Pontifícia Universidade Católica
do Rio de Janeiro

Leonardo de Paula Batista Benevides

Oclusão de Ambiente para Renderização de Linhas

Dissertação de Mestrado

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós–graduação em Informática do Departamento de Informática da PUC-Rio.

Orientador: Prof. Waldemar Celes Filho

Rio de Janeiro
Setembro de 2015

Pontifícia Universidade Católica
do Rio de Janeiro

Leonardo de Paula Batista Benevides

Oclusão de Ambiente para Renderização de Linhas

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós–graduação em Informática do Departamento de Informática do Centro Técnico Científico da PUC-Rio. Aprovada pela Comissão Examinadora abaixo assinada.

Prof. Waldemar Celes Filho
Orientador
Departamento de Informática — PUC-Rio

Prof. Alberto Barbosa Raposo
Departamento de Informática – PUC–Rio

Prof. Luiz Henrique de Figueiredo
Instituto Nacional de Matemática Pura e Aplicada – IMPA

Prof. Marcelo Gattass
Departamento de Informática – PUC–Rio

Prof. José Eugenio Leal
Coordenador Setorial do Centro Técnico Científico — PUC-Rio

Rio de Janeiro, 25 de setembro de 2015

Leonardo de Paula Batista Benevides

Graduou-se em Engenharia da Computação na Pontifícia Universidade Católica do Rio de Janeiro (PUC–Rio) em 2012. Durante a graduação fez iniciação científica no Departamento de Elétrica trabalhando com Sistemas Multi Agentes e posteriormente no laboratório Tecgraf, participando do grupo Célula de Automação de Engenharia. Em 2013 ingressou no Mestrado em Informática na PUC-Rio. Durante o mestrado trabalhou no desenvolvimento de ferramentas de visualização científica 3D no grupo de Visualização do laboratório Tecgraf.

Ficha Catalográfica

de Paula Batista Benevides, Leonardo

Oclusão de Ambiente para Renderização de Linhas / Leonardo de Paula Batista Benevides; orientador: Waldemar Celes Filho. — Rio de Janeiro : PUC-Rio, Departamento de Informática, 2015.

66 f: il. (color.); 29,7 cm

Dissertação (mestrado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, 2015.

Inclui bibliografia.

1. Informática – Teses. 2. Oclusão de Ambiente;. 3. Linhas;. 4. Computação Gráfica;. 5. Tempo Real.. I. Celes Filho, Waldemar. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

Agradecimentos

Gostaria de agradecer inicialmente ao meu orientador Waldemar Celes Filho pela oportunidade de trabalhar ao seu lado e, por ser o maior incentivador na superação de meus limites, me auxiliando de forma incansável nas incontáveis dúvidas que surgiram ao longo deste trabalho.

Aos meus pais que sempre me apoiaram na busca pelo conhecimento e aperfeiçoamento dos meus estudos, não medindo esforços para que eu pudesse alcançar meus objetivos. A minha namorada que muitas vezes me abstive de sua companhia mas sempre esteve em meu coração.

Ao Tecgraf e aos meus colegas Chrystiano, Andrei, Mariana, Bernardo, Vitor, Otávio e Alice que sempre me apoiaram e compartilharam meu dia a dia.

Aos meus amigos Lucas, Carlo, Caio, João, Manoela, Clara, Letícia, Duda, Thaís, Bernardo, Renato, Cauê, André, Pedro, Alexandre, Gustavo, Léo, Guilherme e Henrique que muitas vezes deixei de estar com eles fisicamente mas sempre presentes nos meus pensamentos.

Ao CNPq, a CAPES e a PUC-Rio, pelos auxílios concedidos, sem os quais este trabalho não poderia ter sido realizado.

