3
Método Proposto

A proposta apresentada neste trabalho foi inspirada nas técnicas apresentadas nos trabalhos de Papaioannou et al. (13), Szirmay-Kalos et al. (16) e Crassin et al. (2). A substituição da geometria por uma discretização em uma grade regular foi baseada no trabalho de Papaioannou et al. (13), no qual eles realizam *ray-marching* em uma representação voxelizada da cena. A técnica de obtenção do volume ocluso em uma esfera tangente através do *depth-buffer*, proposto por Szirmay-Kalos et al. (16), foi modificada para produzir um processo muito eficiente de obtenção do volume ocluso de uma esfera no espaço voxelizado. A divisão do hemisfério em cones e obtenção da oclusão em cada cone apresentadas por Crassin et al. (2) inspiraram o traçado de cones utilizado nesta dissertação.

O método desenvolvido nesta dissertação envolve o lançamento de cones no espaço voxelizado da cena para amostrar a geometria e obter a oclusão de cada *pixel* visível pelo observador. O traçado de cones visa substituir o custoso traçado de raios e faz uso de uma eficiente técnica para amostrar a geometria através de esferas. Pode-se dividir este algoritmo nas seguintes etapas:

– Obtenção dos *pixels* visíveis através de *deferred shading*.

– Voxelização da geometria.

– Resolução da oclusão através do traçado de cones.

3.1
Obtenção dos pixels visíveis

O cálculo da oclusão somente é resolvido para os *pixels* visíveis pelo observador, consequentemente é necessário construir *buffers* contendo as informações geométricas necessárias para esse cálculo para cada um desses *pixels*. Em um processo semelhante à *deferred shading* a cena é enviada ao *pipeline* gráfico onde é transformada pela matriz de projeção corrente, rasterizada, e cada fragmento gerado executa um *shader* que direciona informações para *G-Buffers* (*buffers* de geometria).

Os *G-Buffers* contêm a normal e a posição no espaço do olho de cada ponto visível. Esta informação é recolhida diretamente da geometria e é aplicada na determinação do hemisfério de influência do *pixel* e na verificação da vizinhança.

PUC-Rio - Certificação Digital Nº 1012631/CA

3.2 Voxelização

O método proposto nesta dissertação faz uso de uma estrutura que armazena a geometria da cena de uma forma facilmente acessível durante o cálculo da oclusão. Essa estrutura é uma grade regular e é construída a cada quadro utilizando o método proposto por Eisemann e Décoret (6) (detalhado na Seção 2.2).

O processo de voxelização ocorre através da execução de um *shader* que produz uma máscara contendo todos os *bits* a frente do k-ésimo *bit* ativos, e utiliza a operação lógica XOR para construir a borda e o interior dos objetos da cena, tal como feito por Eisemann e Décoret (6).

Com a utilização de uma textura 1D como *lookup table* pode-se facilmente determinar a máscara de *bits* que cada fragmento deve gerar. Esta textura possui 128 entradas, uma pra cada *voxel* na direção z, e utiliza o índice da grade na dimensão z para o acesso. A Equação 3-1 converte a distância do fragmento ($z_{distance}$) até o *near plane* no índice na dimensão z da grade.

