E, que consiste de todos os pontos do espaço $\mathbb{T}_3$ que estão entre E e o observador o. Este volume é simplesmente o prisma triangular semi-infinito com base E, paralelo ao eixo Z, que se estende indefinidamente na direção de Z positivo. Este prisma é delimitado por quatro planos α', β', γ', e δ', definidos da mesma maneira que para o volume de sombra de B; exceto que o plano δ' está orientado no sentido oposto, com o observador $o = [0, 0, 0, 1]$ no lado positivo.

Portanto, antes de aplicar a *pipeline*, podemos também testar os vértices de B contra estes quatro planos. Se, para algum π' dentre eles, todos os vértices de B estiverem em π' ou no lado negativo de π', então B está inteiramente fora do volume de anti-sombra de E; isto quer dizer que E não é ocultado por B, e pode ser retornado imediatamente.

Na implementação desta rotina básica, devemos tomar cuidado com o caso particular em que os triângulos E e B são coplanares e não disjuntos. Não podemos simplesmente convencionar que neste caso E está sempre na frente de B, ou sempre atrás; pois isso nos levaria a dizer que ambos os triângulos são visíveis, ou que ambos são invisíveis porque cada um oculta o outro. (Ou, pior ainda, que todo triângulo da cena é invisível, porque oculta a si próprio!) Uma solução para este dilema é atribuir a cada triângulo de entrada t um "número de série" único $\#t$, que é automaticamente compartilhado por todos os pedaços de t produzidos no decorrer do algoritmo. Este número pode então ser usado para quebrar "empates": se E e B são coplanares e suas projeções se interceptam, então B oculta E se e somente se $\#b < \#e$.

8.1.4 Comparação exaustiva: o algoritmo de Roberts

Dentre os algoritmos e visibilidade que produzem um diagrama, o mais trivial — conhecido como *algoritmo de Roberts* — consiste basicamente em comparar cada elemento da cena contra todos os outros, eliminando partes daquele que resultam ocultas por estes.

Como vimos, a rotina básica de visibilidade pode recortar um único elemento em vários pedaços. Isto complica um pouco a codificação das duas malhas do algoritmo de Roberts (sobre os elementos a desenhar E, e sobre os obstáculos B).

O procedimento recursivo $R(\mathcal{E}, \mathcal{B})$ abaixo é uma maneira de resolver estas dificuldades. Ele recebe um conjunto $\mathcal{E}$ de elementos da cena a desenhar, e um conjunto $\mathcal{B}$ de elementos opacos da cena que podem ocultar os primeiros. (Na chamada inicial, $\mathcal{E} = \mathcal{B} =$ todos os elementos da cena.) O procedimento R devolve o conjunto dos pedaços de elementos de $\mathcal{E}$ que não estão escondidos por nenhum dos elementos $\mathcal{B}$.

1. Se $\mathcal{E} = \{\}$, devolve $\{\}$; se $\mathcal{B} = \{\}$, devolve $\mathcal{E}$.

2. Seja E um elemento qualquer de $\mathcal{E}$, e B um elemento qualquer de $\mathcal{B}$. Usando a rotina básica de visibilidade, determine o conjunto $\mathcal{V}_0$ de partes de E que não são escondidas pelo triângulo B.

3. Calcule e devolva $R(\mathcal{E} \setminus \{E\}, \mathcal{B}) \cup R(\mathcal{V}_0, \mathcal{B} \setminus \{B\})$.

Este algoritmo é passível de algumas otimizações óbvias. Por exemplo, sempre que a rotina básica de visibilidade informa que E é totalmente oculto por B, podemos retirar E não só da lista de objetos a desenhar, mas também da lista de possíveis obstáculos. Isto porque qualquer elemento que E possa ocultar também será oculto por B.

Mesmo com estas melhoria, o custo do algoritmo de Roberts é obviamente proporcional ao quadrado do número de elementos da cena — ou pior, na medida que a rotina básica de visibilidade vai recortando os elementos originais em mais e mais pedaços. Mais adiante veremos técnicas gerais que permitem melhorar bastante o desempenho deste algoritmo, bem como algoritmos mais eficientes.