Resumo

de Paula Batista Benevides, Leonardo; Celes Filho, Waldemar. **Oclusão de Ambiente para Renderização de Linhas**. Rio de Janeiro, 2015. 66p. Dissertação de Mestrado — Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

A interpretação tridimensional de conjuntos densos de linhas exige o uso de modelos de iluminação mais elaborados. A oclusão ambiente é uma técnica utilizada para simular, de forma realista e barata, a iluminação ambiente indireta. Este trabalho apresenta um novo algoritmo para renderização de linhas com oclusão de ambiente. O algoritmo proposto é baseado na voxelização da cena e no cálculo do volume ocupado do hemisfério associado a cada ponto do linha. Propõe-se uma adaptação no algoritmo de voxelização de cenas 3D formada por sólidos para o tratamento correto da cena formada por linhas. Assim, uma descrição volumétrica da geometria é criada em um *buffer* de textura. O hemisfério em torno de cada ponto visível é amostrado por diversos pontos, e para cada amostra se acessa um prisma, cujo volume ocupado é calculados a partir da voxelização. Ao acumular os resultados de cada amostra, estima-se a oclusão de ambiente causada pela geometria em cada ponto visível pelo observador. Esta estratégia mostra-se adequada, pois tem como resultado imagens com alta qualidade em tempo real para cenas com grande complexidade.

Palavras–chave

Oclusão de Ambiente; Linhas; Computação Gráfica; Tempo Real.

Abstract

de Paula Batista Benevides, Leonardo; Celes Filho, Waldemar (Advisor). **Ambient Occlusion for Line Rendering**. Rio de Janeiro, 2015. 66p. MSc Dissertation — Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

The three-dimensional understanding of dense line sets requires the use of more sophisticated lighting models. Ambient occlusion is a technique used to simulate realistically and efficiently, the indirect ambient lighting. This paper presents a new algorithm for rendering lines with ambient occlusion. The proposed algorithm is based on the voxelization of the scene and on the computation of occlusionin the hemisphere associated to each visible point. It is proposed an adaptation of the voxelization algorithm of 3D scenes made up of solids to the correct treatment of the scene formed by lines. Thus, a volumetric geometry description is created in a texture buffer. The hemisphere around every visible point is sampled by several points, and for each sample is generated a prism, which occluded volume is calculated from the voxelization. By accumulating the results of each sample, the estimated ambient occlusion caused by the geometry at each point visible to the observer is computed. This strategy proved to be appropriate, resulting in high-quality images in real time for complex scenes.

Keywords

Ambient Occlusion; Lines; Computer Graphics; Real Time.

Sumário

Lista de figuras

Lista de tabelas

Se vi mais longe foi por estar de pé sobre ombros de gigantes.

Isaac Newton, *Carta para Robert Hooke, 5 de Fevereiro de 1676.*

1
Introdução

A reprodução de uma cena na tela de um computador é um desafio que incentivou os cientistas da computação a pesquisarem e desenvolverem a área da computação gráfica, estudando as formas da interação da luz com o ambiente para a criação de imagens realistas. Esse modelo de iluminação pode ser simplificado de modo a levar em considerações características locais ou pode tentar reproduzir todos os efeitos produzidos por reflexões e refrações dos raios de luz numa cena de forma global.

Dentre as inúmeras aplicações para o uso da iluminação global, podemos destacar: aplicações de visualização científica e médica, jogos, edição e produção de imagens para cinema e televisão. Presente em muitas áreas em especial na área de visualização científica, está a renderização de conjuntos de linhas. É comum encontrarmos linhas para visualizar fluxo e escoamento de líquidos, campos vetoriais em 3D, representação médica e atômica, onde as linhas são usadas de forma intuitiva com o objetivo de simular e expressar características que são facilmente compreendidas [6, 7]. Cada tipo de dado inerente às linhas, e o uso científico associado a elas, tem seu próprio conjunto de propriedades e restrições. No entanto, conjuntos de linhas grandes e densos, independente de sua natureza, sofrem de problemas relacionados à compreensão da forma de suas estruturas e às relações espaciais entre elas como pode ser visto na Figura 1.1.

Muitos métodos foram propostos para resolver esse problema [2, 8–10], buscando melhorar a percepção espacial de linhas individuais ou de conjuntos, mas falham em solucionar a complexa relação entre densos conjuntos de linhas, visto que o cálculo de iluminação global destes modelos é extremamente caro, restringindo-se assim as aplicações que podem fazer uso do mesmo.