$$z_{GridIndex} = \frac{z_{distance} - z_{near}}{z_{far} - z_{near}} \tag{3-1}$$

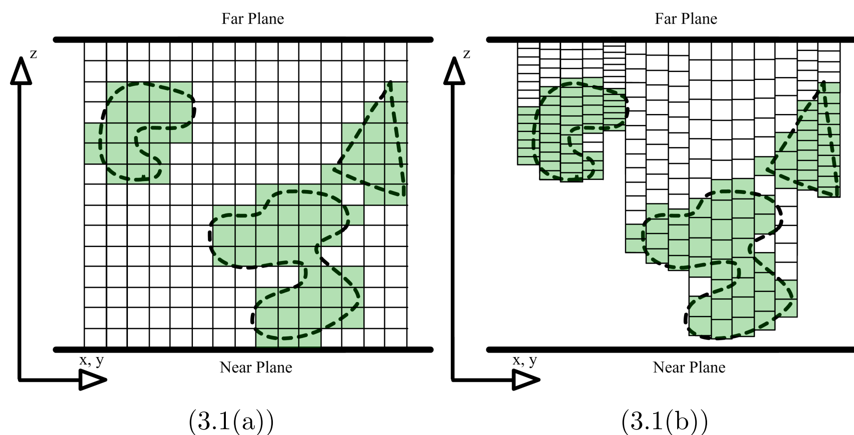
Levando em conta informações presentes nos *G-Buffers*, geradas na etapa anterior, pode-se realizar uma pequena modificação no cálculo do valor que determina o índice da grade. Utilizando a distância no espaço do olho do fragmento mais próximo da tela ($z_{nearest}$) ao invés do *near plane* cria-se uma grade contendo informações mais relevantes na região mais próxima ao observador. Essa otimização é baseada no conceito de *local slicemap* apresentado por Eisemann et al. (5). A nova equação é apresentada em 3-2. A Figura 3.1 apresenta a diferença entre uma grade que usa o *near plane* e outra com esta otimização.

$$z_{GridIndex} = \frac{z_{distance} - z_{nearest}}{z_{far} - z_{nearest}} \tag{3-2}$$

Percebe-se que a mesma abordagem poderia ser utilizada para substituir o *far plane* pela distância no espaço do olho do fragmento mais afastado. Entretanto, a obtenção dessa informação envolveria um novo envio da geometria. A informação do fragmento mais próximo é obtida sem custo, pois ela já foi calculada na etapa anterior.

A construção da grade necessita que a geometria seja enviada para o *pipeline* gráfico, contudo, deve-se garantir que todos os fragmentos cheguem a executar o *fragment shader*. Para isso são desativados o teste de z e o *culling* da geometria.

PUC-Rio - Certificação Digital Nº 1012631/CA



(3.1(a)) (3.1(b))

Figura 3.1: Comparação entre uma grade regular, (a), e uma grade compactada, (b), com a otimização utilizando o fragmento mais próximo da tela.

Os planos *near* e *far* realizam o *clipping* dos polígonos da geometria e evitam a geração de alguns fragmentos. A função *depth clamp*, disponibilizada pelo *hardware*, é utilizada para realizar o truncamento do valor da profundidade entre os planos *near* e *far*, fazendo com que todos os fragmentos possuam uma profundidade entre $[0, 1]$, garantindo que todos os fragmentos sejam gerados.

PUC-Rio - Certificação Digital Nº 1012631/CA

3.3 Traçado de cone

A abordagem de resolução da integral da oclusão ambiente proposta nesta dissertação envolve a substituição do traçado de raios pelo traçado de cones. Para tanto, o hemisfério de influência de um determinado ponto deve ser dividido em diversos cones. A Figura 3.2 ilustra esta divisão. Assim que cada cone obtém a oclusão da geometria na sua direção os resultados são somados e normalizados pelo número de cones no qual o hemisfério foi dividido.

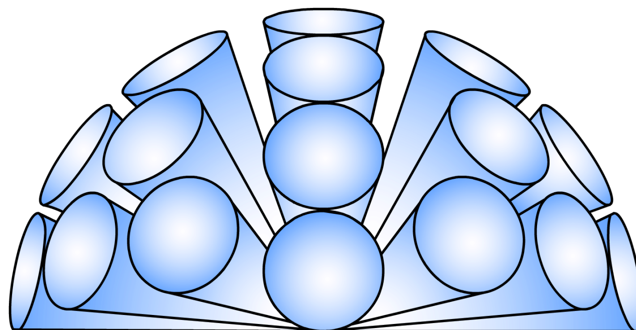


Figura 3.2: Representação da divisão do hemisfério do ponto P em diversos cones.

