2
Trabalhos Relacionados

Diversas soluções foram propostas visando solucionar a oclusão ambiente de forma eficiente. As principais abordagens que se relacionam com este trabalho são apresentadas a seguir. Este capítulo se divide nas seguintes seções: A Seção 2.1 detalha dois algoritmos que calculam a oclusão em espaço de tela. A Seção 2.2 demonstra o funcionamento da técnica usada para fazer a voxelização da cena, a qual é utilizada em um dos métodos na Seção 2.3 e na solução proposta nesta dissertação. A Seção 2.3 dispõe duas abordagens que estimam a oclusão ambiente utilizando informações volumétricas da geometria. Considerações sobre cada um desses trabalhos são feitas na Seção 2.4.

2.1
Oclusão ambiente em espaço de tela

As técnicas mais utilizadas em aplicações em tempo real são compostas por métodos que fazem uso do espaço da tela para aproximar a integral da oclusão (9, 15, 4, 1, 16). Utilizam-se apenas as informações presentes no *depth-buffer* ou em *buffers* construídos através da utilização de *deferred shading* (3). A grande vantagem desta abordagem é a independência da complexidade da geometria e o tratamento automático de cenas dinâmicas. Contudo, ao levar em conta somente a informação presente na tela, a oclusão ambiente se torna dependente da posição do observador, o que pode levar a uma oclusão ambiente incorreta devido à informação limitada (13).

2.1.1
Aproximação da geometria por esferas

Shanmugam e Arikan (15) propuseram um algoritmo que aproxima o efeito de oclusão ambiente em tempo real. Esse trabalho é baseado na aproximação da geometria por uma muito menos detalhada. A oclusão é computada ao calcular para cada *pixel* o quanto seus *pixels* vizinhos causam de oclusão e normalizar o resultado.

A geometria é submetida para o *pipeline* e os atributos de cada *pixel* são armazenados em memória de vídeo no chamado *ND-Buffer*. Esse *buffer* é composto pela normal e profundidade de cada *pixel* e é utilizado na passada subsequente para calcular a oclusão ambiente.

O cálculo da oclusão é realizado para cada *pixel* presente no *ND-Buffer* e consiste em amostrar alguns *pixels* vizinhos e reconstruir a superfície que ele

representa por uma esfera. Através da profundidade de cada amostra obtém-se a posição do centro e o raio de uma esfera no espaço do olho que é projetada para aproximadamente um *pixel*.

A partir desta esfera calcula-se o valor da oclusão causada por ela através da área da calota esférica. A Figura 2.1 apresenta a oclusão da esfera Q de centro C e raio r no ponto P de normal $\hat{n}$. A Equação 2-1 formula este conceito.

$$A'(C, r, P, \hat{n}) = S_\Omega(P, C, r)(\hat{n} \cdot \hat{PC}) \tag{2-1}$$

onde $S_\Omega(P, C, r)$ é a área da calota esférica gerada pela esfera Q no hemisfério unitário de P, Equação 2-2. $\hat{PC}$ é o vetor unitário entre P e C.

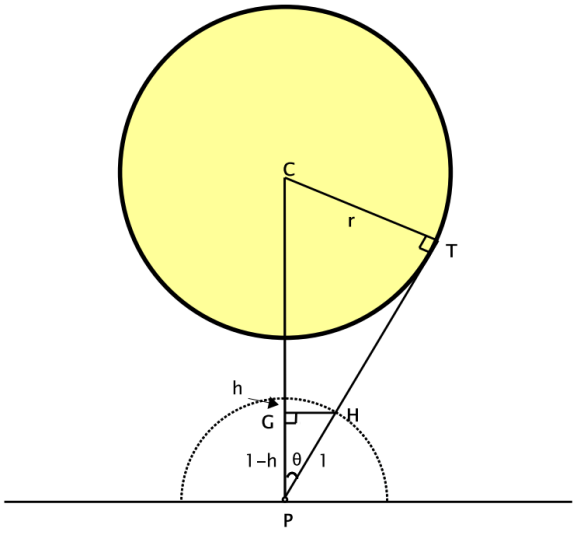


Figura 2.1: A oclusão do hemisfério de P causada por uma esfera. h define a calota esférica obtida pela projeção da esfera. Imagem retirada de (15).