A oclusão de ambiente é um exemplo de uma técnica que produz resultados que se aproximam da iluminação global, mas com a possibilidade de execução em ambientes de tempo real por possuir um custo mais baixo de processamento. Essa diferença de custo se dá porque, na iluminação global, lançam-se raios para cada pixel visível da tela e se realizam cálculos para reflexões e refrações; já na oclusão de ambiente, calcula-se a iluminação ambiente

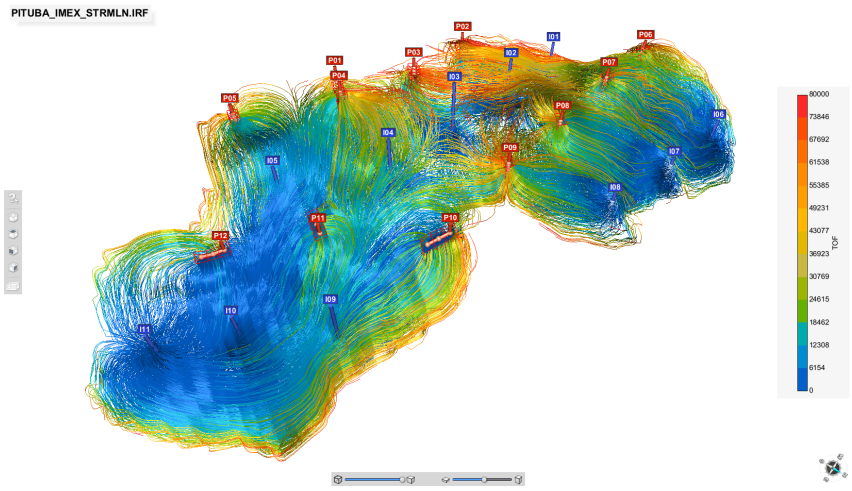


Figura 1.1 – Imagem das linhas de fluxo de um reservatório de petróleo retiradas do software Geresim, ilustrando o problema da percepção espacial entre as linhas.

em cada ponto considerando apenas raios lançados em sua vizinhança. Assim, esse método consegue gerar cenas com qualidade que ajudam na interpretação do modelo. A Figura 1.2 exibe uma aplicação que utiliza oclusão ambiente, ilustrando o ganho de detalhes na percepção da cena.

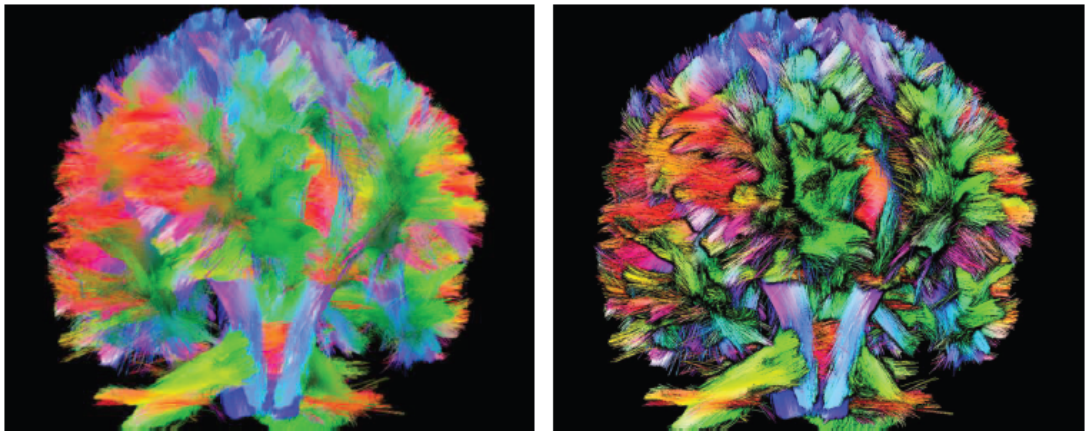


Figura 1.2 – Imagem mostrando a diferença de uma cena sem e com Oclusão de Ambiente [1].

A iluminação indireta é um fator que contribui muito para o realismo da cena. A oclusão ambiente produz efeitos de iluminação como: sombras de contato, cantos obscurecidos, escurecimento por proximidade entre objetos, falhas e rachaduras. O cálculo da oclusão de ambiente almeja representar a incidência da luz refletida proveniente de todas as direções. Logo, avalia-se o entorno de um ponto, se esse possuir muitos oclusores receberá menos iluminação, visto que incidirá nele pouca luz proveniente do ambiente.