8.1.5 Subdivisão retangular: o algoritmo de Warnock

Outro algoritmo clássico de visibilidade (hoje em dia, suspeito, mais popular em livros do que na prática) é o *algoritmo de Warnock*, baseado numa subdivisão recursiva da imagem em sub-imagens retangulares.

O algoritmo de Warnock divide repetidamente ao meio o retângulo I da imagem por linhas alternadamente paralelas aos eixos X e Y, até que a cena visível dentro de cada uma dessas sub-imagens seja suficientemente simples. Cada sub-imagem é então calculada pelo algoritmo trivial (Roberts).

No decorrer desse processo, sempre que o algoritmo encontra um polígono cuja projeção no plano $Z = 0$ cobre toda a imagem I, ele elimina todas as partes da cena que estão atrás desse polígono. Esta precaução garante que quase todas as ramificações recursivas do algoritmo param após um número finito de chamadas.

Mesmo com este cuidado, a recursão descrita acima continuaria indefinidamente em torno de pontos da imagem cuja vizinhança é excessivamente complexa, isto é, pontos que são incidentes a projeções de muitos elementos visíveis. Para resolver este problema, o algoritmo encerra forçadamente a recursão sempre que o tamanho a sub-imagem é menor que um tamanho mínimo ε (tipicamente, uma fração de píxel), e retorna um elemento aleatório da cena dentro dessa sub-imagem. Esta solução pressupõe que tais "pontos complicados" da imagem são poucos e isolados, e que portanto a contribuição dessas sub-imagens "truncadas" para a imagem total é desprezível.

Uma descrição mais detalhada do algoritmo é o procedimento recursivo $W(I, \mathcal{C})$ abaixo. O parâmetro I é um retângulo no plano $Z = 0$ que define a (sub-)imagem a construir. O parâmetro $\mathcal{C}$ é uma lista de triângulos da cena, já transformados pela matriz de perspectiva e podados contra o paralelepípedo visível correspondente à janela I. As constantes K e ε, que definem as condições de parada, dependem da da aplicação.

1. Se $\mathcal{C}$ contém um polígono P cuja projeção no plano $Z = 0$ cobre completamente o retângulo I, então recorte todos os outros polígonos de $\mathcal{C}$ pelo plano π de P, e descarte os pedaços que estão atrás do plano π (i.e., no lado de π oposto ao observador.)

2. Se $|\mathcal{C}| \leq K$, calcule e devolva as partes visíveis de $\mathcal{C}$, usando o algoritmo de Roberts.

3. Caso contrário, se os lados do retângulo I forem todos menores que ε, devolva primeiro polígono da lista $\mathcal{C}$.

4. Caso contrário, divida o retângulo I ao meio por uma linha m horizontal ou vertical, perpendicular à sua maior dimensão. Sejam J e K as metades resultantes de I.

5. Seja α o plano paralelo ao eixo Z que contém a linha m, e orientado de modo que J está no lado positivo. Recorte todos os polígonos da lista $\mathcal{C}$ pelo plano α. Sejam $\mathcal{A}$ e $\mathcal{B}$ as listas dos pedaços no lado positivo e negativo de α, respectivamente.

6. Calcule e devolva $W(J, \mathcal{A}) \cup W(K, \mathcal{B})$.

Este algoritmo básico pode ser otimizado de várias maneiras. Por exemplo, é provavelmente uma boa idéia manter a lista $\mathcal{C}$ em ordem aproximada de Z decrescente. Isso tende a tornar o passo 1 mais eficiente. Uma maneira de definir essa ordem é calcular a coordenada Z do plano de cada polígono no centro do retângulo I (ou, o que dá na mesma, nos quatro cantos de I, e tirar a média).

