4
Implementação de um Ambiente Acústico Virtual

Neste capítulo, descrevemos em detalhe a implementação de um programa que modela a propagação do som em ambientes bidimensionais com feixes de reflexão especular e de difração. No final do capítulo, apresentamos como este sistema realiza a reprodução de áudio, utilizando uma biblioteca capaz de posicionar uma fonte sonora no espaço.

Nosso programa foi implementado em C++, utilizando as bibliotecas IUP [3], CD [4], Lua [5] e CGAL [6]. Da última foram utilizados, principalmente, os algoritmos de triangulação de Delaunay com restrições e a estrutura topológica implementada para mapas planares.

4.1
Construção do Ambiente

Os ambientes bidimensionais que trataremos em nossa implementação são definidos por uma estrutura de grafo em que os vértices representam posições no plano e os arcos representam as paredes (ou oclusores) do ambiente. Adota-se a restrição de que as interseções de paredes podem ocorrer apenas em vértices do grafo, como ilustrado na Figura 4.1. Esta estrutura é bastante conveniente, pois o conceito de conexão nos permite criar ambientes sem que existam falhas nas junções das paredes, por onde feixes de propagação poderiam ser criados indevidamente.

Apesar dos exemplos apresentados neste capítulo e nos capítulos seguintes apresentarem ambientes com paredes paralelas aos eixos cartesianos, não existe nenhuma restrição no algoritmo ou no programa implementado que determine a inclinação das paredes utilizadas.

Como mencionamos anteriormente, existe uma grande semelhança entre os algoritmos utilizados para calcular a propagação de ondas sonoras (ou eletromagnéticas) e algoritmos de visualização (remoção de superfície escondida). Todos esses algoritmos resolvem um mesmo problema básico: determinação de visibilidade. Ao traçarmos feixes de propagação de som,

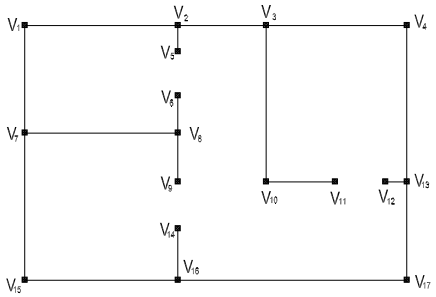


Figura 4.1: Definição de um Ambiente

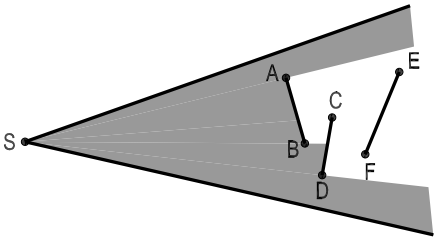


Figura 4.2: Como a Ordem dos Oclusores Afeta a Visibilidade

precisamos determinar sobre qual dos oclusores incidem ondas sonoras, para que as mesmas possam ser refletidas e difratadas corretamente. Determinar essa incidência é determinar se um oclusor é visível ou não em relação à fonte sonora.

A visibilidade de um oclusor será determinada por sua posição em relação aos demais oclusores e em relação à fonte sonora (ou à câmera, no caso de algoritmos de visualização). Essa posição relativa pode ser descoberta realizando uma ordenação dos oclusores em relação à fonte, como feito em [14]. Uma vez ordenados, a determinação de visibilidade de um oclusor O pode ser feita determinando se existe um raio proveniente da fonte de forma que O seja o primeiro oclusor, na ordenação, a interceptar este raio. Esta abordagem, entretanto, apresenta uma séria desvantagem: sempre que houver uma alteração na posição da fonte sonora (ou da câmera), será preciso reordenar os oclusores.

A Figura 4.2 ilustra um feixe irradiando sobre três oclusores. Note que, apesar dos três oclusores estarem localizados dentro da região delimitada pelo feixe, os oclusores mais próximos à fonte S, AB e CD, bloqueiam a visão do terceiro oclusor, EF. Apenas sabendo como esses oclusores são posicionados em relação à fonte, podemos delimitar corretamente as fronteiras do feixe de propagação.

Uma forma bastante comum de acelerar as determinações de visibilidade é a construção de uma subdivisão do espaço composta por regiões

convexas (chamadas *células*) [14, 20], que pode ser realizada introduzindo paredes transparentes aos modelos dos ambientes. A introdução dessas paredes é apenas um artifício geométrico que facilitará as determinações de visibilidade e o cálculo dos feixes de propagação. Assim, não existe uma interação física entre estas paredes transparentes e os raios sendo calculados. Nas figuras dos ambientes, representamos paredes opacas (oclusores) por segmentos de reta contínuos e paredes transparentes por segmentos tracejados.

A decomposição do espaço em células convexas nos permite avaliar de forma bastante eficiente a visibilidade de um oclusor a partir de um determinado ponto, já que ela induz uma ordenação espacial entre os mesmos.

A primeira observação que podemos fazer sobre a decomposição é que não existe oclusão entre as arestas que definem a fronteira de uma mesma célula (isto pode ser verificado diretamente a partir da definição de convexidade). Assim, não é necessário ordenar as arestas de uma mesma região para determinações de visibilidade, já que todas estas arestas são necessariamente visíveis a partir de qualquer ponto pertencente à região por elas delimitada.

A segunda observação está relacionada à determinação de um subconjunto dos oclusores do ambiente que podem ser de interesse para determinações de visibilidade. Uma vez escolhida uma região de interesse (que pode ser a área coberta por um feixe de som ou o campo de visão de uma câmera) em uma célula C, podemos limitar nossa atenção às células imediatamente adjacentes a C através das arestas contidas na região de interesse, já que entre as células adjacentes apenas estas contêm arestas que podem ser visíveis (estar contidas na região de interesse). Utilizando a mesma região de interesse em uma das células adjacentes selecionadas, podemos determinar quais de suas arestas estão contidas na região de interesse e selecionar novas células a serem examinadas. Isso nos dá um algoritmo bastante eficiente para enumerar todas as arestas do ambiente que se encontram dentro da região de interesse. Note que esse algoritmo não apenas enumera as arestas de interesse, mas também as enumera na ordem em que as mesmas devem ser consideradas nos cálculos de visibilidade. Como veremos mais adiante, este percorrer de células é a idéia básica dos algoritmos de *beam tracing*.

A Figura 4.3 ilustra o percorrer de células. Considere a região de interesse (delimitada pelos segmentos de reta com estilo traço-ponto) definida a partir do ponto P na célula C_0. Como a região de interesse contém as arestas AB e BC, o algoritmo prossegue para as células C_1 e C_2. Em C_1,

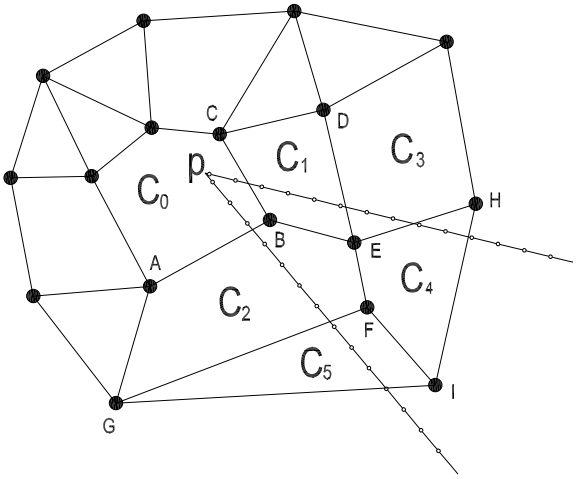


Figura 4.3: Percorrendo uma Decomposição em Células Convexas

podemos ver que a aresta DE está contida na região de interesse, o que leva o algoritmo para a célula C_3 e depois, devido a interseção da aresta EH, para C_4. Já a interseção com a aresta BE, leva o algoritmo para a célula C_2. Voltando a C_2, podemos ver que a arestas FG e EF também estão contidas na região de interesse, o que leva à visita das células C_5 e C_4, respectivamente.

Note que o algoritmo apresentado acima não leva em consideração a opacidade das arestas. Caso a opacidade fosse levada em consideração e alguma das arestas contidas na região de interesse fosse opaca, não seria necessário visitar a célula adjacente através desta aresta, já que nenhuma de suas arestas seriam visíveis devido à oclusão causada pela aresta opaca.

Para que o percorrer da decomposição descrito acima seja possível, necessitamos de uma estrutura de dados que armazene as informações de adjacência das células. Em [32] foi utilizada uma estrutura de grafo em que os vértices correspondem às células e os arcos correspondem às paredes transparentes comuns a duas células adjacentes. Esta estrutura leva em consideração a opacidade das paredes do ambiente, de forma que a existência de um arco entre dois vértices desse grafo de adjacência garante a existência de uma passagem entre duas células. A Figura 4.4 ilustra a decomposição de um ambiente em células convexas e seu grafo de adjacência.

Apesar de permitir percorrer a decomposição celular corretamente, o grafo de adjacência não contém informação suficiente para permitir a inclusão de feixes de difração. A necessidade de identificar quinas e medir o ângulo entre suas paredes faz com que precisemos saber quais as arestas incidentes em um vértice. Assim, é mais apropriado utilizar uma estrutura topológica completa, como uma *half-edge* [51].

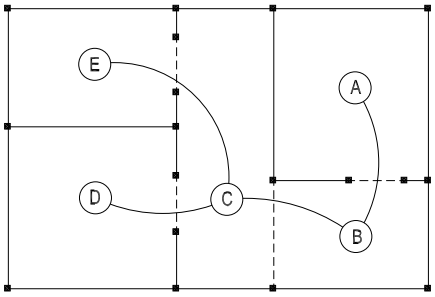


Figura 4.4: Grafo de Adjacência de Células de um Ambiente

Como geralmente não há alteração na posição dos oclusores de um ambiente, a decomposição celular só precisa ser construída uma vez, o que, ao contrário do que ocorre quando utilizamos a ordenação direta dos oclusores, pode ser realizado em uma etapa de pré-processamento.

A decomposição do espaço em células convexas geralmente é feita a partir de duas estratégias diferentes [20]. A primeira, e mais comum, consiste em realizar um particionamento recursivo do espaço, como é feito no algoritmo clássico para construção de árvores BSP [51]. A segunda estratégia consiste na utilização de algoritmos de triangulação com restrições.

As próximas seções descrevem a implementação realizada para as duas estratégias mencionadas. Primeiramente, fazemos uma breve descrição sobre árvores BSP e suas propriedades. Depois mostramos como o algoritmo clássico para construção destas árvores pode ser modificado para criar a decomposição do ambiente em células convexas. Em seguida, descrevemos a estratégia baseada no uso de triangulações.

Como veremos, a utilização de uma triangulação não é muito apropriada para o *beam tracing*. Assim, nossa implementação realiza uma simplificação da decomposição celular obtida a partir de uma triangulação, que acreditamos ser a melhor alternativa para a criação da decomposição celular do ambiente.

4.1.1 Particionamento Recursivo do Espaço

Considere uma reta r, definida no plano pela equação $ax + by + c = 0$. Esta reta divide o plano em três regiões convexas distintas: a região localizada sobre a reta (definida pelos pontos que satisfazem a equação de r) e dois semi-espaços. Estes semi-espaços são geralmente chamados de semi-espaços positivo e negativo, devido às desigualdades que os definem. Dizemos que o semi-espaço positivo, definido pelos pontos que satisfazem

$ax + by + c > 0$, corresponde à região localizada acima da reta r. Da mesma forma, o semi-espaço negativo corresponde à região localizada abaixo de r e é definido pelos pontos que satisfazem $ax + by + c < 0$.

A decomposição em células convexas pode ser feita recursivamente, particionando uma região através da definição de uma reta e depois particionando as regiões geradas com novas retas. Como a interseção de regiões convexas deve ser uma outra região convexa (ou vazia), a convexidade das regiões criadas está garantida. Note que este particionamento não é limitado a espaços de dimensão 2. Em três dimensões, por exemplo, o mesmo pode ser obtido definindo planos ao invés de retas. De forma geral, a subdivisão de uma espaço d-dimensional pode ser obtida através da definição de hiperplanos de dimensão $d - 1$.

Uma árvore BSP (*binary spatial partition tree*) é uma árvore binária que consiste em um registro do processo de um particionamento recursivo. Cada nó interno de uma árvore BSP corresponde a um dos hiperplanos que dividem o espaço e suas folhas correspondem às regiões convexas definidas pelo particionamento. Sub-árvores da direita geralmente correspondem a regiões acima de uma reta e as da esquerda a regiões abaixo da mesma.

A eficiência de diversos algoritmos muitas vezes está associada à existência de uma ordem que pode ser estabelecida entre os objetos por eles manipulados. Árvores binárias de busca [35] são exemplos clássicos de como uma ordenação de elementos pode ser explorada para acelerar algoritmos. Podemos considerar uma árvore BSP como o equivalente geométrico de uma árvore binária de busca.

A Figura 4.5 ilustra como um particionamento recursivo pode ser utilizado para definir regiões no plano. As regiões definidas e sua árvore BSP são mostradas na figura. As sub-árvores correspondentes a regiões localizadas acima da reta de corte possuem o sinal + sobre suas arestas. O sinal − foi utilizado para indicar as regiões localizadas abaixo de uma reta de corte. Observe a região D na figura. Esta região é definida pela interseção da região acima da reta l_1, da região acima de l_2 e da região abaixo de l_4. Note como isso é registrado na árvore BSP. A folha associada à região D é atingida quando percorremos o ramo positivo de l_1, o ramo positivo de l_2 e, finalmente, o ramo negativo de l_4.