Técnicas de renderização de linhas mais simples frequentemente usam iluminação local e *shading* para enfatizar a forma ou as relações espaciais entre

linhas, mas sofrem sérias limitações para mostrar simultaneamente detalhes estruturais locais e globais e relações espaciais [1].

Diferentemente de modelos de iluminação mais simples, que aproximam a luz ambiente sendo constante para todos os pontos por um valor proporcional referente ao material do objeto [11], o fator de oclusão de ambiente expressa o quanto um ponto está obscurecido por outras superfícies, se aproximando assim de ser um modelo físico mais correto, pois verifica a quantidade de luz que não está chegando em um ponto a partir de todas as direções presentes no hemisfério normal à sua superfície. A Figura 1.3 ilustra o processo no qual é calculado o número de raios que atingem geometrias oclusoras.

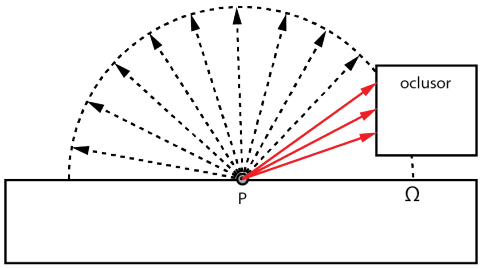


Figura 1.3 – Raios partindo do ponto P e atingindo a geometria oclusora dentro do hemisfério Ω.

Podemos descrever matematicamente a oclusão ambiente no ponto P de normal $\hat{n}$, cuja a superfície definida é tangente ao ponto, através do cálculo da integral sobre o hemisfério unitário, como visto na Equação 1.1.

$$AO(P,\hat{n}) = \frac{1}{\pi}\int_{\Omega} V(P,\hat{\omega})\,\langle\hat{\omega}\cdot\hat{n}\rangle\,d\omega \tag{1.1}$$

A função binária de visibilidade $V(P,\hat{\omega})$ verifica se o ponto no hemisfério está ocludido ou não; o valor deve ser 1 se o ponto estiver obstruído pela geometria na direção $\hat{\omega}$ ou 0 caso contrário; $\hat{\omega}$ representa um vetor unitário partindo do ponto P as várias direções na superfície do hemisfério Ω.

Pode-se notar que essa integral não possui uma solução analítica fácil, sendo normalmente aproximada por uma soma de Riemann:

$$AO(P,\hat{n}) = \frac{1}{\pi}\lim_{s\to\infty}\sum_{i=1}^{s} V(P,\hat{\omega}_i)\,\langle\hat{\omega}_i\cdot\hat{n}\rangle\,\frac{\pi}{s} \tag{1.2}$$

que pode ser truncada por uma série com s amostras, sendo s suficientemente grande e com uma distribuição esparsa de $\omega_i\ \epsilon\ \Omega$ no hemisfério unitário.

$$AO(P,\hat{n}) \approx AO_s(P,\hat{n}) = \frac{1}{s}\sum_{i=1}^{s} V(P,\hat{\omega}_i)\,\langle\hat{\omega}_i\cdot\hat{n}\rangle \tag{1.3}$$

Esse último somatório pode então ser facilmente implementado com programação em placa gráfica (GPU).

Encontramos na literatura diversos modelos de iluminação que podem ser aplicados para linhas 3D [11, 12]. O traçado de raios por exemplo, garante resultados fisicamente corretos, mas com um elevado custo computacional. Encontramos também métodos que fazem o cálculo da oclusão de ambiente em pré-processamento, gerando resultados de boa qualidade, mas falhando em tratar cenas dinâmicas [13]. Métodos que podem ser usados para aplicações em tempo real, dando suporte a cenas iterativas, amostram a vizinhança de cada *pixel* no espaço de tela para calcular sua oclusão [14–16]. Entretanto, os resultado estimados podem variar porque esses métodos são dependentes do ponto de vista da câmera. Esse problema pode ser contornado ao custo de uma perda de desempenho utilizando algoritmos que acessam estruturas de aceleração para ter acesso a informações da geometria da cena [4, 17]. Na maioria dos casos, os métodos propostos tem como alvo cenas compostas por objetos sólidos, falhando em captar características importantes da renderização de conjuntos de linhas. Portanto, esta dissertação tem como motivação a criação de um método que produza resultados com qualidade e desempenho para a aplicação na renderização de conjuntos densos de linhas.