Cada cone tem seu vértice no ponto que se objetiva estimar a oclusão, seu eixo é alinhado com uma das direções obtidas na divisão do hemisfério e

sua altura equivale ao raio do hemisfério de influência ($Rmax$). Ao longo do eixo do cone são dispostas esferas, cada uma com um raio proporcional ao raio da seção do cone correspondente. A Figura 3.3 ilustra esta disposição.

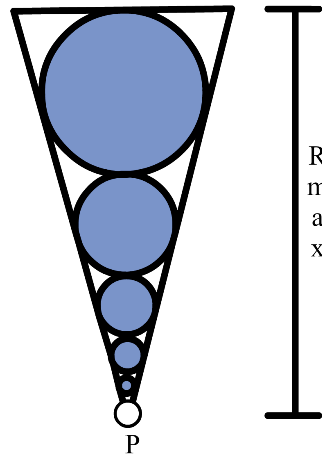


Figura 3.3: Ilustração de um cone composto por esferas. $Rmax$ equivale ao raio de influência do hemisfério de P.

A obtenção do volume ocluso da esfera é feito de forma semelhante ao método proposto por Szirmay-Kalos et al. (16) (detalhado na Subseção 2.1.2). Contudo, ao invés de utilizar o *depth-buffer* para obter o volume da esfera a técnica foi modificada para obter o volume em uma grade regular. Detalhes sobre o funcionamento desta técnica são apresentados na Subseção 3.3.1.

Após cada esfera ter amostrado a geometria ao longo do eixo do cone a contribuição de cada esfera é acumulada para obter a oclusão total do cone. Este processo é expandido na Subseção 3.3.2.

A ideia geral do funcionamento total do algoritmo de traçado de cones é esclarecida na Subseção 3.3.3.

3.3.1 Oclusão da esfera

Como a geometria é simplificada por uma versão discretizada em uma grade regular é necessário um modo eficiente de acessar esta estrutura e avaliar a geometria. A técnica proposta para realizar este acesso consiste na disposição de esferas no espaço da grade e obter a porção do volume de cada esfera que contém *voxels* ativos.

O volume da esfera é aproximado através de uma integral de Monte Carlo onde amostras são distribuídas em um disco de mesmo centro da esfera e cada amostra nesse disco é utilizada para definir uma coluna da esfera. O somatório dessas colunas resulta no volume da esfera.

PUC-Rio - Certificação Digital Nº 1012631/CA

O disco com as amostras é perpendicular ao eixo z da grade, consequentemente, todas as colunas são paralelas a este eixo e estão contidas em um *texel* da textura da grade. Desta forma somente um acesso a textura é necessário para obter informações sobre, até, 128 *voxels*.

A disposição das amostras neste disco utiliza a distribuição de disco de Poisson. Essa distribuição é uma generalização da distribuição de Poisson, mas garante que cada amostra satisfaça uma restrição de distância mínima. Ela é obtida gerando pontos através da distribuição de Poisson regular, mas somente retendo os pontos que satisfazem a restrição. Este processo é bastante custoso, mas é executado em pré-processamento e produz amostras de maior qualidade.

Um único acesso a textura da grade retorna 128 *voxels*, contudo somente os *voxels* que se encontram dentro da esfera devem ser levados em conta. Assim, necessita-se encontrar os pontos na grade que restringem a faixa de *voxels* a serem considerados. A esfera se encontra no espaço do olho, e as amostras estão no espaço 2D local do disco. As amostras são convertidas para o espaço do olho de acordo com o raio da esfera e transladadas para o centro da esfera. Desta forma têm-se a esfera de centro C^{eye} $(c_x^{eye}, c_y^{eye}, c_z^{eye})$, raio r e as amostras S^{eye} $(x^{eye}, y^{eye}, c_z^{eye})$ e deve-se obter os pontos onde o eixo da coluna intercepta a esfera, o ponto de entrada na esfera é chamado de Sin^{eye} e o de saída de $Sout^{eye}$. Ambos os pontos possuem as mesmas coordenadas x e y da amostra devido a coluna ser paralela ao eixo z. Já a posição em z é determinada pelas seguintes equações:

$$Zin^{eye} = c_z^{eye} - \sqrt{1 - (x^{eye})^2 - (y^{eye})^2}$$

$$Zout^{eye} = c_z^{eye} + \sqrt{1 - (x^{eye})^2 - (y^{eye})^2}$$

Os pontos Sin^{eye} $(x^{eye}, y^{eye}, Zin^{eye})$ e $Sout^{eye}$ $(x^{eye}, y^{eye}, Zout^{eye})$ estão no espaço do olho e são convertidos para o espaço da grade. A conversão é feita em duas partes, primeiramente os valores x^{eye} e y^{eye} são convertidos para o espaço de tela (x^{screen}, y^{screen}) através da matriz de projeção. Como a grade foi construída alinhada com a tela os índices dos *pixels* da tela e dos *texels* da textura da grade são equivalentes. Os valores Zin^{eye} e $Zout^{eye}$ determinam qual a faixa de *voxels* deve ser acessada. Para obter o índice do eixo z da grade é utilizada a Equação 3-2 e seu resultado é multiplicado por 128 (número total de *voxels*). Dessa forma pode-se acessar a faixa de *bits* equivalente a estes *voxels* e contar o número de *voxels* com presença de geometria. A Figura 3.4(a) apresenta a disposição dos pontos Zin^{grid} e $Zout^{grid}$ para algumas amostras,

PUC-Rio - Certificação Digital Nº 1012631/CA

a Figura 3.4(b) apresenta a faixa de *voxels* obtida para alguns pontos.

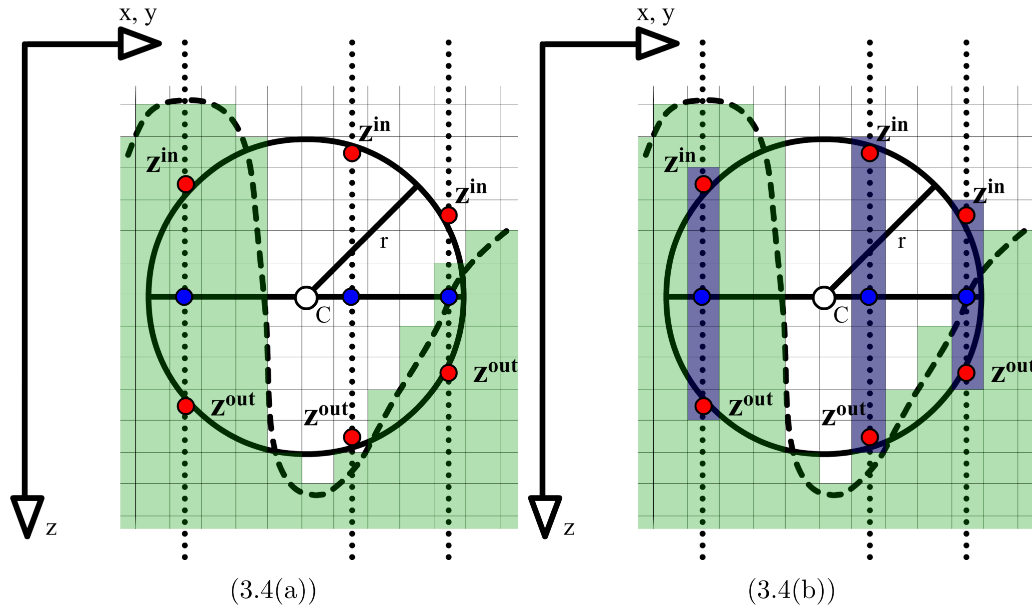


Figura 3.4: Ilustração de uma esfera no espaço voxelizado contendo amostras no disco unitário (pontos azuis), juntamente com os pontos Zin^{grid} e $Zout^{grid}$. Em (b) pode-se observar a faixa de *voxels* resultante (*voxels* azuis).