$$S_\Omega(P, C, r) = 2\pi\left(1 - \cos\left(\sin^{-1}\left(\frac{r}{\left|\overrightarrow{PC}\right|}\right)\right)\right) \tag{2-2}$$

Com a projeção do espaço do mundo para o plano da tela, oclusores vizinhos no espaço do mundo também estão próximos no espaço projetado. Ou seja, a vizinhança de um ponto no espaço projetado fornece uma boa ideia da vizinhança no espaço do mundo. Contudo, pontos que estão próximos no espaço projetado não necessariamente são vizinhos no espaço do mundo. A influência de pontos distantes é minimizada por uma função de atenuação parametrizada pela distância.

Para melhorar a qualidade do resultado, é necessário considerar também a contribuição de pontos oclusores não visíveis. Para tanto, são realizadas duas passadas do método *depth peeling* (7) para preencher *ND-Buffers* de duas camadas da geometria.

2.1.2
Integral volumétrica e esfera tangente

A resolução da integral direcional da oclusão ambiente, Equação 1-1, requer que sejam traçados raios em diversas direções, o que é bastante custoso. A abordagem proposta por Szirmay-Kalos et al. (16) transforma a integral direcional em uma integral volumétrica facilmente resolvida na GPU.

Ao invés de lançar raios para amostrar a visibilidade de cada ponto o método proposto substitui o traçado de raios pela verificação se um determinado ponto é interno ou externo. Além disso, o espaço de integração é reduzido para eliminar as direções de ângulos altos com a normal, as quais contribuem pouco para a oclusão. Assim, Szirmay-Kalos et al. definem a integral volumétrica como:

$$V(P) = \frac{\int_S \tau(P) dP}{|S|} \tag{2-3}$$

onde $|S|$ representa o volume total da esfera tangente.

Esta equação representa o volume ocludido da esfera tangente à superfície. A Figura 2.2 permite ver que ao invés de lançar diversos raios pelo hemisfério (raio r) de P apenas a esfera tangente (raio $\frac{r}{2}$) é considerada.

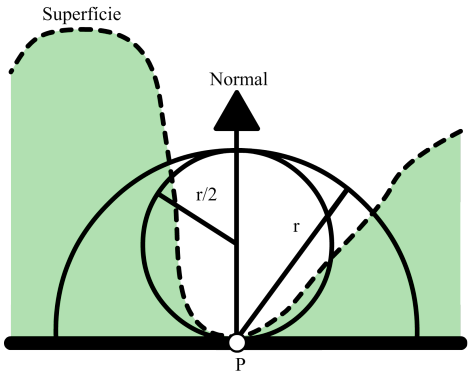
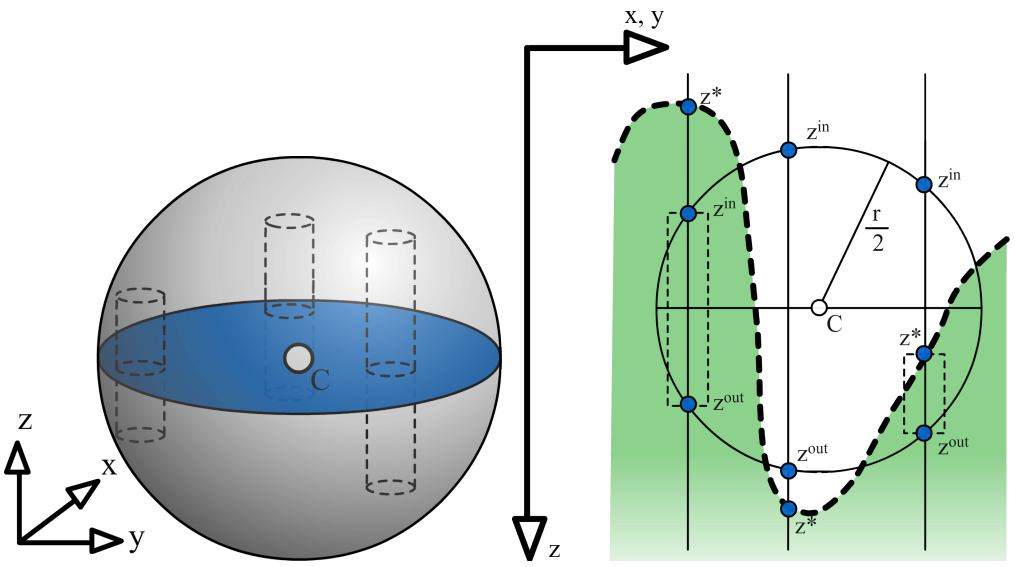