Neste trabalho propõe-se um algoritmo que utiliza uma estrutura de dados que torna o acesso à informação geométrica das linhas mais fácil, permitindo que seja estimado o valor da oclusão ambiente ao redor de cada *pixel*. Podemos comparar essa estrutura à uma grade regular que discretiza a geometria em *voxels*. O cálculo da oclusão ambiente em cada ponto se dá através do uso de amostras distribuídas no hemisfério de interesse. Cada amostra acessa um prisma de geometria contido no hemisfério, e o volume ocludido de cada prisma é usado para determinar a oclusão do ponto.

Analisando os resultados, verificamos que o algoritmo proposto estima corretamente a oclusão ambiente para linhas de forma eficiente, tendo como contribuição o método de voxelização da geometria, que permite que a oclusão seja computada rapidamente considerando apenas uma porção da informação disponível.

Este trabalho está organizado da seguinte forma: O Capítulo 2 discute os trabalhos relacionados existentes que abordam a oclusão ambiente e iluminação de linhas. No Capítulo 3 o método proposto é detalhado. Os resultados são apresentados no Capítulo 4. Por último, o Capítulo 5 apresenta as considerações finais da dissertação.

2
Trabalhos Relacionados

Analisando a literatura, podemos encontrar inúmeras soluções para o cálculo da oclusão de ambiente e para modelos de iluminação de linhas. A seguir, são destacadas as principais abordagens que inspiraram este trabalho: a Seção 2.1 trata de modelos de iluminação de linhas, os quais podem ser combinados com o resultado do método proposto para a sua visualização. A Seção 2.2 explica a técnica usada para fazer a voxelização da cena, a qual é utilizada como modelo na solução proposta nesta dissertação. A Seção 2.3 detalha algoritmos que calculam a oclusão de ambiente em espaço de tela, sendo o segundo um algoritmo que busca solucionar o mesmo problema que abordamos. A Seção 2.4 trata de abordagens que usam informações da geometria da cena para estimar a oclusão ambiente. Ponderações sobre cada um desses trabalhos apresentados são feitas na Seção 2.5.

2.1
Iluminação de Linhas

Os modelos de iluminação de Phong [12] e Blinn-Phong [11] precisam do material da superfície sendo iluminada, as propriedades da luz e três vetores unitários: L (partindo do ponto P que está sendo iluminado na superfície até a fonte de luz), V (partindo de P até a câmera) e N (normal orientada da superfície), para o cálculo da iluminação:

$$I = I_a + I_d + I_s = k_a + k_d L \cdot N + k_s (V \cdot R)^n \tag{2.1}$$

onde k_a, k_s *e* k_s são os coeficiente de reflexão ambiente, difuso e especular, n é o expoente especular e R é a reflexão de L em N, podendo ser substituída, para simplificar o cálculo, pelo vetor *halfway* $H = (V + L)/||V + L||$.

Por outro lado, para um ponto P numa curva no espaço, não existe uma única normal definida, sendo então computada pelos vetores *V,* L e T (tangente da curva no ponto P). Sendo definida a tripla *(T,N,B)*, onde $B = T \times N/||T \times N||$, a normal pode ser expressa por $N = N\cos\theta + B\sin\theta$. Se a curva é entendida como um cilindro infinitesimal ela tem em P o vetor normal $N_\theta = N\cos(\theta) + B\sin(\theta)$ onde a fase θ é escolhida como $N_0 = N$, ilustrado na Figura 2.1 [2].

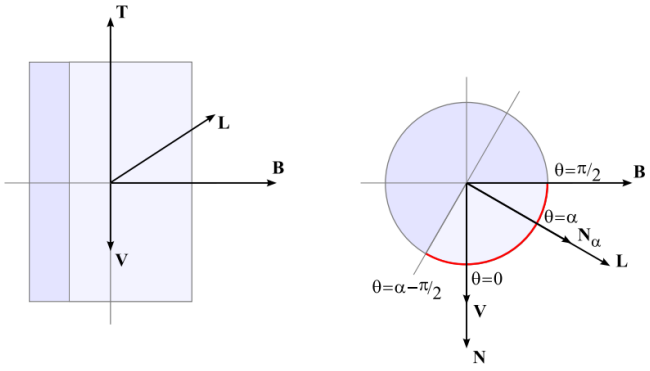


Figura 2.1 – Vista frontal e superior de um cilindro, com vetores unitários. Setor visível da reflexão difusa(arco vermelho). Retirado de [2].