Além de permitir representar o particionamento do plano, quando balanceadas, árvores BSP podem ser eficientes estruturas de acesso para localizar objetos. Quando associamos objetos às folhas da árvore (objetos contidos nas regiões) ou aos nós internos (objetos contidos nos hiperplanos), podemos nos valer da ordem existente entre os hiperplanos de corte para

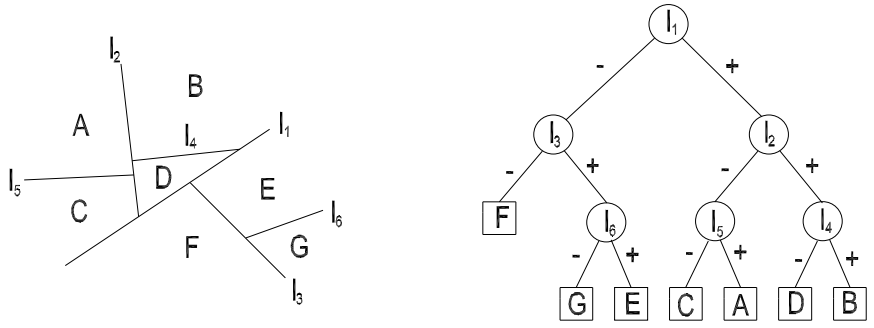


Figura 4.5: Particionamento Recursivo do Plano e sua Árvore BSP

realizar a localização. Suponha que necessitamos determinar se existe um objeto localizado em um ponto P do plano. Para responder a esta pergunta precisamos apenas localizar qual a célula convexa que contém este ponto e verificar se os objetos associados a esta região contêm o ponto sendo testado. A localização da célula é feita percorrendo a árvore BSP, seguindo as direções indicadas pela posição do ponto P em relação aos nós internos da árvore. Se P estiver abaixo da reta associada a um nó, deve-se prosseguir a procura na sub-árvore da esquerda e, na sub-árvore da direita, caso P esteja localizado acima da reta.

Como os feixes traçados no algoritmo de beam tracing são associados às regiões convexas, a árvore BSP pode ser utilizada como uma estrutura de acesso eficiente para localizar os feixes que incidem sobre um determinado ponto do espaço.

Árvores BSP - Algoritmo Clássico

A utilização mais comum de árvores BSP é na remoção de superfície escondida em ambientes definidos por oclusores poligonais. Como vimos anteriormente, a remoção de superfície escondida é, basicamente, um problema de ordenação. Escolhendo os hiperplanos que contêm os oclusores poligonais para realizar a partição, podemos utilizar a ordenação existente entre os nós de uma árvore BSP para determinar o relacionamento espacial existente entre os oclusores de uma cena. A construção de uma árvore BSP é facilmente realizada através de um algoritmo recursivo, que descreveremos agora para o caso bidimensional.

Considere um conjunto S de oclusores 2D (segmentos de reta) localizados no plano. Caso S seja vazio, não há nada a ser feito e a árvore BSP resultante é uma árvore vazia. Este é o caso base do algoritmo recursivo. Veremos agora o passo recursivo do algoritmo que consistirá na divisão de

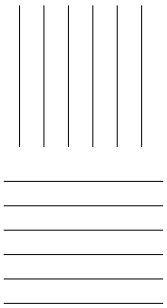


Figura 4.6: Oclusores que Podem Gerar um Número Quadrático de Células

S em conjuntos menores e na aplicação da recursão para cada um destes conjuntos.

Caso S não seja vazio, podemos escolher um de seus segmentos e determinar a reta h que o contém. Escolhida h, devemos dividir S em três subconjuntos distintos: S_+, S_- e S_0. Sendo que estes contêm, respectivamente, os elementos localizados acima, abaixo e sobre a reta h. Se algum elemento de S se localizar tanto abaixo quanto acima da reta, devemos dividí-lo em dois novos segmentos, correspondentes às partes do segmento que se localizam abaixo e acima da reta. Esses dois novos segmentos são então adicionados a S_+ e S_-, como for apropriado, e o segmento original é descartado. Ao realizar a divisão do conjunto, devemos criar a árvore BSP correspondente. Associamos h e S_0 à raiz da árvore sendo criada. As sub-árvores da árvore BSP são calculadas, recursivamente, pelo particionamento de S_+ e S_-.

Ao dividir os oclusores, o processo de particionamento garante que a posição de um oclusor em relação a outro pode ser determinada sem ambigüidade. Assim, como os oclusores estão diretamente associados aos nós internos da árvore BSP, estes podem ser utilizados para determinar a posição relativa entre dois oclusores. Esta posição será a mesma existente, na árvore BSP, entre os nós a que os oclusores estão associados.

A escolha de um hiperplano de corte é crítica na construção de árvores BSP, pois dela dependem o número final de oclusores (que varia devido a divisão dos mesmos) e o balanceamento da árvore resultante. A Figura 4.6 ilustra um caso que pode gerar um número quadrático de objetos, caso as retas verticais sejam selecionadas como os hiperplanos de corte.

Infelizmente, criar uma árvore BSP ótima pode ser inviável, pois para n oclusores, podem existir $n!$ árvores diferentes, o que inviabiliza uma busca exaustiva. Assim, diversas heurísticas são utilizadas para selecionar o hiperplano de corte, como escolher o hiperplano com a maior superfície

opaca ou o hiperplano que minimize o número de divisões de objetos realizadas. Note que minimizar o número de divisões realizadas em um passo do algoritmo não garante que no final teremos um número mínimo de objetos.

Esta revisão sobre árvores BSP teve como objetivo mostrar apenas as características de árvores BSP relevantes para os algoritmos de *beam tracing*, como a utilização das árvores para representar regiões convexas e para localizar objetos eficientemente. Para um tratamento mais completo sobre árvores BSP e suas aplicações sugerimos a consulta de [51, 22, 7]. Veremos agora como o algoritmo apresentado para construção de uma árvore BSP pode ser modificado para criar a decomposição do ambiente em células convexas.

Decomposição em Células Convexas - Modificando o Algoritmo de BSP

A decomposição do ambiente em células convexas é bastante similar à construção da árvore BSP para remoção de superfície escondida. Em ambos os procedimentos, a operação básica consiste em particionar um objeto com um hiperplano. A diferença entre os dois está no tipo de objeto que cada procedimento manipula.

A construção da decomposição do ambiente em células convexas depende da resolução de dois problemas: inserção das arestas transparentes para definição da fronteira das células convexas e obtenção da informação de adjacência entre as células. Ambos os problemas podem ser resolvidos se, durante o processo de particionamento, não trabalharmos com oclusores individuais. Ao invés disso, será utilizada uma estrutura que chamamos de *reta suporte*.

Como no algoritmo clássico de BSP, os hiperplanos de corte selecionados para criar a decomposição celular do ambiente são os mesmos que contêm os oclusores. Sabemos que a interseção das regiões definidas pelos hiperplanos de corte definem as células convexas do ambiente e que suas fronteiras são definidas pelos próprios hiperplanos. Caso os oclusores ocupassem a mesma região ocupada pelos hiperplanos de corte, teríamos definidas as fronteiras das células convexas (a fronteira seria definida pelos próprios oclusores). Infelizmente, isto dificilmente ocorrerá, já que os oclusores de um ambiente geralmente têm área finita e os hiperplanos não. Assim, nos valemos das arestas transparentes para cobrir toda a área ocupada por um hiperplano de corte e, conseqüentemente, obter as fronteiras das células.

Uma reta suporte é a representação de um hiperplano de corte, composta de todos os oclusores que ele contém e das arestas transparentes necessárias para cobrir toda sua área. Como precisamos obter a informação de adjacência da decomposição celular, as retas suporte devem armazenar a informação de adjacência existente entre suas arestas.

A construção da estrutura topológica completa pode ser feita durante o processo de particionamento, que agora lida com as retas suporte ao invés de oclusores individuais. Existe, entretanto, uma alternativa mais fácil para criar esta estrutura. A alternativa consiste em, durante o processo de particionamento, manter a informação de adjacência apenas entre os vértices e arestas dos ambientes, utilizando a mesma estrutura de grafo que os define. Desta forma, podemos representar uma reta suporte como um conjunto de arestas deste grafo. Note que a inclusão de arestas transparentes a uma reta suporte é, na verdade, a inclusão de arestas transparentes ao grafo que define um ambiente.

Mantendo apenas a informação de adjacência entre vértices e arestas, ao final do processo de particionamento, teremos uma instância da estrutura de grafo adicionada dos vértices e arestas necessários para a definição das fronteiras das células. A partir deste grafo, é possível montar a estrutura topológica de que necessitamos. A informação de adjacência entre vértices e arestas já está contida no grafo e os relacionamentos entre faces e arestas podem ser descobertos a partir dos ciclos de arestas existentes, que corresponderão às células convexas (ou faces) da decomposição celular.

Dependendo do ambiente sendo decomposto em células convexas e da ordem em que as retas suporte são escolhidas, regiões convexas abertas podem ser criadas (como as regiões A e B na Figura 4.5). Isto pode ser evitado inserindo no grafo do ambiente as arestas que formam o fecho convexo de seus vértices, antes que o processo de particionamento seja iniciado. Estas arestas incluídas devem ser, logicamente, transparentes.

A Figura 4.7 contém o pseudo-código que ilustra o funcionamento do algoritmo implementado para construir a decomposição celular do ambiente. A primeira tarefa realizada pelo algoritmo é a construção de uma lista de retas suporte a partir das arestas presentes no grafo que define o ambiente. Depois, uma das retas suporte é selecionada e a lista de retas suporte é então dividida por esta em duas outras listas. A função que realiza a partição é então chamada recursivamente sobre estas duas listas.

A seleção de uma das retas suporte corresponde à seleção de um hiperplano de corte no algoritmo clássico para construção de árvores BSP. Assim, as mesmas heurísticas utilizadas para seleção de um hiperplano po-

```
DecomposiçãoCelular(Grafo G)
{
   lrs = Cria_Lista_de_Retas_Suporte(G)
   Particiona(G, lrs)
   Cria_Estrutura_Topológica(G)
}


Particiona(Grafo g, ListaRetasSuporte lrs)
{
   divisor = Seleciona_Reta_Suporte_de_Corte(lrs)

   lrsAcima = Cria_Lista_Vazia_de_Retas_Suporte()
   lrsAbaixo = Cria_Lista_Vazia_de_Retas_Suporte()

   for i = 1 to length(lrs)
   {
      retaCorrent = lrs[i]
      Remove_Reta_Suporte(lrs[i], lrs)

      if Acima(retaCorrente, divisor)
         Adiciona_Reta_Suporte(retaCorrente, lrsAcima)
      else if Abaixo(lrs[i], divisor)
         Adiciona_Reta_Suporte(retaCorrente, lrsAbaixo)
      else
      {
         Divide_Reta_Suporte(g, divisor, retaCorrente,
            retaAcima, retaAbaixo)
         Adiciona_Reta_Suporte(retaAcima, lrsAcima)
         Adiciona_Reta_Suporte(retaAbaixo, lrsAbaixo)
      }
   }

   Particiona(g, lrsAcima)
   Particiona(g, lrsAbaixo)
}
```

Figura 4.7: Algoritmo para Decomposição do Ambiente em Células Convexas

dem ser utilizadas na seleção de uma reta suporte. Em nossa implementação selecionamos, a cada passo, a reta suporte que apresenta a maior área opaca.

Os procedimentos que implementam a criação e a divisão de retas suporte são o tópico da próxima seção.

A Figura 4.4 ilustra a decomposição celular construída pelo algoritmo apresentado na Figura 4.7 a partir do ambiente definido na Figura 4.1.

Criação e Divisão de Retas Suporte

O primeiro passo para construir a decomposição celular de um ambiente consiste em construir suas retas suporte. Esta construção começa com a divisão dos oclusores do ambiente em conjuntos maximais de oclusores colineares, isto é, se dois oclusores são colineares, eles devem pertencer a um mesmo conjunto. Cada um destes conjuntos dará origem a uma reta suporte diferente.

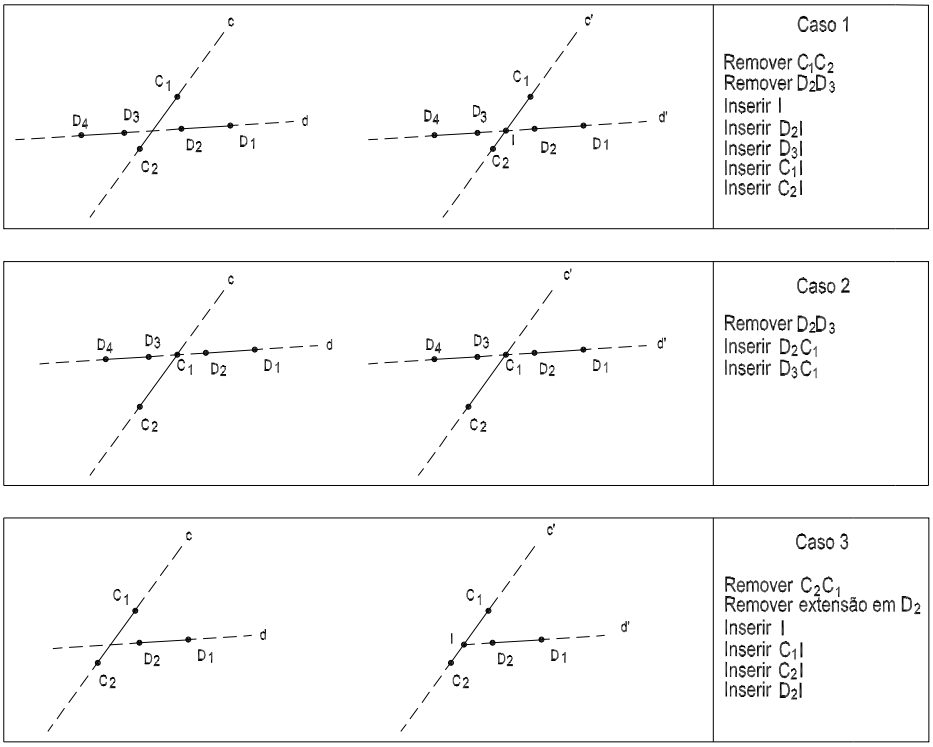


Figura 4.8: Interseção de Retas Suporte

Uma vez criados os conjuntos de oclusores colineares, devemos inserir em cada conjunto as arestas transparentes. Ordenando os oclusores ao longo do hiperplano que os contém, podemos criar as arestas transparentes facilmente. Estas devem ser criadas de forma a preencher as lacunas existentes entre os oclusores. Devemos criar também duas arestas transparentes infinitas, uma em cada extremidade da reta suporte. Chamamos essas arestas infinitas de *extensões* de uma reta suporte. Note que a inserção de uma aresta a uma reta suporte causa a inserção de uma aresta ao grafo que define o ambiente.