Figura 2.2: Espaço de integração; o semi círculo indica o hemisfério em torno de P; a esfera de raio $\frac{r}{2}$ é o novo espaço de integração.

A função $\tau(P)$ equivale ao teste que determina se P é interno ou externo. Define-se que o espaço que contém o observador é externo e outros espaços delimitados por uma superfície de separação são internos. Nesse trabalho o *depth-buffer* é utilizado como superfície de separação, ou seja, qualquer ponto que possuir um valor de z maior que o presente no *depth-buffer* é considerado interno. Consequentemente a verificação se um ponto é externo é realizada facilmente, basta projetar este ponto para a tela (z_{tela}) e acessar a posição correspondente do *depth-buffer* (z^*) e comparar os valores ($z_{tela} < z^*$).

A integral sobre a esfera, $\int_S \tau(P)dP$, resulta no volume ocludido da esfera tangente. Consequentemente o valor da oclusão em P é a razão do volume ocludido pelo volume total.



(2.3(a)) (2.3(b))

Figura 2.3: (a) Representação de uma esfera com alguns cilindros utilizados para amostrar a geometria. (b) Disposição dos pontos z^*, z_{in} e z_{out}.

Pode-se computar a integral $\int_S \tau(P)dP$ ao considerar o volume como uma soma de cilindros no espaço do olho (Figura 2.3(a)). A posição de cada cilindro é determinada por uma amostra disposta em um disco perpendicular ao eixo z e de mesmo centro que a esfera tangente. Os eixos dos cilindros possuem a mesma área de seção e são paralelos ao eixo z. Cada cilindro deve corresponder a porção interna da esfera para que o volume ocupado da esfera seja calculado. Para tanto, a altura do cilindro é delimitada totalmente pela esfera ou pela superfície de separação e pela esfera. A Figura 2.3(b) apresenta os 3 casos possíveis para a altura do cilindro. z_{in} e z_{out} são os pontos de interseção do eixo do cilindro com a esfera e z^* é o valor presente no *depth-buffer*.

- $\Delta z_i = z_{out} - z_{in}$ quando $z^* < z_{in}$. Eixo à esquerda na Figura 2.3(b).
- $\Delta z_i = 0$ quando $z_{out} < z^*$. Eixo central na Figura 2.3(b).
- $\Delta z_i = z_{out} - z^*$ quando $z_{in} < z^* < z_{out}$. Eixo à direita na Figura 2.3(b).

Dessa forma a integral $\int_S \tau(P)dP$ pode ser aproximada computacionalmente por:

$$\int_S \tau(P)dp \approx \frac{(\frac{r}{2})^2\pi}{n} \sum_{i=1}^{n} \Delta z_i \qquad (2\text{-}4)$$

onde $\frac{(\frac{r}{2})^2\pi}{n}$ é a área da seção perpendicular ao eixo do cilindro.