Banks [8] introduziu a ideia de maximizar a luz refletida sobre o perímetro infinitesimal de um cilindro fino, tratando separadamente as componentes difusa e especular. No caso de uma reflexão especular com muito brilho, essa aproximação é boa, já que ângulos próximos do máximo são os que mais contribuem no cálculo integral. O cálculo do vetor normal se dá então pela escolha dos ângulos θ que o produto interno no cálculo da componente difusa e especular é máximo. Os termos difuso e especular são dados abaixo:

$$I_d = k_d\sqrt{1 - L_T^2} \tag{2.2}$$

$$I_s = k_s(-V_T L_T + \sqrt{1 - V_T^2}\sqrt{1 - L_T^2})^n \tag{2.3}$$

A principal desvantagem da iluminação utilizando o princípio da máxima reflexão é a aproximação do cálculo da reflexão difusa, pois a iluminação difusa não é dependente do observador.

Mallo et al. [2] utilizam o método da média do cilindro como uma alternativa para o princípio da reflexão máxima, onde a curva no espaço é tratada como o limite de um tubo cilíndrico de raio tendendo a zero. O cilindro é considerado opaco gerando auto-oclusão, e as reflexões difusa e especular são calculadas então integrando sobre a parte visível.

Este método calcula a reflexão difusa e especular integrando a reflexão nas faces infinitesimais de um cilindro, onde a contribuição de cada face para o total refletido depende da área projetada vista da direção V [2].

$$I_d = k_d\sqrt{1 - L_T^2}\frac{\sin\alpha + (\pi - \alpha)\cos\alpha}{4} \tag{2.4}$$

sendo α:

$$\alpha = \arccos\frac{V \cdot L - V_T L_T}{\sqrt{1 - V_T^2}\sqrt{1 - L_T^2}} \tag{2.5}$$

$$I_s = \sqrt{1 - H_T^2}^n \int_{\alpha-\frac{\pi}{2}}^{\frac{\pi}{2}} \cos^n(\theta - \beta)\frac{\cos(\theta)}{2}d\theta \tag{2.6}$$

Nota-se que β é o ângulo entre o vetor V projetado e o vetor H projetado, e pode variar entre 0 e π.

2.2 Voxelização em tempo real

Eisemann e Décoret [3] propõem um processo eficiente, com uma única passada de renderização chamado de voxelização sólida, cujo objetivo é a conversão de um modelo *watertight* em um volume binário com interior sólido.

Um modelo é considerado *watertight* se, para cada componente conectado no espaço, todos os seus pontos possuem a mesma classificação: serem externos ou internos [18].

Através do envio da geometria para o *pipeline* gráfico, utiliza-se a rasterização da placa para discretizar a geometria e cada fragmento contribui para a construção da estrutura volumétrica sólida, composta por *voxels* indicando a presença de geometria no interior e na borda dos objetos.

Dispõe-se de uma textura 2D para armazenar a grade regular binária que mantém a informação do volume voxealizado. A dimensão da textura define as dimensões x e y da grade, e a dimensão z é dada a partir do número de *bits* da representação RGBA do *texel*. Um *voxel* (i, j, k), por ser binário, só possui duas opções de valores: 0 (ausência de geometria) ou 1 (presença de geometria), e está codificado no k-ésimo *bit* da representação RGBA de um *texel*, que tem resolução máxima 128 em uma textura RGBA(4 canais) de 32 *bits*. A Figura 2.2 ilustra essa organização.