Criadas as retas suporte, o algoritmo de decomposição celular prossegue para o particionamento recursivo, cuja operação básica é a divisão de uma reta suporte por outra.

Ao contrário do que ocorria no caso da árvore BSP criada para visualização, a divisão de uma reta suporte afeta tanto a reta de corte quanto a reta cortada. Isso ocorre devido à necessidade de atualizar a informação de adjacência do grafo sempre que ocorre uma divisão. A Figura 4.8 ilustra alguns casos existentes na divisão de uma reta suporte d por uma reta suporte de corte c.

No primeiro caso, d e c não se cruzam em um vértice do grafo. Logo, devemos criar um novo vértice, correspondente ao ponto de interseção

das retas. Criado o vértice de interseção (I), devemos remover as arestas envolvidas na interseção ((C_1, C_2) e (D_2, D_3)) e inserir quatro novas arestas: (C_1, I), (C_2, I), (D_2, I) e (D_3, I).

No segundo caso, c e r se cruzam em um vértice já existente, C_1. Assim, é necessário apenas remover a aresta de d que foi cortada e inserir duas novas arestas (D_2, C_1) e (D_3, C_1).

O terceiro caso ilustra o procedimento adotado quando uma das extensões de d cruza c fora de um vértice. Como no primeiro caso, devemos inserir um novo vértice I no grafo, remover a aresta de c que foi cortada ((C_1, C_2)),e inserir duas novas arestas: (C_1, I) e (C_2, I). Devemos inserir também uma terceira aresta, (D_2, I), correspondente à parte da extensão de d necessária para a definição da fronteira das células. Como não existem oclusores de d no lado esquerdo de c, a porção de d localizada à esquerda de c pode ser desprezada. Note que se todos os elementos de d estivessem localizados à esquerda de c, a porção à direita de d teria sido desprezada.

Caso a extensão de d cruzasse c em um vértice, apenas uma nova aresta teria sido adicionada ao grafo, pois nenhum vértice seria criado e uma parte de d poderia ser desprezada.

Desvantagens do Particionamento Recursivo

Como vimos, quando utilizamos um particionamento recursivo para a construção da decomposição do ambiente em células convexas, novos vértices podem ser inseridos no grafo do ambiente. Esta inserção pode ser prejudicial, pois ela aumenta o número de feixes traçados desnecessariamente, principalmente quando vértices isolados são criados (vértices cujas arestas adjacentes são transparentes).

Como veremos mais adiante, os feixes utilizados no algoritmo de *beam tracing* são delimitados pelas arestas do ambiente. Assim, sempre que um feixe incide sobre um vértice isolado, ele se fragmenta em um número de feixes igual ao número de arestas transparentes que incidem no vértice. Esta fragmentação do feixe não é necessária, pois não existe nenhuma mudança nas propriedades do feixe, já que não ocorre a reflexão ou difração do mesmo ao incidir sobre este vértice. Como os feixes de som são refletidos e difratados no ambiente, podemos esperar que eles passem pelos vértices isolados repetidas vezes, o que pode tornar a fragmentação relevante em relação ao número de feixes criados e, conseqüentemente, em relação à quantidade de memória utilizada.

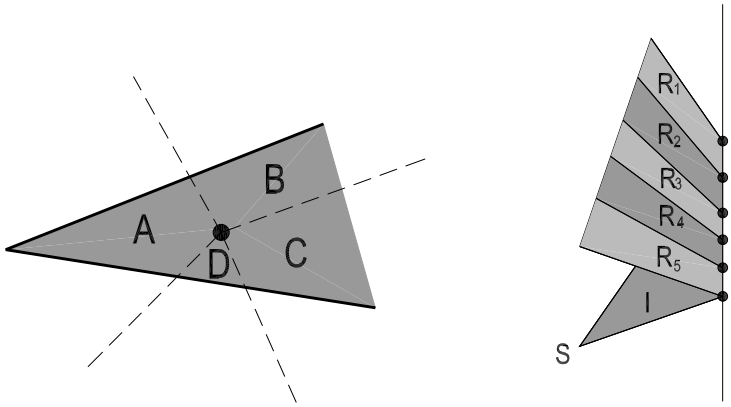


Figura 4.9: Fragmentação de um Feixe

Um tipo similar de fragmentação ocorre quando um feixe incide sobre uma superfície opaca composta de arestas colineares. Para cada aresta contida no feixe incidente, teremos um feixe de reflexão diferente. A Figura 4.9 ilustra os dois tipos de fragmentação.

Uma outra desvantagem no uso de partições recursivas é a complexidade das operações envolvidas quando trabalhamos em três dimensões. Um *plano suporte*, o equivalente 3D de uma reta suporte, seria, basicamente, uma subdivisão planar. Assim, a divisão de um plano suporte por outro pode ser uma operação bastante complexa dada a necessidade de criar a informação de adjacência das células, como é feito na divisão de uma reta suporte.

4.1.2 Decomposição Celular Através de Triangulações com Restrições

Uma triangulação de um conjunto de pontos do plano consiste em uma subdivisão planar composta de triângulos, ou seja, a triangulação de um conjunto de pontos é também uma decomposição do plano em células convexas. Triangulações com restrições são triangulações em que algumas arestas que devem ser parte da triangulação final são especificadas previamente. Estas arestas são as restrições que devem ser obedecidas por um algoritmo que realize a triangulação.

Podemos utilizar uma triangulação com restrições para construir a decomposição de um ambiente em células convexas. Para isso, devemos apenas realizar a triangulação dos vértices existentes no grafo do ambiente, tendo os oclusores como restrições.

Uma das vantagens na utilização de uma triangulação é que, ao

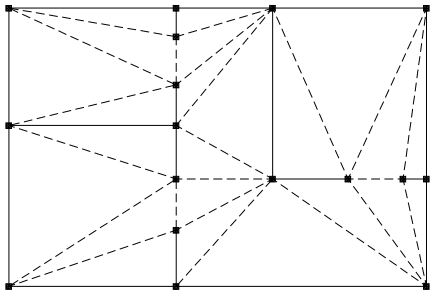


Figura 4.10: Triangulação de um Ambiente

contrário do que ocorre no particionamento recursivo, o número de células criadas pela triangulação é linear em relação ao número de vértices e, conseqüentemente, de arestas do ambiente. Um menor número de células significa um menor número de arestas transparentes no ambiente e, conseqüentemente, uma menor fragmentação dos feixes. Além disso, triangulações não criam novos vértices, o que minimiza ainda mais a fragmentação.

Em nossa implementação, utilizamos uma triangulação de Delaunay com restrições. Triangulações de Delaunay [51] são muito populares, pois maximizam os ângulos internos dos triângulos gerados. Essa maximização faz com que os triângulos de uma triangulação sejam o mais equiláteros possível, tornando a mesma mais apropriada para diversas aplicações. Para a construção de decomposições em células, entretanto, acreditamos que outras triangulações podem ser utilizadas sem comprometer a qualidade da decomposição criada. A Figura 4.10 ilustra a triangulação de Delaunay com restrições criada para o ambiente ilustrado na Seção 4.1.

Note na Figura 4.10 que as arestas inseridas por uma triangulação definem um número grande de células, que na maior parte dos casos poderia ser menor. Assim como ocorre no particionamento recursivo do plano, o número excessivo de arestas transparentes pode também aumentar significativamente o número de feixes traçados. Felizmente, isso pode ser contornado através da remoção de algumas arestas da triangulação.

Podemos remover uma aresta da triangulação sempre que sua remoção não afete a convexidade das células. Isto pode ser determinado localmente, verificando a vizinhança de cada aresta. A convexidade das células estará garantida se não removermos nenhuma aresta que resulte na criação de um ângulo maior que 180 graus dentro de uma célula. A Figura 4.11 ilustra a vizinhança de duas arestas. Note que a aresta AB não pode ser removida, pois resultaria na criação de uma célula côncava já que o ângulo α, entre AD e AC, é maior que 180 graus. Da mesma forma, a aresta GH pode ser removida, pois os ângulos criados por sua remoção (γ, entre GI e GJ, e δ,

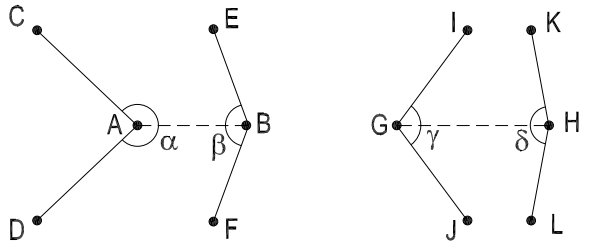


Figura 4.11: Critério Local para Remoção de Arestas

entre HK e HL) são menores que 180 graus.

Infelizmente, a ordem em que as arestas de uma triangulação são removidas pode influenciar o resultado final. Como ocorre no uso do particionamento recursivo, o número de soluções possíveis é muito grande para permitir uma busca exaustiva pela melhor decomposição celular. Assim, a escolha das arestas a serem removidas pode ser feita através de uma heurística.

Além do grande número de soluções possíveis (para n arestas transparentes, há $n!$ possíveis soluções), temos o problema de definir o que seria a melhor decomposição celular. Como os feixes traçados são refletidos e difratados diversas vezes, podemos esperar que todo o ambiente será coberto por feixes. Assim, as decomposições que possuam células com grandes áreas, que seriam grandes áreas não obstruídas para os feixes, ou as que possuam o menor número de arestas transparentes são boas candidatas para serem soluções ótimas, no sentido de minimizar a fragmentação dos feixes. Entre esses dois critérios, o primeiro pode ser mais facilmente transformado em uma heurística para a seleção da aresta a ser removida.

Como a triangulação de Delaunay gera triângulos o mais equiláteros possível, podemos considerar que o comprimento de uma aresta está diretamente relacionado com a área dos triângulos a que ela pertence. Ou seja, se uma aresta a tem um comprimento maior que uma aresta b, consideramos que a área dos triângulos associados a a é maior que a área dos triângulos associados a b. Assim, a heurística implementada consiste em ordenar decrescentemente as arestas transparentes pelo seu comprimento e tentar removê-las nesta ordem. Com isso esperamos que, ao remover a cada passo a maior aresta, possamos maximizar a área das células da decomposição. Também foi implementada a remoção das arestas na ordem crescente de seus comprimentos. A Figura 4.12 ilustra as decomposições resultantes quando aplicamos as duas heurísticas à triangulação exibida na Figura 4.10.

Um algoritmo similar ao implementado para o caso bidimensional pode ser utilizado em três dimensões. Seu primeiro passo seria a construção de

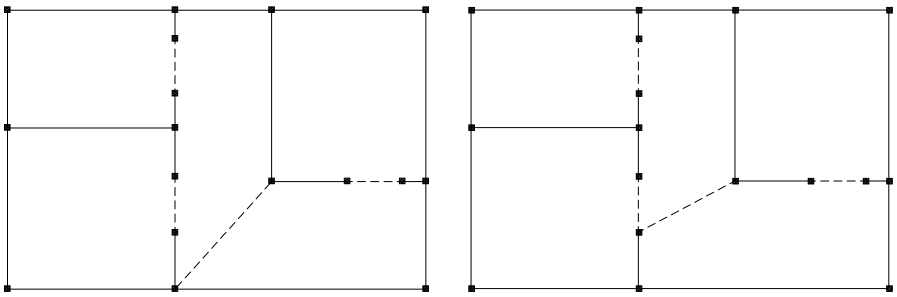


Figura 4.12: Remoção de Arestas da Triangulação

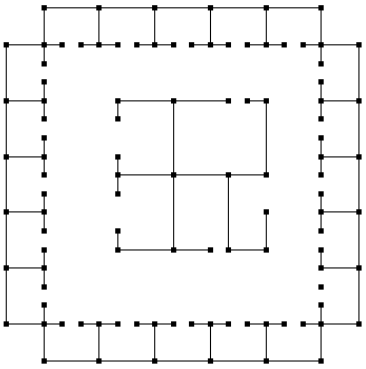
uma triangulação 3D (ou tetraedralização) de um conjunto de pontos obedecendo a restrições que seriam os oclusores de um ambiente. Em seguida, uma etapa de simplificação, correspondente à remoção de arestas transparentes, seria realizada. Isto envolveria a junção de diversos tetraedros em poliedros mais complexos. Apesar de depender de uma primitiva bastante complexa (a triangulação 3D), acreditamos que este procedimento seja mais adequado para a construção de decomposições celulares, pois este pode oferecer melhores resultados que o uso de um particionamento recursivo.