8.1.6 Subdivisão poligonal: o algoritmo de Weiler e Atherton.

O algoritmo de Weiler e Atherton é baseado, como o de Warnock, na divisão recursiva da imagem e da cena. A diferença é que, ao invés de cortes simples verticais ou horizontais, ele usa cortes poligonais, baseados nos contornos projetados dos elementos da própria cena.

O algoritmo pode ser descrito pelo procedimento recursivo $WA(I, \mathcal{C})$ abaixo. O parâmetro I é um polígono no plano $Z = 0$ (possivelmente com buracos e/ou várias componentes conexas), que especifica a parte da imagem a ser calculada. Subentende-se que o volume da imagem é um prisma I^* paralelo ao eixo Z, cuja secção é o polígono I, limitado pelos planos de frente ($Z = Z_{\max}$) e fundo ($Z = Z_{\min}$). O parâmetro $\mathcal{C}$ é a cena a desenhar, na forma de um conjunto de polígonos coloridos no espaço, já devidamente transformados pela matriz de perspectiva, e podados contra o volume da imagem.

1. Se $|\mathcal{C}| \leq 1$, devolva $\mathcal{C}$ e pare.

2. Caso contrário, seja P um polígono qualquer de $\mathcal{C}$. Seja J a projeção de P no plano $Z = 0$, e K o polígono $I \setminus J$.

3. Recorte todos os demais polígonos de $\mathcal{C}$ contra o prisma infinito J^*, paralelo ao eixo Z, cuja secção é o polígono J. Seja $\mathcal{A}$ o conjunto dos pedaços desses polígonos que estão dentro de J^* (incluindo P), e $\mathcal{B}$ os pedaços que estão em $I^* \setminus J^*$.

4. Recorte todos os polígonos de $\mathcal{A}$ contra o plano π do polígono P, e descarte todos os pedaços que estiverem atrás de π (i.e. no lado de π que não contém o observador).

5. Devolva $WA(J, \mathcal{A}) \cup WA(K, \mathcal{B})$.

Ao contrário do algoritmo de Warnock, esta recursão sempre termina após um número finito de passos, pois cada sub-divisão da imagem diminui o número de arestas de polígonos de $\mathcal{C}$ cujas projeções não estão na fronteira da imagem I.

A principal dificuldade do algoritmo de Weiler e Atherton são os passos 3 e 4, que juntos constituem a rotina básica de visibilidade para dois polígonos arbitrários no espaço. O passo 4 corta um polígono de $\mathbb{T}_3$ por um plano, e está longe de ser trivial (vide fascículo anterior). O passo 3 equivale (mediante projeção no plano $Z = 0$) a calcular a intersecção de dois polígonos arbitrários de $\mathbb{T}_2$; o que é um problema ainda mais complicado que o anterior.

Assim como no algoritmo de Warnock, é desejável manter os polígonos de cada sub-cena aproximadamente ordenados por Z decrescente; e, no passo 2, tomar para P o primeiro polígono da lista, nesta ordem.

8.1.7 Weiler e Atherton para polígonos convexos

Uma maneira de facilitar a implementação do algoritmo de Weiler e Atherton é trabalhar com polígonos convexos (ou, em particular, triângulos), em lugar de polígonos arbitrários. Com esta mudança, no passo 4 precisaríamos apenas calcular a intersecção de dois polígonos convexos; que por sua vez pode ser reduzida a repetidas intersecções entre um polígono convexo e um semiplano.

O preço desta simplificação é que o algoritmo fica mais demorado, pois a imagem e os elementos da cena precisam ser recortados num número maior de pedaços.

Segue o procedimento $WA(I, C)$ acima, modificado segundo esta idéia. Note que, agora, tanto I quanto os elementos de $\mathcal{C}$ são polígonos convexos.