Assim que os pontos no espaço da grade foram obtidos e a textura da grade foi lida, o número de *voxels* ativos deve ser contado. O processo de contagem envolve determinar quantos *bits* na faixa entre Zin^{grid} e $Zout^{grid}$ são 1. O processo de contagem é realizado através de duas etapas, primeiramente os *bits* desnecessários (fora da faixa Zin^{grid} e $Zout^{grid}$) são convertidos para 0. Após a eliminação dos *bits* a faixa restante é utilizada para indexar uma tabela que retorna o número de *bits* ativos.

Como o valor lido da textura é um *texel* composto por quatro canais de cor (RGBA) o processo de eliminação dos *bits* desnecessários deve ser feito em mais de um passo. A partir da faixa monta-se uma máscara para cada canal de cor e aplica-se usando a operação binária E em cada canal. A Figura 3.5 ilustra esta etapa.

A contagem de *bits* ativos da faixa de interesse é realizada de forma semelhante. Para cada canal são feitos até dois acesso a uma textura de contagem, a qual irá determinar o número de *bits* ativos. A textura de contagem funciona como uma tabela que recebe um valor inteiro como índice e retorna o número de *bits* ativos nesse índice. Se a tabela permitisse o acesso de qualquer combinação de 32 *bits*, ela teria que comportar 2^{32} entradas. Mesmo usando a menor quantidade de espaço disponível, ou seja, um *byte* para cada entrada, a tabela ainda usaria $4Gb$ de memória. Utiliza-se, então, uma tabela que permite o acesso de qualquer combinação de 16 *bits*, requerendo apenas $64Kb$

PUC-Rio - Certificação Digital Nº 1012631/CA

de memória. Assim, são necessários até dois acessos a textura para descobrir quantos *bits* ativos existem em uma variável de 32 *bits*. A Figura 3.6 clarifica o processo de contagem.