4.1.3 Exemplos de Decomposições de Ambientes em Células Convexas

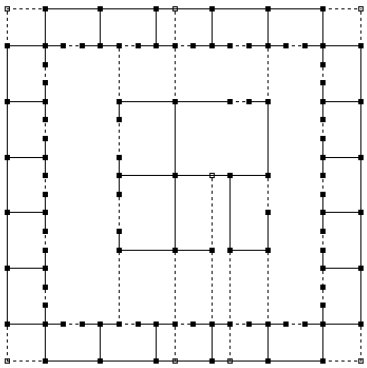
As Figuras 4.13 e 4.14 apresentam um ambiente de entrada e suas decomposições em células convexas, obtidas pelo uso dos algoritmos implementados. Na primeira, podemos ver que o particionamento recursivo apresenta bons resultados, comparáveis aos obtidos através da simplificação de triangulações. Isso ocorreu devido ao pequeno número de retas suporte existentes no ambiente de entrada. Note o grande número de oclusores colineares. Para exemplos mais irregulares, como o da Figura 4.14, a diferença entre as decomposições criadas pelos diferentes métodos aumenta significativamente.

4.2 Representação dos Feixes

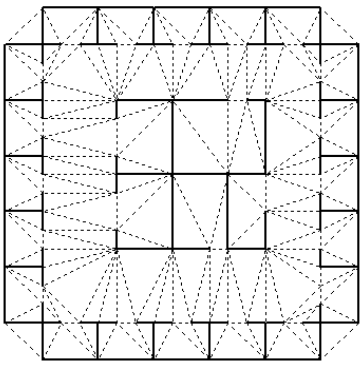
A vantagem do *beam tracing* em relação aos demais algoritmos apresentados anteriormente para a descoberta de caminhos de propagação consiste em que, lidando com regiões ao invés de raios individuais, é possível representar a propagação de um infinito número de raios (todos aqueles contidos em uma região) de forma bastante compacta.



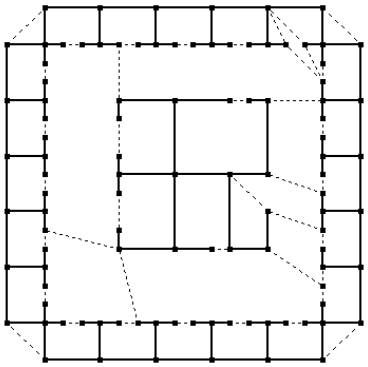
a) Ambiente de Entrada



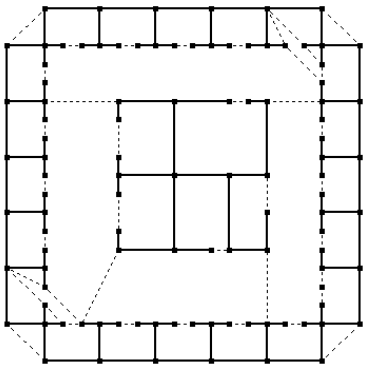
b) Particionamento Recursivo



c) Triangulação de Delaunay



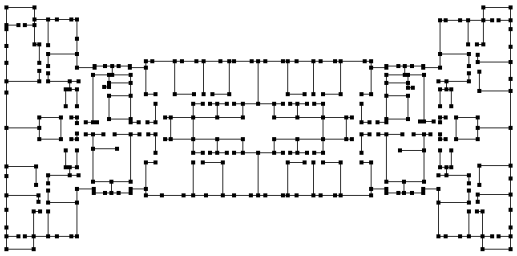
d) Remoção das Arestas em Ordem Crescente de Comprimento



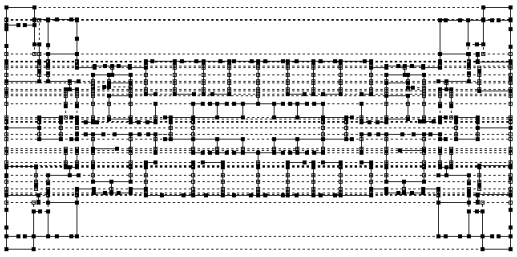
e) Remoção das Arestas em Ordem Decrescente de Comprimento

Decomposição	Células	Vértices	Oclusores	Arestas Transparentes
a	–	103	101	0
b	40	110	105	44
c	180	103	101	181
d	39	103	101	40
e	39	103	101	40

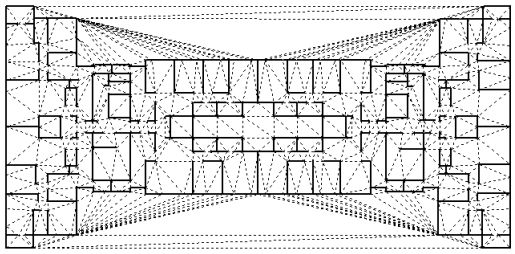
Figura 4.13: Comparação entre Diferentes Decomposições em Células



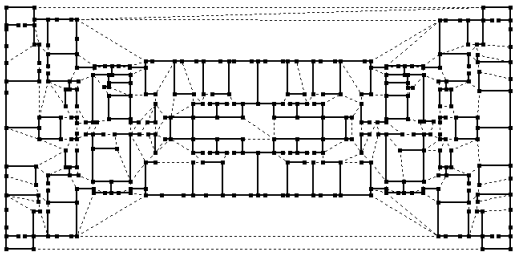
a) Ambiente de entrada



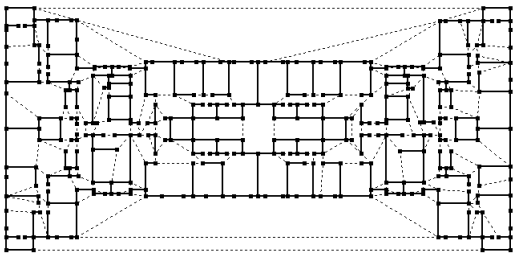
b) Particionamento Recursivo



c) Triangulação de Delaunay



d) Remoção das Arestas em Ordem Crescente de Comprimento



e) Remoção das Arestas em Ordem Decrescente de Comprimento

Decomposição	Células	Vértices	Oclusores	Arestas Transparentes
a	–	380	378	0
b	584	883	851	615
c	724	380	378	725
d	214	380	378	215
e	206	380	378	207

Figura 4.14: Comparação entre Diferentes Decomposições em Células

O primeiro passo para a implementação do *beam tracer* para propagação do som em duas dimensões consiste na escolha de uma representação apropriada para os feixes de propagação. Esta escolha pode ser feita analisando a facilidade com que cada representação permite implementar as operações básicas exigidas de um feixe.

Devemos ser capazes de realizar as seguintes operações com um feixe:

- Pertinência (determinar se um ponto está contido no feixe)
- Detecção de interseção com oclusores
- Cálculo da interseção com oclusores

Como é feito em problemas de programação linear [31], poderíamos representar os feixes como a região do plano definida pela interseção de um conjunto de semi-espaços. A operação de pertinência pode ser facilmente implementada sobre essa representação (basta verificar se o ponto sendo testado satisfaz a cada uma das inequações que definem a região do plano). Além disso, essa representação também é apropriada para feixes 3D. Mas, infelizmente, as operações de interseção não são facilmente implementadas.

Feixes também podem ser representados em duas dimensões por dois raios e um ponto de referência. Os raios são utilizados para delimitar a fronteira do feixe e o ponto de referência para indicar qual das duas regiões definidas pelos raios corresponde à área ocupada pelo feixe. Essa pode ser a alternativa mais intuitiva e, talvez, a mais eficiente para o caso bidimensional, mas não pode ser facilmente estendida para três dimensões.

A alternativa escolhida para a nossa implementação foi a mesma utilizada em [14] na implementação de seu *beam tracer* para visualização, onde cada feixe é representado por um sistema de coordenadas localizado em sua origem e por sua seção transversal, descrita no sistema de coordenadas do feixe. Essa representação é adequada tanto para feixes 2D quanto para feixes 3D e também permite implementar facilmente as operações básicas exigidas de um feixe, como veremos a seguir.

4.2.1 Representação Projetiva para Feixes de Propagação

Nesta seção veremos como são criados feixes utilizando a representação projetiva, como implementar as operações básicas citadas na seção anterior e também como criar os diferentes tipos de feixe necessários para o cálculo da propagação.

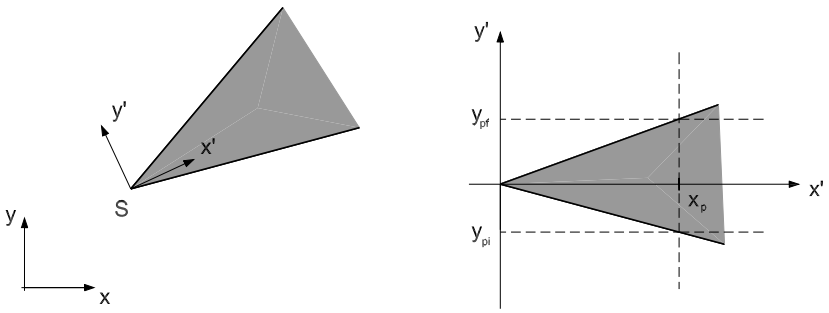


Figura 4.15: Sistema de Coordenadas de um Feixe

Criação de Feixes

Como mencionamos anteriormente, os feixes 2D utilizados na nossa implementação são representados por um sistema de coordenadas e por sua seção transversal, expressa no sistema de coordenadas do feixe. O sistema de coordenadas é armazenado na forma de duas matrizes de transformação: M_{GL}, que faz a transformação do sistema de coordenadas global para o sistema de coordenadas local (ou do feixe) e M_{LG}, que faz a transformação no sentido inverso.

A seção transversal dos feixes consiste em um segmento de reta vertical e pode ser representada com três números reais: x_p, y_{pi} e y_{pf}. x_p indica a posição da seção transversal sobre o eixo x', enquanto y_{pi} e y_{pf} indicam os limites da seção no eixo y'.

A Figura 4.15 ilustra um feixe de propagação no sistema de coordenadas global (definido pelos eixos x e y), mostrando também o sistema de coordenadas do feixe (eixos x' e y') e as variáveis que o definem. Note que no sistema de coordenadas do feixe, x_p será sempre maior que zero.

Uma vez obtida a representação projetiva de um feixe, o algoritmo de propagação poderá criar automaticamente os feixes decorrentes de reflexões, transmissões e difrações. Mas o feixe inicial deve ser criado explicitamente. Apesar de permitir implementar as operações básicas de forma bastante eficiente, a determinação das matrizes de projeção e da seção transversal não é a mais intuitiva para manipulação por pessoas. É mais fácil especificarmos um feixe a partir de dados como a posição global de sua fonte, sua abertura e sua inclinação em relação ao sistema de eixos global. Mostraremos agora como obter a representação projetiva de um feixe a partir desses dados.

Considere um feixe com fonte no ponto S (s_x, s_y), como ilustrado na Figura 4.16. Escolhendo um raio de referência r (contido no feixe), podemos utilizar sua inclinação com o eixo x (ângulo θ) como a inclinação do feixe. Dessa forma, a abertura do mesmo é definida pelos dois ângulos ϕ e α,

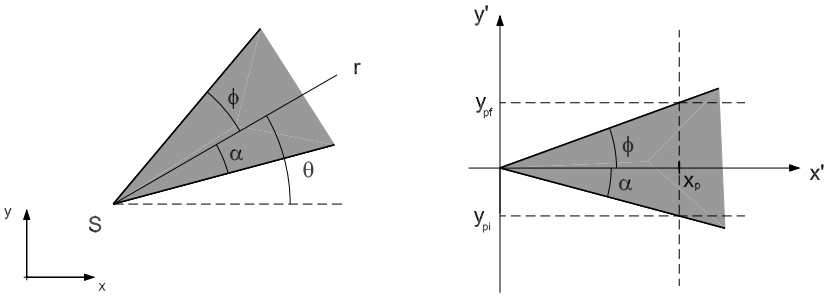


Figura 4.16: Construindo um Feixe

indicados na figura.

Uma vez escolhida a posição da seção transversal sobre o eixo x' (qualquer valor maior que zero pode ser escolhido) os limites da seção podem ser determinados facilmente através das seguintes fórmulas:

$$y_{pf} = x_p \tan(\phi)$$
$$y_{pi} = -x_p \tan(\alpha)$$

Já as matrizes de transformação podem ser determinadas a partir de transformações geométricas mais simples de rotação e translação, como indicado nas equações abaixo, onde $R(\alpha)$ é uma matriz de rotação de α graus e $T(x, y)$ uma matriz de translação.

$$M_{LG} = T(s_x, s_y)R(\theta)$$
$$M_{GL} = R(-\theta)T(-s_x, -s_y)$$

Operação de Pertinência

Para determinar se um ponto P está localizado dentro de um feixe ou não, devemos calcular a projeção de P sobre a reta que contém a seção transversal do feixe. Uma vez expresso no sistema de coordenadas do feixe, para determinar sua projeção, devemos verificar se a semi-reta com origem no ponto (0,0) e que passa por P, cruza a reta $x' = x_p$. Caso essa interseção exista, dizemos que o ponto P é projetável na seção transversal.