1. Se $|\mathcal{C}| \leq 1$, devolva $\mathcal{C}$ e pare.
2. Caso contrário, se existir em $\mathcal{C}$ algum polígono P cuja projeção no plano $Z = 0$ cobre inteiramente a imagem I, então

   2.1. Recorte todos os polígonos de $\mathcal{C}$ contra plano π do polígono P, descartando todas as partes que estão atrás de π;

   2.2. Se $|\mathcal{C}| = 1$, devolva $\mathcal{C}$ e pare.
3. Seja P um polígono qualquer de $\mathcal{C}$, e e uma aresta de P cuja projeção no plano $Z = 0$ não está na fronteira da imagem I. Seja α o plano paralelo ao eixo Z que contém a aresta e. Recorte a janela da imagem I com o plano α; sejam J e K as metades positiva e negativa da mesma, respectivamente.
4. Recorte todos os polígonos da cena $\mathcal{C}$ com o plano α; sejam $\mathcal{A}$ e $\mathcal{B}$ os conjuntos de partes nos lados positivo e negativo, respectivamente.
5. Calcule e devolva $WA(J, \mathcal{A}) \cup WA(K, \mathcal{B})$.

O mesmo procedimento funciona para triângulos, com a ressalva que os passos 2.1 e 4 podem quebrar cada triângulo em dois ou três triângulos, respectivamente; enquanto que, na versão com polígonos convexos, esses passos cortam cada polígono em no máximo dois pedaços, um em cada lado do plano de corte.

8.1.8 Eliminação de faces posteriores

Além das otimizações específicas para cada algoritmo, há certas técnicas que podem ser opcionalmente usadas para melhorar o desempenho de qualquer um deles.

Um exemplo é a eliminação prévia de faces na parte de trás de objetos sólidos. Tais faces são completamente invisíveis, pois estão ocultas pelo interior do objeto (e pelas faces da frente do mesmo).

Para implementar esta otimização, basta tomar cuidado, ao definir um objeto da cena, de orientar o plano des uporte π de cada face P de maneira consistente em relação ao objeto; digamos, de tal forma que o interior do objeto (na vizinhança da face em questão) fica no lado negativo do plano. Com essa convenção, uma face do objeto estará voltada para observador se e somente se o ponto o estiver no lado positivo do respectivo plano. As faces que não passam por este teste podem ser imediatamente descartadas.

A solução acima pressupõe que as faces dos objetos são polígonos gerais, representados como descrito no fascículo anterior (com o plano de suporte π definido implicitamente pelo mapa μ_P). No caso em que as faces são triângulos, cada um representados pelos seus três vértices, a solução análoga é sempre enumerar os vértices em ordem anti-horária quando vistos do lado de fora da face. Nesse caso, depois da transformação de perspectiva, uma face está voltada para o observador se e só se as projeções de seus três vértices, no plano $Z = 0$, definirem um triângulo positivo.

8.1.9 Caixas envoltórias

Outra técnica genérica que pode ser usada para acelerar todos os algoritmos vistos acima é o uso de *caixas envoltórias.*

A idéia é organizar os elementos em grupos, preferivelmente compactos, e possivelmente hierárquicos, e associar a cada grupo $\mathcal{G}$ um sólido geométrico simples $\mathrm{bbox}(\mathcal{G})$ — sua caixa envoltória — que engloba todos os elementos do grupo $\mathcal{G}$ no seu interior. Obviamente, um objeto A num grupo $\mathcal{A}$ só pode ocultar um objeto B de outro grupo $\mathcal{B}$ se as projeções das caixas $\mathrm{bbox}(\mathcal{A})$ e $\mathrm{bbox}(\mathcal{B})$ no plano da imagem tem algum ponto em comum. (É claro que a recíproca nem sempre vale.) Portanto, antes de testar todos os objetos de $\mathcal{A}$ contra todos os de $\mathcal{B}$, é aconselhável testar as respectivas caixas envoltórias; se as projeções delas forem disjuntas, todos esses testes de visibilidade podem ser omitidos.