Substituindo a equação 2-4 em 2-3 obtém-se:

$$V(P) \approx \frac{\frac{(\frac{r}{2})^2\pi}{n}}{|S|} \sum_{i=1}^{n} \Delta z_i \tag{2-5}$$

Percebe-se que $\frac{(\frac{r}{2})^2\pi}{n} / |S|$ pode ser resolvido analiticamente. Contudo, ao invés de utilizar o volume exato da esfera ($|S| = \frac{4}{3}\pi(\frac{r}{2})^3$) aproxima-se o seu volume usando os mesmos cilindros, mas ao invés da altura do cilindro ser Δz utiliza-se $z_{out} - z_{in}$, o que reduz o erro da integração. A equação final é:

$$V(P) \approx \frac{\sum_{i=1}^{n} \Delta z_i}{\sum_{i=1}^{n} (z_{out} - z_{in})} \tag{2-6}$$

2.2 Voxelização sólida em tempo real

A abordagem apresentada por Eisemann e Décoret (6) objetiva a conversão eficiente de um modelo *watertight* em um volume binário com interior sólido de boa resolução em uma única passada. Este processo é chamado de voxelização sólida.

A voxelização sólida consiste em enviar a geometria para o *pipeline* gráfico e obter uma grade regular composta por *voxels* indicando a presença de geometria. Este processo é considerado sólido pois produz *voxels* no interior dos objetos e não somente na borda da geometria. Utiliza-se a rasterização do *hardware* para discretizar a geometria e cada fragmento apoia a construção da estrutura volumétrica

A estrutura que mantém a informação do volume é uma grade regular binária, e é representada por uma textura 2D. Um determinado *voxel* (i, j, k) pode ter os valores 0 (ausência de geometria) ou 1 (presença de geometria), e está codificado no k-ésimo *bit* da representação RGBA de um *texel* (i, j). A Figura 2.4 ilustra essa organização.

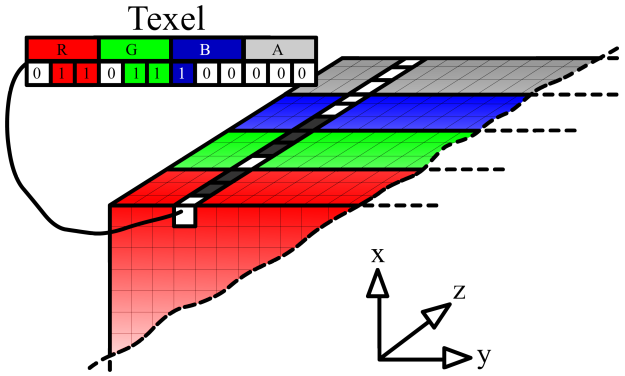


Figura 2.4: Exemplo de uma grade regular. As direções x e y delimitam a textura. A direção z é composta pelos *bits* dos *texels*.