Podemos verificar na Figura 4.17 que o único requisito para que um ponto seja projetável é que sua coordenada x' seja maior que zero. Caso isso não ocorra (como acontece para o ponto C), o ponto não está contido no feixe.

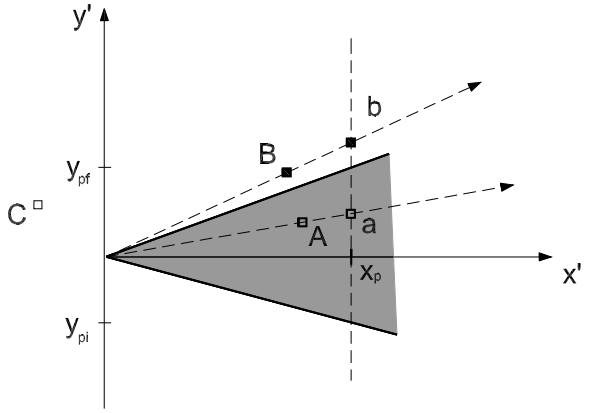


Figura 4.17: Operação de Pertinência

Além de ser projetável, para que um ponto esteja contido dentro de um feixe, a coordenada y' de sua projeção deve estar localizada dentro do intervalo $[y_{pi}, y_{pf}]$, que são os limites da seção transversal. Na figura, podemos ver que o ponto A está contido no feixe, uma vez que sua projeção (ponto a) pertence ao intervalo $[y_{pi}, y_{pf}]$, e o ponto B não, já que sua projeção (ponto b) se encontra fora da seção transversal do feixe.

Note que não é necessário calcular a interseção de duas retas para determinar a projeção de um ponto P sobre a seção transversal. A coordenada x' da projeção já é conhecida a priori (x_p). Assim, para calcular a projeção necessitamos apenas calcular sua coordenada y'. Isso pode ser feito de forma muito eficiente através da semelhança de triângulos. Sendo P'_x e P'_y as coordenadas de P no sistema de coordenadas do feixe, sua projeção sobre a seção transversal (p) será dada por:

$$p_x = x_p$$
$$p_y = \frac{x_p P'_y}{P'_x}$$

Como veremos a seguir, a projeção de uma ponto sobre a seção transversal do feixe é a operação mais utilizada em todo o algoritmo de *beam tracing*, tanto na fase de pré-processamento, quando os feixes são criados, quanto na fase de determinação dos caminhos de propagação. Daí a importância de calcular essa projeção da forma mais eficiente possível.

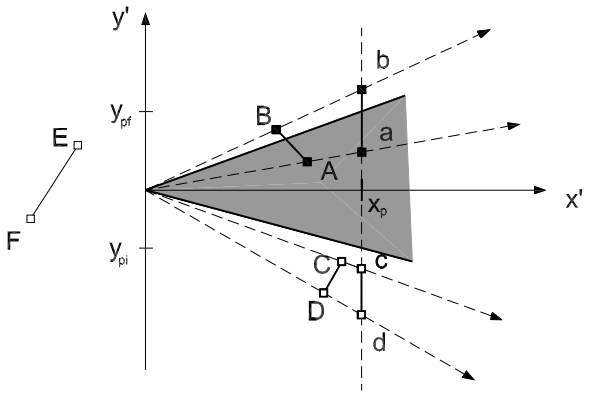


Figura 4.18: Detecção de Interseção entre Oclusores e Feixes

Detecção de Interseção entre Oclusores e Feixes

Assim como foi feito para determinar se um ponto está localizado ou não dentro de um feixe, a detecção de interseção entre feixes e oclusores (representados em 2D por segmentos de reta), é baseada na projeção de pontos na seção transversal de um feixe. Para determinar se um oclusor está contido ou não em um feixe, começamos determinando a projeção dos pontos extremos do oclusor.

Caso nenhum dos pontos extremos do oclusor seja projetável, não existe interseção entre o feixe e o oclusor, pois este se encontra totalmente atrás da fonte do feixe (como o oclusor EF na Figura 4.18).

Caso os dois extremos sejam projetáveis, podemos definir um intervalo sobre a reta que contém a seção transversal utilizando as projeções dos pontos extremos. Caso exista uma superposição entre este intervalo e o intervalo $[y_{pi}, y_{pf}]$, existe uma interseção entre o oclusor e o feixe. Na Figura 4.18, podemos ver que o oclusor AB intercepta o feixe, uma vez que dá origem ao intervalo $[a, b]$, que se sobrepõe à seção transversal do feixe. O mesmo não acontece com o oclusor CD, já que o intervalo $[c, d]$ e a seção transversal são disjuntos.

Quando apenas um dos extremos do oclusor é projetável, teremos um intervalo que parte da projeção que pôde ser calculada e que se estende para $+\infty$ ou para $-\infty$, dependendo da posição em que o oclusor cruza o eixo y'. Caso a interseção aconteça na parte positiva do eixo y', o intervalo se estende para $+\infty$. Se a mesma ocorrer na parte negativa de y', o intervalo se estenderá para $-\infty$. Isso pode ser visto na Figura 4.19, onde podemos ver que o oclusor GH intercepta o feixe, pois gera o intervalo $[g, +\infty]$, que possui uma região em comum com a seção transversal. Já o oclusor IJ, não intercepta o feixe, pois o intervalo por ele definido ($[-\infty, i]$) e a seção

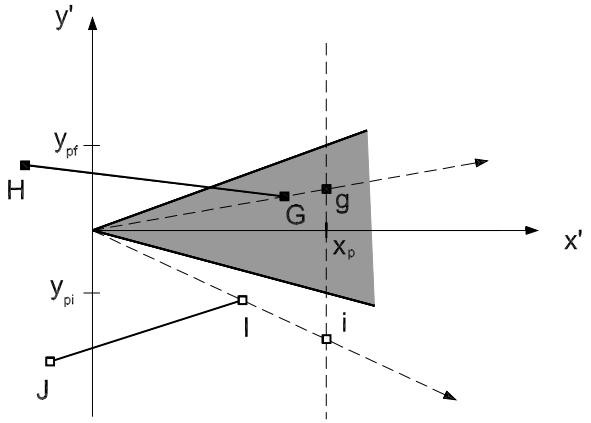


Figura 4.19: Caso Especial de Interseção

transversal são disjuntos.

Como consideramos que os oclusores têm espessura nula, se o oclusor cruzar o eixo y' em sua origem, diremos que a interseção é inexistente.

Cálculo da Interseção com Oclusores

O cálculo da interseção entre feixes e oclusores é necessário para que possamos criar feixes de reflexão e de transmissão, já que a área ocupada por estes tipos de feixes depende desta interseção. Como a Figura 4.20 ilustra, tanto a interseção do feixe com o oclusor AB (segmento AC) quanto a interseção da projeção do oclusor e da seção transversal (intervalo $[a, c]$) delimitam a mesma região do plano (a região delimitada pelas semi-retas SA e SB é mesma delimitada pelas semi-retas Sa e Sb). Assim, podemos trabalhar apenas com os intervalos definidos pela projeção dos oclusores e seções transversais de feixes. Isso facilita, principalmente em três dimensões, a operação de interseção para feixes e oclusores, como veremos adiante.

Quando nos referirmos à interseção entre um feixe e um oclusor, estaremos nos referindo ao intervalo resultante da interseção entre os intervalos da seção transversal e da projeção do oclusor. Assim, na Figura 4.20, a interseção entre o feixe e o oclusor AB será o intervalo $[a, c]$.

Criação de Feixes de Transmissão

Como mencionado anteriormente, sempre que um feixe incide sobre uma aresta transparente do ambiente, devemos criar um novo feixe de transmissão, que são a forma encontrada para permitir que o som se propague de uma região convexa do ambiente para outra.

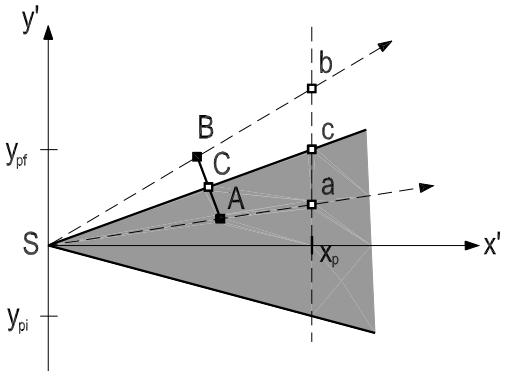


Figura 4.20: Cálculo da Interseção entre Feixes e Oclusores

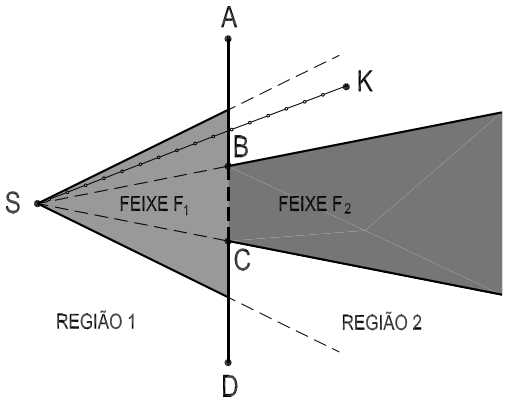


Figura 4.21: Feixes de Transmissão

Como os feixes são modelados como regiões infinitas, a associação de um feixe a uma região convexa é essencial para que os caminhos de propagação sejam calculados corretamente. Esta associação significa que um feixe só pode ser utilizado para criar caminhos de propagação a partir de pontos pertencentes à sua região associada, ou caminhos inválidos serão calculados.

A Figura 4.21 nos permite ilustrar o tipo de erro que aconteceria caso um feixe fosse utilizado com pontos fora de sua região. Imagine que o ponto K fosse usado como o início de um caminho de propagação com o feixe F_1 (presente na região 1). O caminho de propagação calculado (linha com estilo traço-ponto) atravessaria o oclusor AB, o que é claramente incorreto. A única forma de F_1 atingir a região é através da criação de um feixe de transmissão (feixe F_2 na figura), utilizando o segmento transparente BC.

Feixes de transmissão são, basicamente, cópias dos feixes que incidem sobre arestas transparentes, já que não existe uma mudança na direção de propagação dos raios. As diferenças entre um feixe de transmissão e o feixe

que o originou consistem nas regiões convexas associadas a cada feixe e também na seção transversal dos mesmos.

A seção transversal de um feixe de transmissão será sempre a interseção da aresta transparente (por onde ele se propaga) e o feixe incidente. Os dados referentes ao sistema de coordenada do feixe serão os mesmos do feixe incidente.

Criação de Feixes de Reflexão

Ao contrário do que ocorre com os feixes de transmissão, existe uma mudança na direção de propagação dos raios pertencentes ao feixe refletido. Isso significa que o feixe refletido terá um sistema de coordenadas diferente do feixe incidente. Esse sistema de coordenadas pode ser calculado refletindo o sistema do feixe incidente em relação à reta que contém o oclusor. Como a origem do sistema de coordenadas de um feixe corresponde à fonte emissora de raios, sua reflexão corresponde à reflexão utilizada para criação das fontes virtuais de raios refletidos, como é feito na ótica geométrica.

O novo sistema de coordenadas (M_{LGR} e M_{GLR}) pode ser calculado facilmente através das equações abaixo, onde $Ref(a,b,c)$ é a matriz que reflete um ponto em torno da reta $ax + by + c = 0$. Nas fórmulas abaixo, consideramos que a reta cruza o eixo y. Essa consideração aparece nos parâmetros das matrizes de translação, que são utilizadas para levar a reta até a origem do sistema de coordenadas e depois devolvê-la a sua posição original.

$$M_{LGR} = M_{LG}Ref(a,b,c)$$
$$M_{GLR} = M_{GL}Ref(a,b,c)$$

$$Ref(a,b,c) = T(0,-c/b)Ref(a,b)T(0,c/b)$$

$$Ref(a,b) = \begin{bmatrix} -\frac{a^2-b^2}{a^2+b^2} & -\frac{2ab}{a^2+b^2} \\ \\ -\frac{2ab}{a^2+b^2} & \frac{a^2-b^2}{a^2+b^2} \end{bmatrix}$$

Assim como ocorre para os feixes de transmissão, a seção transversal de um feixe refletido será a interseção da seção transversal do feixe incidente com o oclusor. Isto pode ser visto facilmente na Figura 4.22 (à esquerda),

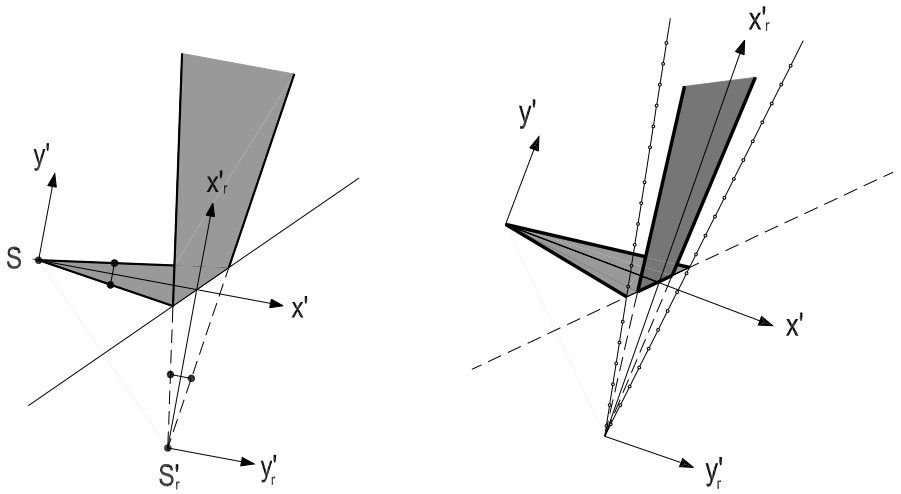


Figura 4.22: Feixes de Reflexão

que ilustra um feixe sendo refletido em um oclusor grande o suficiente para refletir todo o feixe incidente.