O tipo de sólido geométrico usado para as caixas envoltórias deve ser simples, para reduzir o custo do teste acima e da construção das caixas. Ao mesmo tempo, ele deve deve ter um número suficiente de graus de liberdade para que as caixas se ajustem bem aos objetos que contém. Uma família de objetos que preenche esses requisitos são as caixas retangulares de tamanho e orientação arbitrária, e suas imagens por transformações projetivas. Outra família nessas condições é a dos objetos quádricos, definidos por superfícies de segundo grau — elipsóides, parabolóides, etc. (Entretanto, algoritmos para construir caixas envoltórias deste tipo parecem ser demasiado complexos e demorados para uso prático.)

8.1.10 Arvore de partição binária do espaço

Ainda outra técnica para acelerar algoritmos de visibilidade é a estrutura de dados conhecida como *árvore de partição binária do espaço* (em inglês, *BSP*).

Esta estrutura consiste numa árvore binária, cada nó v da qual está associado a um plano orientado $\pi(v)$, e a uma lista de polígonos $\mathcal{L}(v)$ da cena situados no plano $\pi(v)$. As duas sub-árvores de cada nó são rotuladas *positiva* e *negativa*, e denotadas por $\operatorname{tpos}(v)$ e $\operatorname{tneg}(v)$. Por construção, se um nó u pertence à sub-árvore positiva do nó v, então o interior de todo polígono da lista $\mathcal{L}(u)$ está totalmente contido no lado positivo do plano $\pi(v)$. Para a sub-árvore negativa de v, vale a propriedade simétrica.

Uma árvore BSP define portanto uma partição hierárquica do espaço $\mathbb{T}_3$. A cada nó v da árvore corresponde uma região convexa $R(v)$ de $\mathbb{T}_3$, que contém todos os polígonos associados aos nós descendentes de v. Esta região $R(v)$ é dividida em duas sub-regiões pelo plano $\pi(v)$, que são as regiões associadas aos nós filhos de v. A raiz da árvore está associada ao espaço $\mathbb{T}_3$ inteiro.

A utilidade da árvore BSP em algoritmos de visibilidade decorre da seguinte propriedade: para todo nó v, se o observador está no lado positivo do plano $\pi(v)$, então nenhum polígono da sub-árvore $\operatorname{tneg}(v)$ pode esconder algum polígono da lista $\mathcal{L}(v)$; e nenhum destes polígonos pode esconder algum polígono da sub-árvore $\operatorname{tpos}(v)$. Isto porque $\operatorname{tneg}(v)$ está inteiramente atrás do plano $\pi(v)$, enquanto que $\operatorname{tpos}(v)$ está na frente dele, visto do observador. Obviamente, se o observador está no lado negativo de $\pi(v)$, os papéis de $\operatorname{tpos}(v)$ e $\operatorname{tneg}(v)$ se invertem.

Estendendo esta propriedade recursivamente para todos os nós, concluímos que, para uma dada posição do observador, a árvore BSP define uma enumeração linear $P_1, .. P_n$ dos polígonos da cena, tal que um polígono P_i pode esconder um polígono P_j só se $i < j$. Esta enumeração é obtida percorrendo-se a árvore em inordem (isto é, ordem simétrica), sendo que para cada nó v percorremos $\mathrm{tpos}(v)$ antes de $\mathrm{tneg}(v)$ se o observador o está no lado positivo de $\pi(v)$, e $\mathrm{tneg}(v)$ antes de $\mathrm{tpos}(v)$ caso contrário.

No algoritmo de Roberts, por exemplo, esta propriedade nos permite reduzir o número de testes básicos de visibilidade: em vez de testar cada polígono P_j contra todos os outros polígonos P_i, basta testá-lo contra os polígonos P_i com $i < j$.

Nos algoritmos de Warnock e Weiler-Atherton, em vez de ordenar os polígonos pelas suas coordenadas Z num ponto qualquer da janela, podemos usar a ordem definida pela enumeração acima. Em primeiro lugar, isto nos poupa o trabalho de re-ordenar os polígonos cada vez que a janela é dividida. Mais ainda, quando encontramos um polígono P cuja projeção cobre a janela inteira, a eliminação das partes da cena ocultas por P fica trivial: basta eliminar todos os polígonos que seguem P na enumeração, e conservar todos os demais.