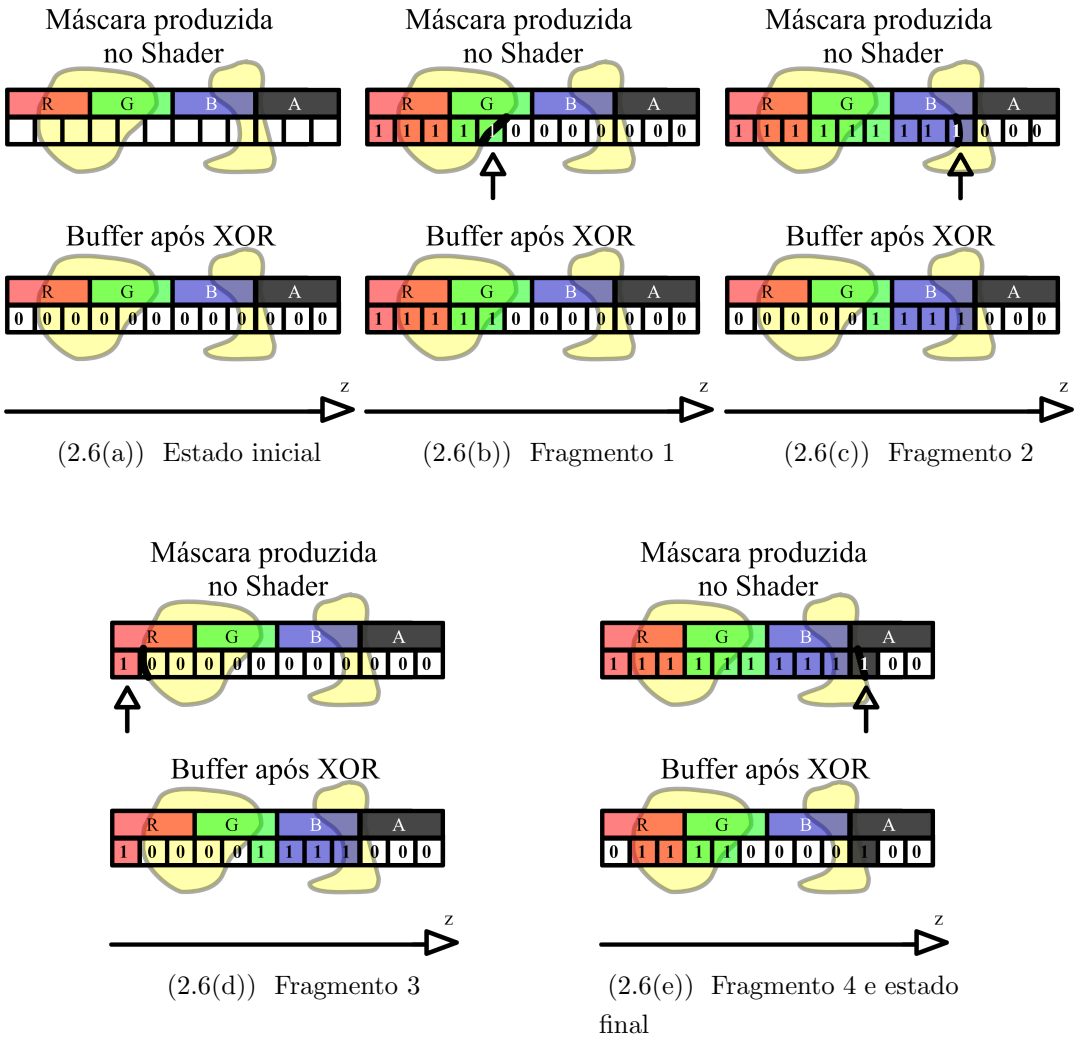
A dimensão da textura define as dimensões x e y da grade, e o número de *bits* da representação RGBA do *texel* forma a dimensão z. A resolução de uma grade representada em uma textura RGBA de 32 *bits* na dimensão z é igual a 128 (4 canais x 32 *bits*).

O método de voxelização proposto é bastante eficiente, mas possui a restrição de aceitar somente modelos chamados de *watertight*. Segundo Nooruddin e Turk (10), um modelo é *watertight* se para cada componente conectado no espaço, todos os seus pontos compartilham a mesma classificação: serem internos ou externos. Um ponto no espaço é considerado interno/externo se o número de interseções de um raio qualquer originado deste ponto com o modelo é par/impar (Teorema de Jordan). A Figura 2.5 apresenta exemplos de alguns modelos e como estes são classificados.

Figura 2.5: Exemplo de modelos. O modelo mais à esquerda é considerado *watertight*, os outros não se encaixam na definição.

O teorema de Jordan é utilizado para verificar se um determinado fragmento está no interior/exterior contando o número de fragmentos que estão na frente deste. Ou seja, o *voxel* é considerado interior se o número de fragmentos a sua frente for ímpar ($n \ mod \ 2 = 1$). Uma maneira de realizar esta operação é utilizar um contador por *voxel* e sempre que um fragmento for processado incrementar o contador de todos os *voxels* a sua frente. Manter um contador por *voxel* seria impraticável, mas através da operação XOR pode-se encapsular a operação de incrementar e realizar o *mod* 2 em um contador de um único *bit*, para isso basta que cada fragmento gere uma máscara de *bits* onde todos os *bits* menores que o índice do *voxel* que ele representa sejam 1.