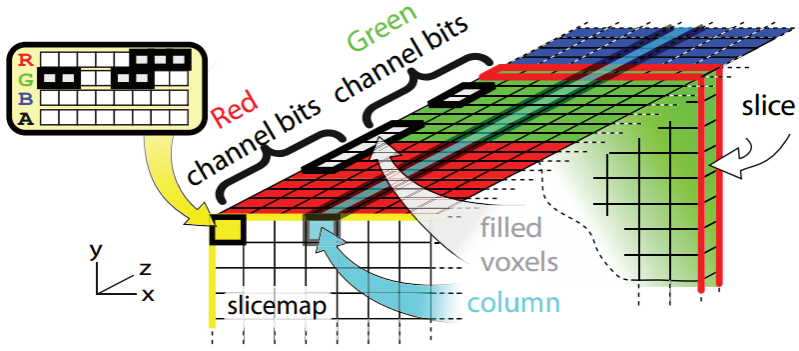


Figura 2.2 – Exemplo de uma grade regular. As direções x e y delimitam a textura. A direção z é composta pelos *bits* dos *texels*. Imagem retirada de [3]

Se o número de fragmentos à frente de um *voxel* for ímpar (n *mod* 2 = 1), o *voxel* é considerado interior. Para realizar esse cálculo de forma eficiente, utiliza-se a operação *XOR* para incrementar e realizar o *mod* 2 em um contador de um único *bit*. Para isso basta gerar uma máscara de *bits* para cada fragmento, onde todos os *bits* menores que o índice do *voxel* estejam acesos.

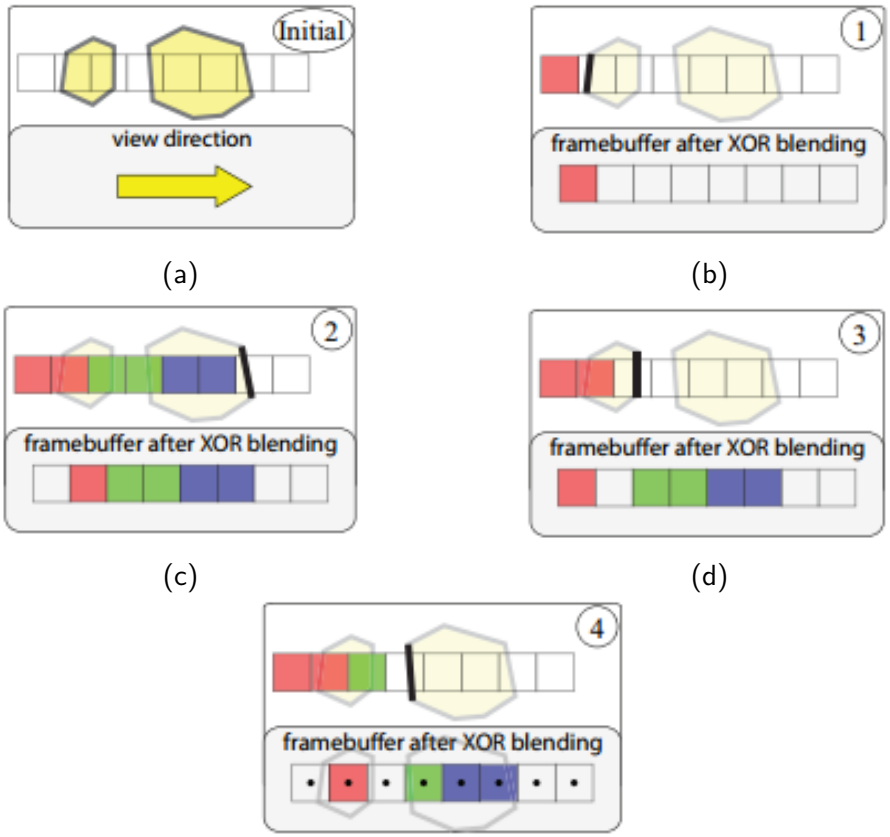


Figura 2.3 – Voxelização para um textel. Por simplicidade considera-se um *framebuffer* com dois bits por canal de cor. A cena consiste de dois objetos *watertight*, onde durante o *rendering* os fragmentos chegam em ordem aleatória (a). Iteração do algoritmo (b) - (e) com a saída do *shader* e o resultado do *buffer* após a operação XOR para cada passo do algoritmo (o resultado é apresentado na coluna RGBA inferior das figuras). Em (e) o *framebuffer*contém a voxelização sólida no *grid*. Imagem retirada de [3]

A voxelização sólida para uma coluna paralela ao eixo z da grade de uma cena com dois objetos está ilustrada na Figura 2.3. São produzidos fragmentos sem ordem definida que são emitidos pelo *pipeline* e cada um produz uma máscara de bits indicando todos os voxels existentes na frente do fragmento corrente. A saída de cada fragmento realiza a operação binária XOR com o valor corrente do *buffer*. A grade resultante com a voxelização sólida da geometria se encontra no *buffer* de saída após a computação de todos os fragmentos.