A figura ilustra o sistema de coordenadas do feixe incidente (eixos x' e y') e do feixe refletido (eixos x'_r e y'_r). Note que apesar dos sistemas de coordenadas terem sido modificados, as seções transversais permanecem iguais. Dessa forma, podemos pensar na reflexão de um feixe como sendo a transmissão do feixe refletido através do oclusor (Figura 4.22 à direita). Note também que esse feixe refletido será associado à mesma região associada ao feixe incidente.

Também é necessário armazenar em um feixe refletido a equação da reta em que a reflexão ocorreu. Isso é necessário para o cálculo dos caminhos de propagação, pois precisamos saber o ponto da reta em que ocorreu a reflexão.

Criação de Feixes de Difração

Como os feixes traçados em nossa implementação são bidimensionais, consideramos que os feixes de som incidem perpendicularmente sobre as arestas difratoras (que são, na verdade, os vértices de nossos modelos). Dessa forma, ao invés dos cones de difração ilustrados anteriormente, teremos discos de difração.

A única limitação existente na representação projetiva adotada para os feixes consiste na impossibilidade de criar feixes com abertura maior que 180 graus. O espalhamento do som causado pela difração pode facilmente criar feixes com abertura maior que esse limite. Nesse caso podemos criar

mais de um feixe para que, em conjunto, eles possam representar todo o campo de difração. Dessa forma, as operações de interseção e de pertinência podem ser as mesmas utilizadas para os demais tipos de feixes.

Seguindo o exemplo de [55], nossos feixes de difração abrangem apenas a região de sombra definida por uma quina ou por uma parede. Assim, não será necessário criar mais de um feixe para cobrir todo o campo de difração. Precisaremos criar apenas um feixe, já que as regiões de sombra não terão abertura maior que 180 graus.

Para traçar feixes de reflexão e transmissão, necessitamos apenas dos feixes que os gerarão e dos segmentos que determinarão suas formas. Já para a criação de feixes de difração, precisamos também analisar a vizinhança de um vértice difrator para determinar se devemos ou não criar um feixe de difração. Por isso é necessário termos a informação sobre a topologia do modelo.

A Figura 4.23 ilustra três casos em que existem regiões de sombra e um caso em que não existe uma região de sombra. Note que existirá uma região de sombra sempre que as paredes de uma quina estiverem do mesmo lado da fronteira de sombra (reta FS, que passa pela fonte e pelo vértice da quina). No primeiro e no segundo exemplos temos regiões de sombra, já que ambas as paredes da quina de cada exemplo se encontram, respectivamente, à esquerda e à direita da fronteira. No terceiro exemplo, temos uma parede localizada à direita e a outra à esquerda, o que caracteriza a não existência de uma região de sombra. No caso de paredes simples (apenas uma aresta opaca incidindo sobre o vértice difrator), sempre haverá uma região de sombra, como é mostrado no exemplo 4.

Uma vez verificada a existência de uma região de sombra, devemos verificar também se o feixe realmente incide sobre a quina, ou seja, ele deve conter o oclusor (aresta opaca), a aresta transparente e o vértice de difração. Essa condição é satisfeita em todos os casos apresentados na Figura 4.23.

Depois de determinar a existência de uma região de sombra e de garantir que o feixe incide sobre uma quina, podemos criar o feixe difratado. Os feixes de difração criados terão como fonte o vértice difrator (vértices D na figura) e deverão cobrir as regiões de sombra. Nos exemplos ilustrados, vemos que as regiões de sombra são delimitadas pela fronteira de sombra e por uma aresta opaca. Mais especificamente, nos exemplos 1 e 2, os feixes deverão cobrir as regiões delimitadas pelas semi-retas DV_2 e DF. No exemplo 4, a região a ser coberta será delimitada pelas semi-retas DF e DV_1. Os sistemas de coordenadas e as seções transversais desses feixes podem ser calculados da mesma forma como foram calculados os feixes iniciais na

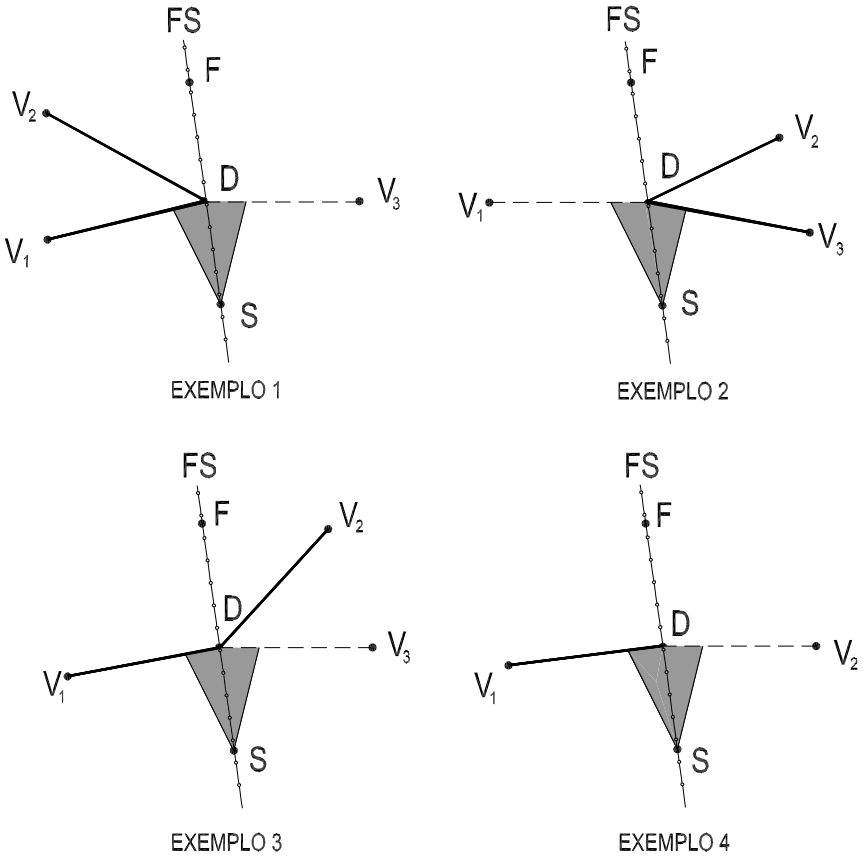


Figura 4.23: Verificando a Existência de Regiões de Sombra

Seção 4.2.1, onde a inclinação e a abertura de um feixe são medidas no sistema de coordenadas global com o auxílio de um raio de referência.

Existe apenas um último detalhe na criação dos feixes de difração. Muitas vezes, a vizinhança de um vértice difrator conterá mais arestas transparentes que as ilustradas na Figura 4.23 (onde o mínimo de arestas necessárias à definição de uma quina foi incluído). A Figura 4.24 ilustra a vizinhança típica de um vértice de difração. Nesta vizinhança, cada par de arestas consecutivas corresponde à fronteira de uma região convexa. Note que o feixe de difração deve cobrir a região delimitada pelas semi-retas DV_3 e DF, ou seja, ele cobrirá três regiões convexas distintas. Como os feixes são utilizados apenas com os pontos de suas região convexas associadas, devemos associar esse novo feixe de difração às três regiões que ele cobre ou criar três cópias do mesmo, associando cada uma a uma região convexa. Em nossa implementação, escolhemos a última alternativa, já que esta facilita o gerenciamento dos feixes.

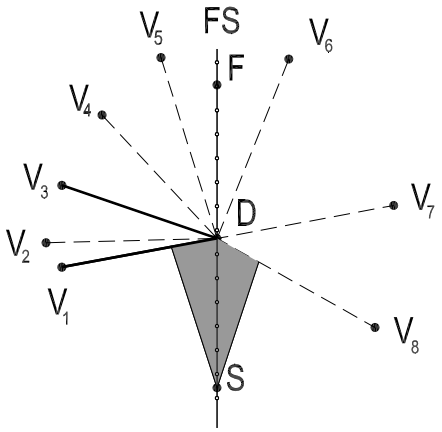


Figura 4.24: Associando Feixes de Difração a Regiões Convexas

Fontes Pontuais Simples

Fontes pontuais simples, que dão origem a ondas esféricas, emitem som em todas as direções ao seu redor. Assim, como não podemos representar feixes com abertura maior que 180 graus, a representação projetiva não é adequada para modelar esse tipo de fonte. Para isso, poderíamos utilizar a mesma técnica sugerida para lidar com o campo completo de difração de uma aresta (criar diversos feixes para cobrir todas as direções necessárias), mas, como veremos a seguir, isso não é necessário.

Ao contrário do que acontece com os demais tipos de feixes, sabemos de antemão o resultado de todas as operações básicas que envolvem fontes pontuais. A operação de pertinência deve sempre retornar uma resposta afirmativa, já que todos os pontos da região convexa associada à fonte estão contidos em seu feixe. Da mesma forma, a operação que verifica a ocorrência de interseções com oclusores também retornará respostas afirmativas e as interseções serão sempre iguais ao oclusor sendo testado.

A particularidade dessas operações implica que não é necessário modelá-las usando a representação projetiva, já que tudo que precisamos saber sobre uma fonte pontual é sua posição. Isso significa que a criação dos feixes de reflexão, transmissão e difração deve ser feita utilizando dados do sistema global de coordenadas (como feito na Seção 4.2.1), já que as fontes pontuais não possuem um sistema de coordenadas local que possa ser herdado pelos feixes criados a partir das mesmas.

Fontes pontuais simples são um excelente exemplo de como a representação utilizada no *beam tracing* é vantajosa em relação à utilizada no *ray tracing*, já que um número infinito de raios seria necessário para cobrir a mesma região coberta por uma fonte pontual simples como a descrita acima.

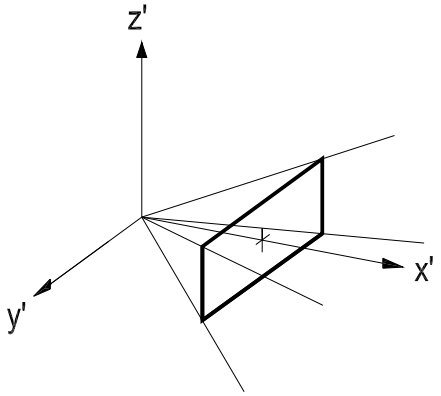


Figura 4.25: Representação Projetiva de um Feixe 3D Piramidal

Feixes em Três Dimensões

Uma das razões para escolher a representação projetiva para os feixes foi o fato de que a mesma pode ser utilizada também em três dimensões. Como os feixes 2D, feixes 3D também podem ser representados por um sistema de coordenadas posicionado sobre sua fonte e por sua seção transversal. As matrizes que representam o sistema de coordenada dos feixes 3D serão as mesmas utilizadas na visualização 3D de objetos para transformar coordenadas do mundo em coordenadas do sistema de uma câmera [28]. A seção transversal do feixe 3D será representada por um polígono localizado sobre um plano perpendicular a um dos eixos dos sistema de coordenadas do feixe, como indica a Figura 4.25.

Para os feixes 3D, a operação de pertinência é muito semelhante à utilizada em 2D. Para os feixes bidimensionais, projetamos o ponto sendo testado na seção transversal e testamos se esta projeção está localizada dentro de um intervalo. Em 3D, deveremos testar se a projeção do ponto está contida ou não dentro do polígono que define a seção transversal.

A operação de interseção entre feixes e oclusores também é muito similar à feita em 2D. Ao invés de determinar a interseção de dois intervalos, devemos determinar a interseção de dois polígonos: o polígono que define a seção transversal e o polígono resultante da projeção do oclusor sobre a seção transversal. Note que, assim como devemos tratar em 2D os casos de intervalos que se estendem a $+\infty$ ou $-\infty$, devemos tratar casos de polígonos que se estendem ao infinito em determinadas direções.

4.3 Traçadores de Feixes

Uma vez definida a estrutura de dados que modela o ambiente e a representação dos feixes de som, passamos a descrição dos algoritmos responsáveis pela construção de uma *beam tree*.

A primeira observação que podemos fazer sobre estes algoritmos, conhecidos como *beam tracers* ou traçadores de feixes, é que eles devem ter um controle preciso sobre a criação de feixes. Aparentemente, bastaria aos traçadores utilizar o predicado de interseção entre feixes e oclusores para que esse controle fosse realizado. Mas, como veremos a seguir, além de serem bastante suscetíveis a erros causados pela imprecisão na representação de números reais, os predicados de interseção entre feixes e oclusores não são suficientes para garantir a correção na criação dos feixes.

Considere o feixe refletido ilustrado na Figura 4.22. Assim como ocorre para o feixe incidente, o predicado de interseção indica que o feixe refletido contém o oclusor, pois não leva a existência deste em consideração. Como este feixe não deve interagir com o oclusor, cabe ao traçador impedir que esta interseção cause a criação de feixes inválidos.

Para impedir esta reflexão é necessário armazenar um registro que indique se um feixe deve ou não interagir com uma aresta. Desta forma, antes de verificar se um feixe e um oclusor interagem, o traçador pode invalidar qualquer interação entre os dois baseado na existência deste registro.