A Figura 2.6 ilustra o processo de voxelização sólida para uma coluna paralela ao eixo z da grade. A cena ilustrada é composta por dois objetos e produz quatro fragmentos na coluna apresentada na figura. Os fragmentos são emitidos pelo *pipeline* sem ordem definida e cada um produz uma máscara de bits em sua saída (coluna RGBA superior). A saída de cada fragmento realiza a operação binária XOR com o valor corrente do *buffer* de saída (o resultado é apresentado na coluna RGBA inferior das figuras). Após a execução de todos os fragmentos o *buffer* de saída contém a grade resultante com a voxelização sólida da geometria.



(2.6(a)) Estado inicial (2.6(b)) Fragmento 1 (2.6(c)) Fragmento 2

(2.6(d)) Fragmento 3 (2.6(e)) Fragmento 4 e estado final

Figura 2.6: Voxelização sólida para uma coluna da grade. Foi considerada uma textura com apenas 3 *bits* por canal. As imagens representam a iteração do algoritmo. A saída do *shader* e o resultado do *buffer* de saída após a operação lógica XOR são exibidas para cada passo.

2.3 Oclusão ambiente em espaço do volume

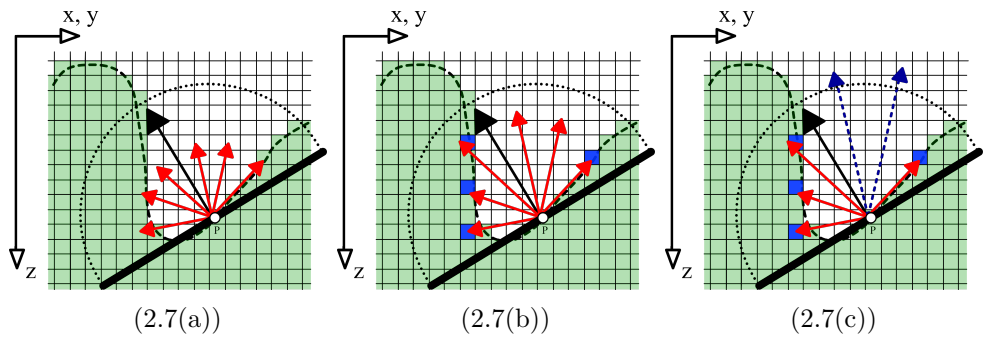
A oclusão ambiente é um efeito global e leva em conta a contribuição da luz em diversas direções, para tanto, a obtenção de um resultado fisicamente coerente deve considerar a geometria na sua totalidade ao invés de somente a porção visível pelo observador (13). As técnicas que utilizam uma representação baseada em *voxels* para estimar a geometria são categorizadas como oclusão ambiente em espaço do volume.

A seguir são apresentados dois trabalhos que almejam resolver a oclusão ambiente fazendo uso de estruturas que discretizam a geometria através de *voxels*.

2.3.1
Ray-marching em espaço do volume

O trabalho apresentado por Papaioannou et al. (13) propõe um método para amostrar a visibilidade utilizando dados volumétricos em tempo real visando produzir oclusão ambiente de aspecto global. O método proposto desacopla a representação da superfície dos cálculos de visibilidade ao utilizar *ray-marching* em volumes, pré-calculados ou dinamicamente gerados. Devido à utilização da representação volumétrica a abordagem é independente do ponto de vista do observador.