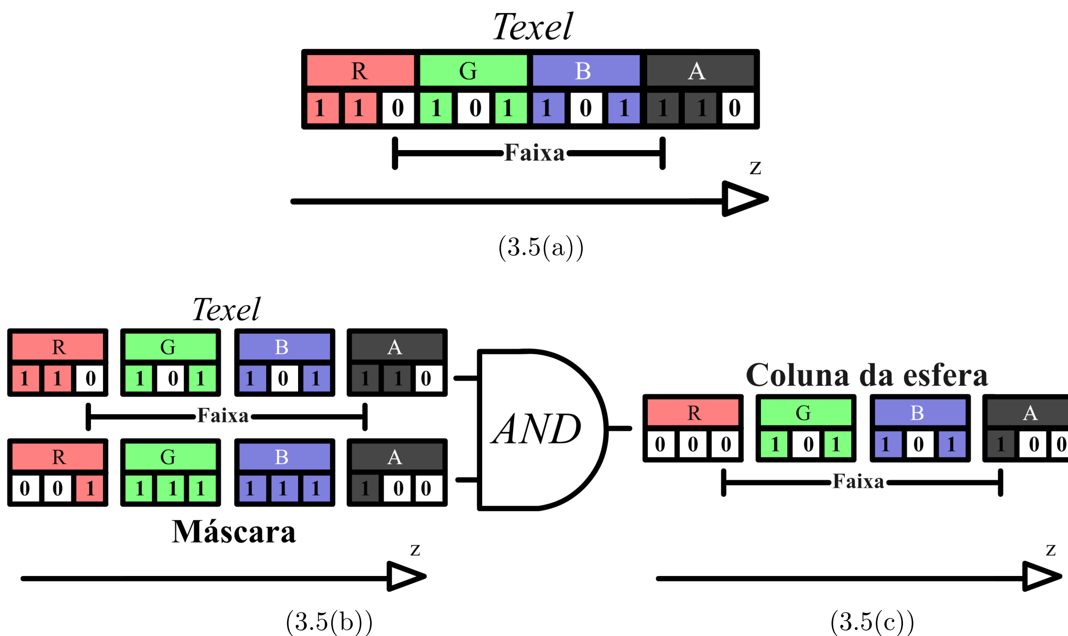


Figura 3.5: Ilustração da aplicação de máscara de *bits* em uma coluna da grade. Para simplificar foi considerada uma textura com apenas 3 *bits* por canal.

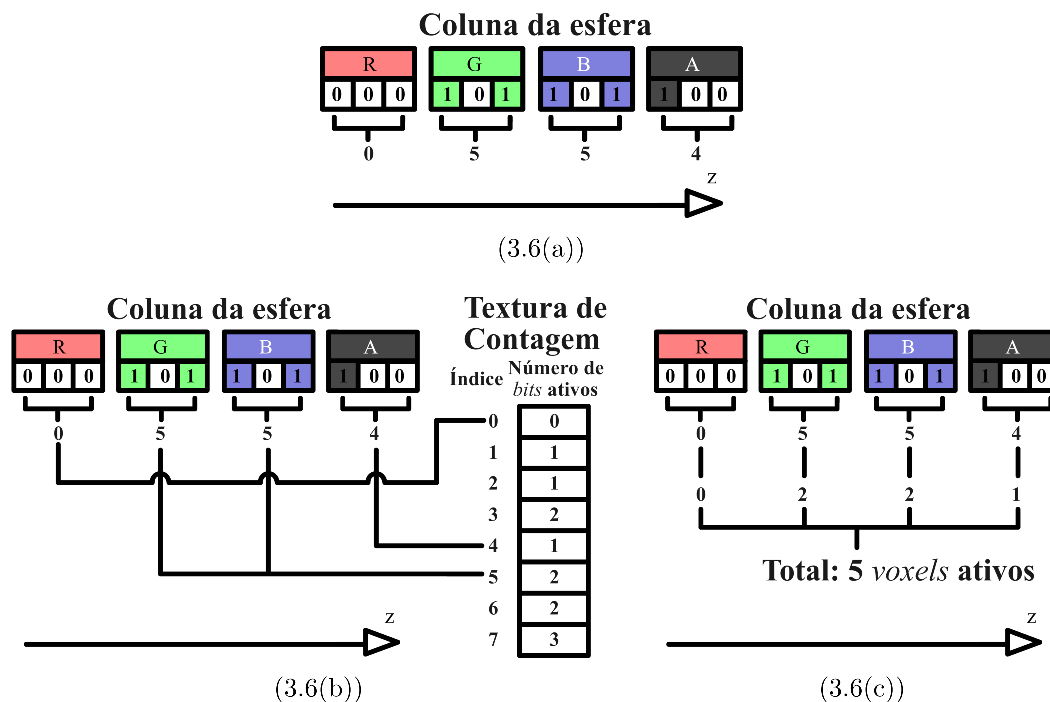


Figura 3.6: Processo de contagem de *bits* ativos em um *texel*. Assume-se que o *texel* já sofreu o descarte dos *bits* fora da faixa. Para simplificar foi considerada uma textura com apenas 3 *bits* por canal. Em uma textura com 32 *bits* são feitos dois acessos a textura de contagem por canal.

PUC-Rio - Certificação Digital Nº 1012631/CA

Após o processo de contagem de cada canal os valores são somados e obtém-se o número total de *voxels* nesta faixa. Então, o número de *voxels* ativos é utilizado para determinar qual o tamanho da faixa ocupada pela geometria no espaço do olho. Como o tamanho dos *voxels* é constante para uma mesma coluna da grade basta multiplicar o número de *voxels* ativos pelo tamanho de um *voxel*. Através de Zin^{eye} e $Zout^{eye}$ no espaço do olho temos o comprimento total da coluna da esfera. Aplicam-se os valores de todas as colunas da esfera na Equação 2-6, e obtém-se o volume ocluso da esfera.