O mesmo problema ocorre com feixes de transmissão, já que estes também contêm as arestas transparentes que os originam.

De forma geral, este registro consiste em um par ordenado contendo os identificadores do feixe e do oclusor em questão. Armazenando o registro no próprio feixe, no momento de sua criação, podemos simplificar o registro para que ele contenha apenas o identificador do oclusor. Contido no feixe, o registro pode ser recuperado eficientemente quando necessário.

4.3.1 Lidando com Erros de Arredondamento

Como mencionamos anteriormente, os predicados de interseção são suscetíveis a erros de arredondamento causados pela representação aproximada de números reais, especialmente em regiões próximas à fronteira de um feixe. Observe a Figura 4.21, que ilustra um feixe de transmissão. Devido a erros de aproximação, interseções entre o feixe F_2 e as arestas AB e CD podem ser acusadas indevidamente.

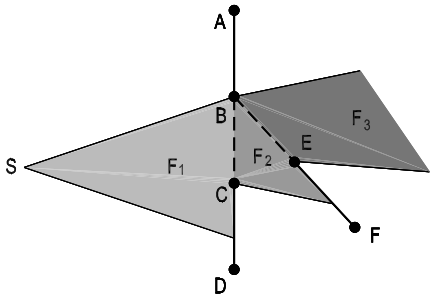


Figura 4.26: Invalidando a Interação entre Feixes de Transmissão e Arestas

Um problema similar ocorre com os feixes de difração que se estendem sobre mais de uma região convexa, como ilustrado na Figura 4.24. As arestas que contêm o vértice difrator não devem interceptar o feixe de difração, pois passam pela origem do mesmo. Mas, novamente devido a erros de arredondamento, interseções inválidas podem ser detectadas e novos feixes criados indevidamente.

Podemos minimizar estes erros de arredondamento utilizando o mesmo registro que invalida a interação entre um feixe refletido e o oclusor que o originou. Para isto basta armazenar nos feixes os identificadores das arestas com as quais podemos garantir a não existência de interseção. Para feixes de difração, estas são as arestas que os limitam (como a aresta DV_3 na Figura 4.24) e as demais arestas contidas em seus interiores. Para os feixes de transmissão, são as arestas da célula adjacentes à aresta transmissora, como as arestas AB e CD na Figura 4.21.

Os feixes de transmissão exigem um cuidado adicional. Sempre que um feixe de transmissão é criado a partir de um outro feixe de transmissão, ele deve herdar os identificadores de arestas do feixe original. Isso é necessário, pois diversas células podem compartilhar um mesmo vértice, como indica a Figura 4.26. De acordo com nossa afirmação anterior, o feixe de transmissão F_2 deve conter os identificadores das arestas AB e CD, pois estas são adjacentes à aresta transparente que o criou. Assim, o feixe F_3 deve conter os seguintes identificadores: AB, CD, EF e BC. Sendo que os dois primeiros foram herdados de F_2 e os dois últimos são os identificadores das arestas adjacentes à aresta transparente, BE, que deu origem a F_3.

Utilizando a herança de identificadores entre transmissões consecutivas, podemos garantir que as restrições corretas a interações estarão presentes nos feixes de transmissão. Além disso, como sempre que ocorre uma reflexão ou difração não existe a herança dos identificadores, o número de identificadores associados a um feixe geralmente será pequeno.

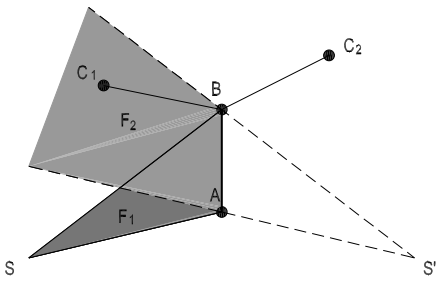


Figura 4.27: Invalidando a Interação entre Feixes de Reflexão e Arestas

Os feixes de reflexão também sofrem de problemas causados por erros de arredondamento. Para estes, entretanto, devemos ter mais cuidado ao invalidar a interação entre um feixe refletido e uma aresta, já que em alguns casos ela pode ser válida. A Figura 4.27 ilustra os casos em que podemos ou não invalidar esta interação. Na figura, um feixe F_1 incide sobre um oclusor AB, dando origem a um feixe refletido, F_2. As arestas BC_1 e BC_2 correspondem, respectivamente, aos casos em que a interação com F_2 é válida e inválida.

No primeiro caso, a interação de BC_1 e F_2 é válida, pois F_2 deve refletir em BC_1, já que esta está realmente contida em seu interior. No segundo caso, entretanto, a interação entre BC_2 e F_2 é inválida, pois BC_2 não está contido em F_2. A identificação destes dois casos pode ser feita com o auxílio da semi-reta $S'B$. Como C_1 está localizado à esquerda da semi-reta, a aresta BC_1 se localiza no interior do feixe e, portanto, sua interação com F_2 deve ser válida. Similarmente, por C_2 se localizar à direita da semi-reta, a interação entre BC_2 e F_2 deve ser inválida.

Infelizmente, a estratégia adotada para analisar a validade da interação entre um feixe refletido e um oclusor também está sujeita a problemas causados por erros de aproximação, que podem ressurgir quando C_1 e C_2 estiverem localizados sobre a semi-reta $S'B$. Esta é claramente uma das ocasiões em que trocamos um problema que ocorre com muita frequência por outro que ocorrerá com menos frequência.

Todas as estratégias apresentadas nesta seção para evitar erros de aproximação são baseadas em critérios locais, pois apenas a adjacência das arestas é levada em consideração. Apesar de existirem casos em que critérios locais são insuficientes para garantir o funcionamento correto do algoritmo frente a erros de aproximação, na prática, o uso destes critérios se mostrou bastante eficaz.

Uma alternativa para o uso destes critérios locais é a utilização de predicados geométricos exatos, como os implementados nas bibliotecas

CGAL [6] e LEDA [8]. Estes predicados, entretanto, não são apropriados para simulações de tempo real, devido ao alto custo computacional envolvido na avaliação de seus predicados.

4.3.2 Traçadores para Pré-Processamento

A construção de uma *beam tree* para fontes fixas geralmente é feita em etapa de pré-processamento, buscando a maior cobertura possível do ambiente, uma vez que a posição do receptor ainda não foi determinada. Esta construção é muito parecida com o algoritmo de busca em largura em grafos [35], pois a *beam tree* é geralmente criada nível a nível. Isto é, todos os nós com altura i são criados antes dos nós com altura $i + 1$.

A Figura 4.28 apresenta o algoritmo mais simples para a construção de uma *beam tree*. No algoritmo, omitimos propositalmente detalhes sobre como identificar o tipo de interação existente entre um feixe e uma aresta, como criar os diferentes tipos de feixes e como validar a interação entre um feixe e uma aresta. Nosso objetivo ao apresentar o algoritmo nesta forma é mostrar como todas as partes previamente explicadas trabalham em conjunto. As operações omitidas foram explicadas com detalhe em seções anteriores. Mais especificamente, nas seções 4.2.1, 4.3 e 4.3.1.

Como a *beam tree* é uma árvore que só pode ser percorrida na direção de sua raiz, precisamos de uma estrutura auxiliar de fila [25] para permitir sua construção.

O primeiro passo na construção de uma *beam tree* consiste na criação do feixe inicial e sua associação a uma das células do ambiente. Em seguida, o feixe inicial, geralmente definido por uma fonte pontual simples (Seção 4.2.1), é inserido na fila auxiliar, para que o laço principal do algoritmo possa começar.

A cada iteração do laço principal, o feixe presente na frente da fila é removido e testado contra as arestas da célula a que está associado. Se a interação entre um feixe e uma aresta for considerada válida, o algoritmo procura determinar o tipo desta interação através das funções *Ocorre_Transmissão*, *Ocorre_Reflexão* e *Ocorre_Difração*, que implementam as condições apresentadas na Seção 4.2.1.

No caso de uma transmissão, um novo feixe de transmissão é criado e sua interação com a aresta que o originou e suas vizinhas, invalidada. O novo feixe é então associado à célula vizinha através da aresta com a qual o feixe original interage. É esta associação com a célula vizinha que permite que o

```
BFS_BeamTracer(Real x, Real y, Int numFeixes)
{
   feixeInicial = Cria_Fonte_Pontual(x, y)
   célulaInicial = Localiza_Célula(x, y)

   Associa_Célula(feixeInicial, célulaInicial)

   Fila F
   Insere_Fim(F, feixeInicial)

   n = 0

   while (Fila_não_Vazia(F) e n < numFeixes)
   {
      feixe = Remove_Primeiro(F)
      célula = Recupera_Célula(feixe)

      for i = 0 to Número_de_Arestas(célula)
      {
         aresta = Recupera_Aresta(célula, i)

         if Interação_Válida(feixe, aresta)
         {
            if Ocorre_Transmissão(feixe, aresta)
            {
               novoFeixe = Cria_Feixe_Transmissão(feixe, aresta)
               novoFeixe.pai = feixe
               célulaVizinha = Recupera_Célula_Vizinha(célula, aresta)
               Invalida_Interação_Transmissão(novoFeixe, aresta)
               Associa_Célula(novoFeixe, célulaVizinha)

               Insere_Fim(F, novoFeixe)
               n = n + 1
            }

            if Ocorre_Reflexão(feixe, aresta)
            {
               novoFeixe = Cria_Feixe_Reflexão(feixe, aresta)
               novoFeixe.pai = feixe
               Invalida_Interação_Reflexão(novoFeixe, aresta)
               Associa_Célula(novoFeixe, célula)

               Insere_Fim(F, novoFeixe)
               n = n + 1
            }

            if Ocorre_Difração(feixe, aresta)
            {
               novoFeixe = Cria_Feixe_Difração(feixe, aresta)
               novoFeixe.pai = feixe
               Invalida_Interação_Difração(novoFeixe)
               Associa_Células_Cobertas_por_Difração(novoFeixe)

               Insere_Fim(F, novoFeixe)
               n = n + 1
            }
         }
      }
   }
}
```

Figura 4.28: Procedimento para Construção de uma *Beam Tree*

som se propague de uma região do ambiente para outra. Após a associação, o feixe é então inserido na fila auxiliar, para que seja processado em outra iteração do laço principal.

O tratamento dos outros dois tipos de interação feixe-aresta é bastante similar ao tratamento dado para a interação que cria feixes de transmissão. Na verdade, podemos dizer que todos estes consistem em etapas de criação de novos feixes, invalidação de interação, associação a células e, finalmente, inserção na fila auxiliar. Existem, entretanto, algumas particularidades em cada caso de interação.

Note que no tratamento de uma reflexão, o novo feixe é associado à mesma célula que o feixe original está associado. Além disso, a operação que invalida as interações destes feixes, *Invalida_Interação_Reflexão*, deve levar em consideração as condições estabelecidas na seção anterior. Da mesma forma, a função *Invalida_Interação_Difração*, deve ser responsável por determinar quais as arestas que devem ser invalidadas para interação com o feixe de difração.

Como mencionamos na Seção 4.2.1, sempre que um feixe de difração se estende sobre mais de uma célula do ambiente, associamos a cada uma destas uma cópia diferente do feixe de difração. No algoritmo apresentado na Figura 4.28, cabe à função *Associa_Células_Cobertas_por_Difração* criar cada uma destas cópias para que elas possam então ser associadas às regiões cobertas pelo feixe de difração original.

A *beam tree*, propriamente dita, é montada a cada iteração do laço principal. Note como é atribuído ao campo *pai* de cada feixe, o feixe que o originou. É interessante notar que, ao contrário do que acontece com diversos algoritmos de árvores, a função que constrói a *beam tree* não retorna o seu nó (ou feixe) raiz. Isso não é necessário, pois a *beam tree* não é percorrida a partir de sua raiz e sim a partir de seus nós intermediários, que ficam armazenados nas diversas células do ambiente. Este armazenamento é realizado pelas funções de associação de feixes a células.

O algoritmo apresentado pára quando não existem feixes para serem processados ou quando o número máximo de feixes a serem criados é alcançado.

Sendo o algoritmo mais simples que permite a construção de uma *beam tree*, é natural que o algoritmo apresentado na Figura 4.28 apresente algum problema. Este consiste em não levar em consideração princípios acústicos. Ele é capaz, por exemplo, de permitir a existência de caminhos na árvore contendo diversas difrações, o que pode causar uma atenuação muito grande em uma onda sonora, tornando-a inaudível.

A audibilidade é o critério de parada mais natural para controlar a criação dos feixes de propagação. Este, entretanto, não pode ser utilizado sem conhecimento prévio dos sinais que serão convoluídos com a resposta ao impulso calculada a partir dos feixes de propagação traçados. Um outro critério possível é estabelecer a atenuação máxima permitida em cada caminho de propagação.

Uma alternativa bastante comum [32] consiste em estabelecer o número máximo de reflexões especulares e difrações que podem existir em um caminho de propagação. O algoritmo apresentado na Figura 4.28 pode ser facilmente modificado para adotar esse critério de parada. Para isto basta adicionar contadores aos feixes que indiquem o número de reflexões e difrações realizados até sua criação. Estes contadores são então incrementados sempre que um feixe de reflexão ou difração é criado.

Além de incluir os contadores nos feixes, precisamos modificar a cláusula que permite o processamento de um feixe para que feixes que tenham atingido os limites estabelecidos não gerem novos feixes.