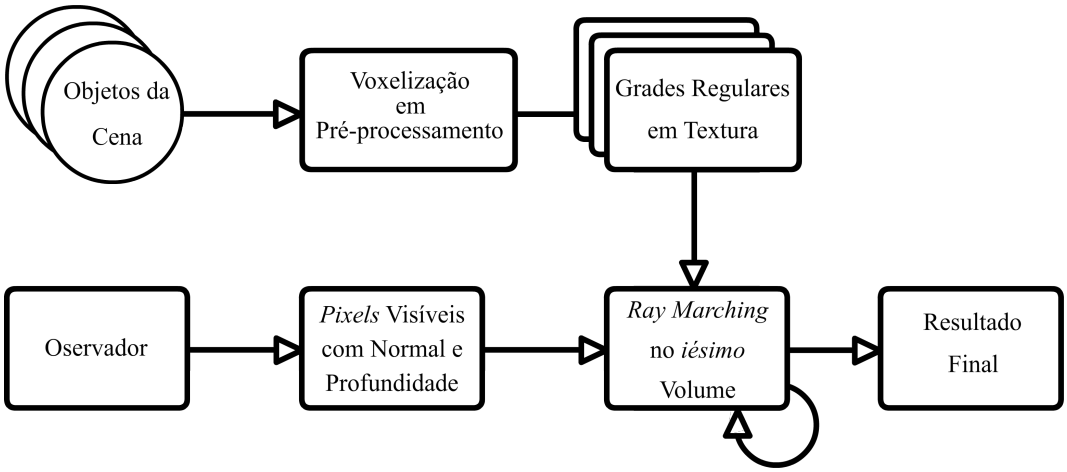
Papaioannou et al. sugerem que o traçado de raios seja substituido pela técnica de *ray-marching* afim de verificar a visibilidade. *Ray-marching* consiste no lançamento de raios em diversas direções, contudo os raios são iterados a passos constantes em uma grade regular e a cada passo verifica-se a presença de geometria no *voxel* que ele se encontra. Caso o *voxel* esteja preenchido a iteração desse raio pára. Devido à utilização de passos pequenos e de um processo de voxelização sólida esta técnica aproxima com eficiência o traçado de raios. A Figura 2.7 demonstra o conceito.



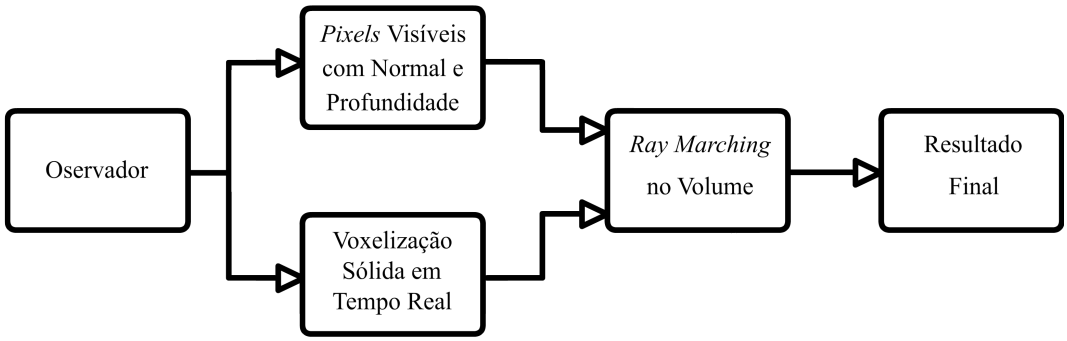
(2.7(a)) (2.7(b)) (2.7(c))

Figura 2.7: Iterações do método de *ray-marching*. (a) Raios lançados a partir do ponto P para amostrar o volume voxelizado. (b) Uma iteração dos raios, os *voxels* marcados em azul foram atingidos e a iteração destes raios parou. (c) Estado final dos raios, os raios tracejados ultrapassaram a distância do hemisfério sem atingir a geometria.

Um esquema do funcionamento do algoritmo proposto pode ser visto na Figura 2.8. A ideia central consiste em, para cada *pixel* visível, realizar *ray-marching* em todos os volumes da cena através de múltiplas passadas. Os volumes utilizados podem ser criados em pré-processamento ou gerados dinamicamente a cada quadro pela técnica de voxelização apresentada anteriormente na Seção 2.2 deste capítulo. Se somente for utilizado o processo de voxelização sólida em tempo real apenas uma passada é necessária (Figura 2.8(b)).



(2.8(a))



(2.8(b))

Figura 2.8: Esquema do funcionamento do algoritmo. (a) Considera a utilização dos volumes construidos em pré-processamento. (b) É utilizado somente a voxelização em tempo real para construir a informação voluémtrica da geometria.