3.3.2
Contribuição da esfera

O resultado da oclusão de cada esfera deve ser acumulado para contabilizar a oclusão final do cone. Baseado no fato de que cada esfera resulta o quanto de seu volume está ocupado pela geometria e, assumindo que este resultado equivale à razão do número de raios emitidos a partir do ponto inicial que atingem alguma geometria, o modelo de absorção *front-to-back* (8), é utilizado para acumular os resultados de cada esfera.

A Figura 3.7 apresenta o cálculo da oclusão de um cone utilizando os valores de oclusão de cada esfera. Nota-se que no exemplo da figura existem três esferas que definem o cone partindo de P. Cada esfera apresenta à sua esquerda o quanto de seu volume total está ocupado pela geometria. Considerando que são lançados x raios a partir do ponto P, se a primeira esfera (mais próxima de P) bloqueia 30% dos raios, então a segunda esfera será atingida por $x*(1-0{,}3)$ raios. A segunda esfera bloqueia 25% dos raios, consequentemente, o número de raios que a atravessará é $x * 0{,}7 * (1 - 0{,}25)$. A última esfera bloqueia 65% dos raios, o que resulta em $x * 0{,}7 * 0{,}75 * (1 - 0{,}65)$ raios que não atingiram a geometria. A oclusão total desse cone equivale ao número de raios que atingiram a geometria, ou seja, $1 - (0{,}7 * 0{,}75 * 0{,}35) = 81\%$ de oclusão.

PUC-Rio - Certificação Digital Nº 1012631/CA

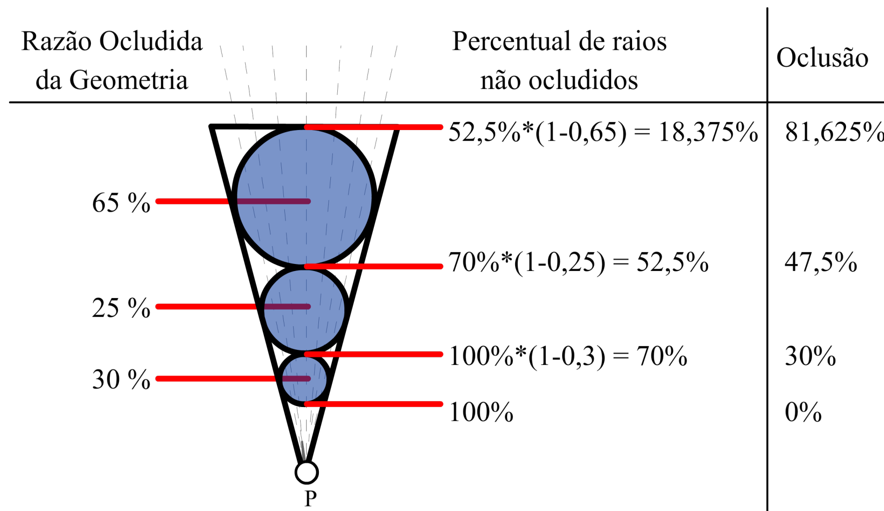


Figura 3.7: Esquema apresentando um cone com a oclusão obtida em cada esfera. A coluna do meio indica a porção de raios não ocludidos antes e depois de cada esfera. A oclusão ao longo do eixo é exibida na coluna da direita.

PUC-Rio - Certificação Digital Nº 1012631/CA

3.3.3 Algoritmo

Essencialmente o algoritmo de traçado de cones é composto pela divisão do hemisfério em cones, pela divisão dos cones em esferas e pelo cálculo do volume ocluso das esferas através de amostras. Cada um desses passos interage para que a oclusão total de um ponto seja estimada por meio de um *fragment shader*, o qual é executado para todos os *pixels* visíveis.