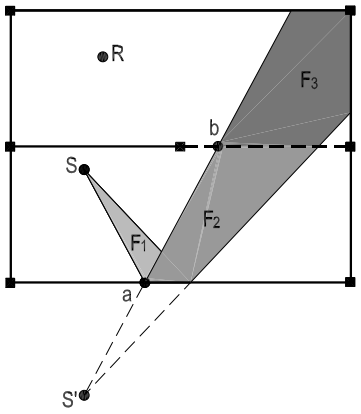
Em nossa implementação foram criados o traçador mais simples (apresentado na Figura 4.28) e o traçador que utiliza um número máximo de reflexões e difrações.

4.3.3 Traçadores com Funções de Prioridade

Como mencionamos anteriormente, uma das modificações realizadas em [36] para permitir o uso do algoritmo de *beam tracing* em tempo real é levar em consideração a posição do receptor ao traçar os feixes de propagação. Isto é feito atribuindo diferentes prioridades a cada feixe, buscando traçar o menor número possível de feixes que levem ao receptor.

O algoritmo apresentado na Figura 4.28 pode ser facilmente modificado para traçar feixes com diferentes prioridades. Basta substituir a fila utilizada no algoritmo por uma fila com prioridades. A atribuição de diferentes prioridades aos feixes é que representa um desafio.

Em [2] diferentes funções de prioridades são apresentadas e comparadas. Entre as funções apresentadas, a que apresenta os melhores resultados busca traçar os caminhos de propagação mais importantes acusticamente, sendo estes os caminhos que percorrem a menor distância entre a fonte e o receptor. Esta função, denominada *potential path length* (PPL) é definida como a soma de duas outras funções de prioridade: *traversed path length* (TPL) e *remaining path length* (RPL).



Feixe	TPL	RPL
F_1	0	0
F_2	$dist(a, S')$	$dist(a, R)$
F_3	$dist(b, S')$	$dist(b, R)$

Figura 4.29: Atribuindo Diferentes Prioridades aos Feixes

TPL é uma estimativa do comprimento do menor caminho de propagação que pode ser definido em uma seqüencia de feixes. Como feixes definem regiões ilimitadas, a estimativa é feita até a última aresta encontrada por uma seqüencia de feixes. Em seqüencias compostas apenas de feixes de reflexão especular e de transmissão, a estimativa TPL será igual à menor distância existente entre a fonte do último feixe na seqüencia e a última aresta. Quando a seqüencia envolve feixes de difração, a estimativa pode ser calculada por partes, calculando a menor distância até o primeiro vértice de difração, depois a distância existente entre o primeiro e o segundo vértice de difração e assim por diante.

RPL consiste em uma estimativa da distância que ainda deve ser percorrida pela onda sonora até que esta alcance o receptor. Esta é calculada como a distância euclideana entre o receptor e um ponto pertencente à última aresta encontrada por uma seqüencia de feixes (o mesmo utilizado no cálculo de TPL). A Figura 4.29 ilustra alguns feixes de propagação e as prioridades associadas a cada um.

As modificações necessárias para traçar feixes com a função de prioridade PPL foram feitas em nossa implementação. A mesma, entretanto, não apresentou bons resultados, requerendo mais testes.

4.4 Cálculo dos Caminhos de Propagação

Como mencionamos na descrição do algoritmo de *beam tracing*, os caminhos de propagação são calculados percorrendo os nós de uma *beam tree* na direção de sua raiz, como foi ilustrado na Figura 2.4 para feixes de

reflexão especular. O algoritmo presente na Figura 4.30 ilustra como é feita a construção dos caminhos de propagação com mais detalhe.

Na figura 4.30, a função *Enumera_Caminhos* cria uma lista com todos os caminhos de propagação que atingem a posição ocupada pelo receptor. Isto é feito localizando a célula que contém o receptor e, em seguida, verificando se cada um dos feixes associados a esta célula contém o mesmo. Quando é determinado que um feixe contém o receptor, um caminho de propagação é criado através da função *Constrói_Caminho*.

Para construir cada caminho de propagação é necessário considerar o tipo de cada feixe encontrado no percurso até a raiz da *beam tree*, pois cada tipo de feixe modifica o caminho de propagação de uma forma diferente.

Como podemos ver na função *Constrói_Caminho*, os feixes de difração e os de fontes pontuais simples adicionam suas próprias fontes à lista de vértices que forma cada caminho. Já um feixe de reflexão adiciona a um caminho de propagação o ponto em que o oclusor a ele associado cruza a reta que passa por sua origem e pelo ponto do caminho previamente calculado, como ilustrado na Figura 2.4.

Outra diferença existente entre os diferentes tipos de feixes é a forma como cada um atenua a onda sonora. Feixes correspondentes a fontes pontuais simples não causam nenhum tipo de atenuação, ao contrário dos feixes de reflexão e difração.

Apesar de depender de fatores como o ângulo de incidência e a frequência da onda, na acústica geométrica é comum modelar a atenuação sofrida por uma onda ao refletir em uma parede como um fator constante [49]. Note que esta simplificação ignora também possíveis mudanças de fase causadas pela reflexão.

Assim, a amplitude de pressão P de uma onda gerada por uma fonte pontual simples, com amplitude inicial P_0, que sofre n_{ref} reflexões e percorre uma distância r até atingir o receptor é dada por

$$P = P_0 \frac{\alpha^{n_{ref}}}{r}$$
$$kr \gg 1$$

Na expressão acima, α é o fator que modela a atenuação causada por uma reflexão. Como em [32], utilizamos um valor igual a 0.8 para este fator.

O fator k presente na restrição acima é o número de onda [49]. Esta restrição garante a validade do decaimento inversamente proporcional à distância, característico do espalhamento esférico de ondas criadas por

```
Enumera_Caminhos(Ponto receptor)
{
   célula = Localiza_Célula(receptor)

   ListaDeCaminhos lc

   for i = 0 to Número_de_Feixes(célula)
   {
      feixe = Recupera_Feixe(célula, i)

      if Dentro_Feixe(feixe, receptor)
         Adiciona_Caminho(lc, Constrói_Caminho(feixe, receptor))
   }

   return lc
}


Constrói_Caminho(Feixe feixeInicial, Ponto receptor)
{
   ListaDeVertices caminho

   Real atenuação = 1.0
   Real comprimento = 0.0

   Adiciona_Vértice(caminho, receptor)

   Ponto p = receptor
   Ponto q

   feixe = feixeInicial

   while feixe != NULL
   {
      if Fonte_Pontual_Simples(feixe)
      {
         q = Posição_da_Fonte(feixe)
         comprimento = comprimento + distância(p, q)
         Adiciona_Vértice(caminho, q)
         p = q
      }
      else if Feixe_Difração(feixe)
      {
         q = Posição_da_Fonte(feixe)
         comprimento = comprimento + distância(p, q)
         atenuação = atenuação * Atenuação_Difração(feixe, p)
         Adiciona_Vértice(caminho, q)
         p = q
      }
      else if Feixe_Reflexão(feixe)
      {
         q = Calcula_Interseção_com_Oclusor(feixe, p)
         comprimento = comprimento + distância(p, q)
         atenuação = atenuação * Atenuação_Reflexão()
         Adiciona_Vértice(Caminho, q)
         p = q
      }

      feixe = feixe.pai
   }

   caminho.atenuação = atenuação
   caminho.comprimento = comprimento
   return caminho
}
```

Figura 4.30: Construção dos Caminhos de Propagação

fontes pontuais simples. Como para ondas com frequência de 1000 Hz, k vale aproximadamente 19 rad/m, utilizamos a expressão acima sempre que r for maior que 1 metro. Para distâncias menores que 1 metro, consideramos que a onda não sofre nenhum tipo de atenuação, ou seja, sua amplitude de pressão continua a ser P_0.

Ao contrário da atenuação causada por uma reflexão, a avaliação da atenuação causada por uma difração pode ser bastante complexa. A amplitude de pressão de uma onda difratada depende de diversos fatores como o tipo de onda incidente (esférica, plana ou cilíndrica), sua freqüência, a distância da fonte sonora e do receptor ao ponto de difração, os ângulos formados por estes com a quina que causa a difração e até mesmo o ângulo entre as paredes que definem a quina. A Figura 4.31 ilustra alguns desses parâmetros.

O maior problema ao avaliar a atenuação causada por uma difração consiste em lidar com as regiões de descontinuidade que ocorrem em soluções como a UTD (*Uniform Geometrical Theory of Diffraction*) [12] ou a DLSM (*Directive Line Source Method*) [52]. Ambas as formulações se tornam descontínuas em regiões próximas à fronteira de sombra (ilustrada na Figura 4.31 como a semi-reta tracejada FS), que delimita a região em que o campo de pressão conta apenas com a contribuição da onda difratada.

Resolvemos então adotar uma aproximação para o cálculo da atenuação causada na difração que busca imitar sua característica mais marcante, que é a crescente atenuação à medida que o receptor se aprofunda na região de sombra. O ângulo existente entre a fronteira de sombra e o receptor (indicado como θ na Figura 4.31) nos permite medir o quão dentro da região de sombra o receptor está localizado. A atenuação implementada é então, uma função deste ângulo.

Incluindo a atenuação causada por uma difração (função δ), a amplitude de pressão de uma onda que sofre n_{ref} reflexões, n_{dif} difrações e percorre uma distância r é dada por

$$P = P_0 \frac{\alpha^{n_{ref}} \prod_{i=1}^{n_{dif}} \delta(\theta_i)}{r}$$
$$kr \gg 1$$

Como δ é utilizada como um fator que modifica a atenuação causada pelo espalhamento esférico, nos valemos dos gráficos presentes em [55] para realizar sua modelagem. Estes gráficos apresentam, para diferentes freqüências, como a razão entre a amplitude de pressão com e sem difração

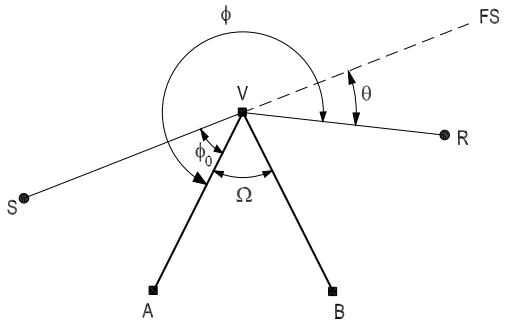


Figura 4.31: Alguns Parâmetros do Cálculo da Atenuação por Difração

varia à medida que o ângulo θ aumenta. Como as mesmas distâncias foram utilizadas para o cálculo da amplitude com e sem difração, esta razão é uma forma de medir a influência de uma difração em função do ângulo θ.

Como as curvas apresentadas em [55] se assemelham a curvas de forma $y = 1/x$, nossa função é definida pela expressão abaixo, que na região da fronteira de sombra tem uma atenuação igual a 1.0, eliminando possíveis problemas de continuidade no campo de pressão. As variáveis K e n foram incluídas para permitir ajustes à nossa função de atenuação. Seus valores, indicados abaixo, foram obtidos a partir de valores extraídos da curva correspondente à freqüência de 1000 Hz.

$$\delta(\theta) = \frac{1}{1 + K\theta^n}$$
$$K = 130$$
$$n = 1.66$$

Além da atenuação da onda sonora, para calcular a resposta do ambiente ao impulso, precisamos calcular o tempo que o caminho de propagação leva para atingir o receptor. O tempo, ou atraso, de um caminho de propagação é calculado como sendo a razão L_{total}/c, onde c corresponde à velocidade de propagação do som (aproximadamente 334.1 m/s [49]) e L_{total} é o comprimento do caminho de propagação.

4.5 Reprodução de Áudio

Em nossa implementação, optamos por utilizar uma biblioteca de áudio já existente, DirectX [9], para realizar a reprodução de áudio. Outras

bibliotecas como Miles [10] ou OpenAL [11] também podem ser utilizadas.

A utilização destas bibliotecas nos dá acesso às funções de transferência (HRTFs) necessárias para o posicionamento de fontes sonoras no espaço e à aceleração por hardware realizada em placas de som. Esta aceleração parece ser essencial para a reprodução de áudio em tempo real, pois em [55] e [32] são utilizados processadores dedicados exclusivamente para realizar a convolução da resposta ao impulso calculada pelos algoritmos de *beam tracing* com um sinal de áudio.

As fontes sonoras implementadas pelas bibliotecas mencionadas acima são utilizadas para representar a contribuição ao campo sonoro de cada caminho de propagação calculado entre a fonte sonora e o receptor. Idealmente, para cada caminho de propagação, uma fonte sonora diferente deveria ser criada. Entretanto, devido ao limitado número de fontes sonoras que podem ser tratadas em hardware e ao alto custo computacional das fontes sonoras tratadas em software, é criado um número limitado de fontes sonoras. Em nossa implementação são criadas, no máximo, 200 fontes sonoras. Sendo que estas correspondem aos caminhos de propagação com maior intensidade sonora.

As fontes são criadas a uma distância igual ao comprimento dos caminhos de propagação que elas representam e com os mesmos ângulos com os quais os caminhos incidem sobre o receptor. Posicionadas desta forma, a atenuação das fontes sonoras devido à distância e o gerenciamento dos diferentes atrasos apresentados pelas fontes são de responsabilidade da biblioteca de áudio sendo utilizada. Cabe a nós, entretanto, incluir a atenuação devido às reflexões e às difrações ocorridas em cada caminho. Isso foi feito alterando a intensidade (volume) de cada fonte sonora criada.

Além de serem capazes de posicionar uma fonte sonora no espaço, estas bibliotecas também possuem implementações de algoritmos de reverberação, que podem ser utilizados, no futuro, para criar as reverberações tardias da resposta ao impulso.