2.3.2 Traçado de cones em voxels

Crassin et al. (2) introduzem uma abordagem construída em torno de uma representação voxelizada hierárquica pré-filtrada da geometria da cena. Essa representação é mantida na GPU na forma de uma *octree* dinâmica de *voxels* gerada a partir de malhas de triângulos. Através dessa estrutura são feitos os testes de visibilidade usando traçado de cones.

A utilização de cenas dinâmicas é possível devido ao algoritmo de atualização da *octree*. A *octree* é construída uma vez para as cenas estáticas e somente atualizada com a movimentação de objetos ou alterações na geometria. A estrutura definida é baseada em ponteiros na GPU e a organização da memória permite tirar proveito dos filtros disponíveis pelo *hardware*. Após

a *octree* ter sido construída, os valores das folhas sofrem o processo de *mipmapping* para preencher os níveis superiores da *octree*.

A ideia do traçado de cones é aproximar os resultados de um pacote de raios partindo de um ponto. O eixo do cone é percorrido realizando acessos à estrutura hierárquica em níveis diferentes de acordo com o raio do cone, Figura 2.9. A oclusão ambiente é estimada através do lançamento de diversos cones no hemisfério em torno da normal de um *pixel* e o resultado em cada cone é somado e normalizado pelo número total de cones para obter a oclusão total.

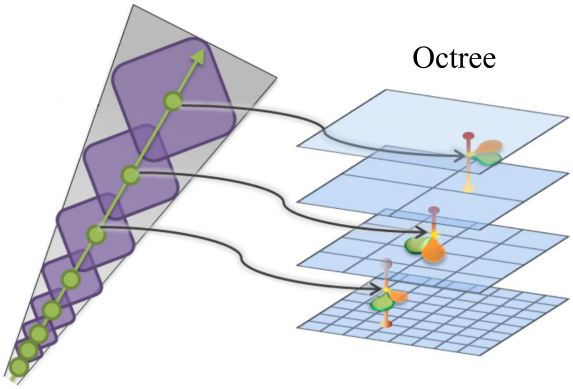


Figura 2.9: Traçado de cones usando uma *octree*. A cada iteração ao longo do eixo do cone uma amostra acessa um nível diferente da *octree*. Imagem retirada de (2).

2.4 Considerações e proposta

Os métodos apresentados nas seções anteriores visam estimar o valor da oclusão para cada *pixel* visível em tempo real. Os dois métodos apresentados na Seção 2.1 obtêm resultados com certa eficiência, pois trabalham no espaço da imagem; contudo, somente a utilização da informação presente na tela produz resultados incorretos e variáveis de acordo com o ponto de vista (13). Pode-se aumentar a informação utilizando várias passadas de *deepth peeling*; entretanto, dessa forma as múltiplas passadas de geometria acabam interferindo no desempenho.

A abordagem proposta por Papaioannou et al. (13) utiliza *ray-marching* em um volume, o que acaba com a dependência do ponto de vista. Contudo o processo de *ray-marching* ainda necessita de uma grande quantidade de raios para produzir um resultado de qualidade, o que reduz o desempenho.

A técnica descrita por Crassin et al. (2) constrói uma estrutura muito complexa que tira proveito da arquitetura do *hardware* e lança cones para amostrar a geometria voxelizada em diversos níveis de uma *octree* que é atualizada dinamicamente. O desempenho é eficiente e os resultados apresentam boa

qualidade. Apesar disso, a criação e manutenção dessa estrutura hierárquica é complicada e a *octree* nem sempre produz resultados bons para amostras devido a sua divisão espacial baseada em octantes.

Com esta dissertação objetiva-se desenvolver um método que aproxime a oclusão ambiente com qualidade e desempenho. A voxelização em tempo real e a utilização de um método bastante eficiente de amostrar a geometria torna o método proposto nesta dissertação uma solução interessante para aplicações que busquem qualidade e desempenho.