Para que este método possa ser configurado para a execução mais adequada para cada aplicação algumas variáveis podem ser alteradas para modificar a relação entre qualidade e desempenho. Estas variáveis são:

- Número de cones
- Número de esferas por cone
- Número de amostras por esferas

A divisão do hemisfério em cones é controlada pela variável *número de cones* e determina as direções em torno da normal do ponto onde serão posicionados os cones. O vértice de todos os cones é o ponto que se quer calcular, as direções são geradas em pré-processamento e lidas em tempo de execução através de uma textura. Como estas direções devem estar dentro do

hemisfério, cada direção lida é transformada por uma matriz de rotação que leva o vetor direção lido para a direção equivalente no hemisfério deste ponto.

Assim que a direção do cone é obtida as esferas que o compõe são posicionadas no espaço do olho. A quantidade de esferas é determinada pela variável *número de esferas por cone*. Cada esfera possui um raio e uma distância do vértice do cone, estas informações são criadas em pré-processamento e acessadas em tempo de execução através de uma textura. A distância do vértice do cone também é utilizada como parâmetro em uma função de atenuação, a qual dá pesos menores para esferas mais distantes.

O valor da oclusão de cada esfera depende da variável *número de amostras por esfera* e teve seu cálculo detalhado anteriormente na Subseção 3.3.1. Após a obtenção da oclusão em cada esfera calcula-se o valor da atenuação dessa esfera e a contribuição de cada esfera é estimada. A contribuição de cada esfera e o processo de acumular o valor da oclusão no cone foi apresentado na Subseção 3.3.2. Assim que cada cone avalia a sua oclusão esta é acumulada e posteriormente a oclusão total do ponto é normalizada pelo número de cones.

Este algoritmo depende da leitura de diversos valores de textura, os quais determinam fatores importantes tais como direção de cada cone, posicionamento de cada esfera e localização de cada amostra em cada esfera. Se os mesmos valores forem utilizados em todos os *pixels* nota-se o surgimento de padrões visualmente perceptíveis. Para reduzir este efeito adiciona-se *jitter* nas informações de amostragem. *Jitter* consiste em perturbar a periodicidade de um sinal, neste caso, perturbam-se os valores utilizados como direção, posicionamento e amostras. Quando os dados são gerados, ao invés da textura conter somente uma distribuição de dados, ela contém diversas distribuições diferentes. Em tempo de execução cada fragmento faz uma rápida geração de um número pseudoaleatório usando a sua posição no espaço da tela como semente para determinar qual a distribuição será utilizada, eliminando assim o efeito da periodicidade.

A Tabela 3.1 apresenta o funcionamento do algoritmo em pseudocódigo.

PUC-Rio - Certificação Digital Nº 1012631/CA

Tabela 3.1: Pseudocódigo do funcionamento do algoritmo.

$normal, eyeposition, depth \leftarrow readInputData()$
if $depth < 0.0$ **then**
  $discard$
end if
$rotMatrix \leftarrow calcHemisphereRotationMatrix(normal)$
$ao \leftarrow 0$
for $i = 0, numCones$ **do**
  $coneDir \leftarrow readConeDir(i)$
  $coneAo = 0$
  for $j = 0, numSpheresByCone$ **do**
    $distance \leftarrow readSphereDistance(j)$
    $radius \leftarrow readSphereRadius(j)$
    $center \leftarrow calcSphereCenter(distance, eyeposition)$
    $sphereAO \leftarrow calcSphereAO(j, center, radius, rmax)$
    $att \leftarrow calcAttenuation(distance)$
    $coneAO \leftarrow calcSphereContribution(sphereAO, att, coneAO)$
  end for
  $ao += coneAO$
end for
$finalAo \leftarrow normalize(ao, numCones)$

PUC-Rio - Certificação Digital Nº 1012631/CA