Se qualquer destes três algoritmos for modificado para aproveitar a árvore BSP, como descrito acima, o teste básico de visibilidade entre dois polígonos (ou triângulos) somente será usado quando obstáculo potencial B está na frente de algum plano $\pi(v)$, que está na frente do polígono a desenhar E. Nesse caso, podemos concluir de antemão que (1) os dois polígonos não se cruzam no espaço, e (2) se a projeção de B intersecta a projeção de E, então B esconde a parte de E correspondente a essa intersecção. Portanto, podemos simplificar o teste básico: não é necessário recortar E pelo plano de B, basta recortá-lo pelo prisma duplamente infinito cuja secção é B. No caso de triângulos, isso significa que podemos eliminar o estágio δ da *pipeline* de visibilidade.

Note que a árvore BSP em si depende apenas da cena; ela não depende da posição do observador, ou de quaisquer outros parâmetros da perspectiva. Assim, a mesma árvore BSP pode ser usada para ordenar os polígonos da cena em relação a vários pontos de vista diferentes.

Uma árvore BSP para uma cena $\mathcal{C}$ pode ser construída pelo seguinte algoritmo recursivo:

1. Se $\mathcal{C}$ contém apenas um polígono P, então

    1.1. Devolva um nó v com $\mathcal{L}(v) = \{P\}$, $\pi(v) =$ plano de P, $\text{tpos}(v) = \text{tneg}(v) = \Lambda$.

2. Caso contrário,

    2.1. Escolha um plano π que divida a cena $\mathcal{C}$ em duas partes aproximadamente iguais.

    2.2. Recorte todos os polígonos da cena contra o plano π. Sejam $\mathcal{A}$, $\mathcal{L}$, e $\mathcal{B}$ os conjuntos dos polígonos e pedaços de polígonos de $\mathcal{C}$ que estão no lado positivo de π, no plano π, e no lado negativo de π, respectivamente.

    2.3. Construa recursivamente as árvores BSP T_A para o conjunto $\mathcal{A}$ e T_B para o conjunto $\mathcal{B}$.

    2.4. Devolva um novo nó v com $\pi(v) = \pi$, $\mathcal{L}(v) = \mathcal{L}$, $\text{tpos}(v) = T_A$, e $\text{tneg}(v) = T_B$.

A parte mais difícil deste algoritmo é escolher o plano π no passo 2.1. Uma solução simples é usar o plano suporte de um polígono da cena, escolhido aleatoriamente. Para a maioria das cenas que ocorrem na prática, esta escolha quase sempre divide a cena em duas metades com aproximadamente a mesma complexidade.

Entretanto, para certas cenas patológicas — por exemplo, uma cena que consiste num único poliedro convexo — este método sempre resultará numa árvore extremamente desbalanceada. Uma maneira de resolver este problema é escolher três *vértices* aleatóriamente, dentre todos os vértices da cena, e tomar o plano π determinado por esses três pontos. Outra maneira é determinar uma direção d (genérica, ou paralela a um dos eixos) na qual a cena tem maior extensão, ordenar os vértices segundo essa coordenada, e tomar um plano π perpendicular a d que divide essa lista ao meio.

Note que o corte de polígonos que cruzam π, no passo 2.2, aumenta a complexidade total da cena. No pior caso, uma cena com n polígonos pode resultar numa árvore BSP com $\Omega(n^3)$ nós (talvez um pouco menos, se os polígonos originais não se cruzarem, e se os planos π forem escolhidos cuidadosamente.) Felizmente, tais cenas não costumam aparecer na prática; em geral a árvore BSP para n polígonos tem $O(n)$ nós.

O ideal seria encontrar um plano π que separe a cena em duas metades aproximadamente iguais, *sem cortar muitos polígonos*. Mas este é um problema difícil, que é ainda assunto de pesquisa.

Figura 8.1: Volume de sombra de um triângulo

Figura 8.2: O *pipeline* de visibilidade para um obstáculo triangular