2.3 Oclusão ambiente em espaço de tela

Métodos que fazem uso do espaço da tela para aproximar o cálculo da integral de oclusão (*SSAO* - Screen Space Ambiente Occlusion) são bastante utilizados em aplicações em tempo real, pois são independentes da complexidade

da geometria dos modelos da cena e podem ser usados em cenas dinâmicass, necessitando apenas das informações presentes no *depth-buffer* ou em *buffers* construídos através do uso da técnica de *deferred shading* [14–16, 19–21].

Entretanto, esses métodos podem levar ao cálculo incorreto da oclusão ambiente porque são dependentes da posição do observador que só tem a informação disponível do espaço de tela, sendo necessário muitas vezes o uso de uma estrutura de dados para complementar a informação sobre a cena.

2.3.1 Integral Volumétrica com o uso de esfera tangente

Szirmay-Kalos et al. [16] apresentam a proposta de transformar a integral direcional da oclusão ambiente, Equação 1.1, em uma integral volumétrica, não sendo mais necessário que sejam traçados raios em diversas direções para amostrar a visibilidade de cada ponto.

O método proposto pode ser resolvido na *GPU* através da verificação se um determinado ponto é interno ou externo ao volume da cena. Reduz-se também o espaço de integração eliminando as direções que contribuem pouco para a oclusão (direções de raios que formam ângulos grandes com a normal).

O volume ocludido da esfera tangente de raio $\frac{r}{2}$ substitui o lançamento de raios no hemisfério como é ilustrado na Figura 2.4.

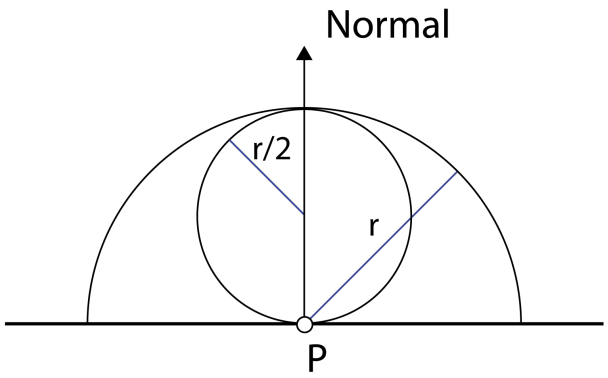


Figura 2.4 – Espaço de integração; o semi círculo indica o hemisfério em torno de P; a esfera de raio $\frac{r}{2}$ é o novo espaço de integração.

Dada a função $\tau(P)$ que testa se o ponto P pertence ao volume e $|S|$ representando o volume total da esfera tangente, podemos representar o volume ocludido da esfera tangente à superfície por:

$$V(P) = \frac{\int_S \tau(P) dP}{|S|} \tag{2.7}$$

O *depth-buffer* é utilizado como superfície de separação. Define-se o espaço que contém o observador como externo e o espaço delimitado pela superfície definida *depth-buffer* como interno. Assim sendo, qualquer ponto que possuir um valor de z maior que o valor no *depth-buffer* é considerado interno. Basta projetar um ponto para a tela (z_{tela}), acessar a posição correspondente do *depth-buffer* (z^*) e comparar os valores ($z_{tela} < z^*$) para verificar se um ponto é externo.

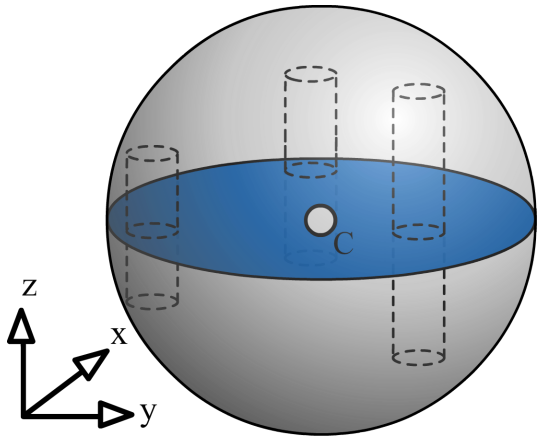


Figura 2.5 – Representação de uma esfera que tem sua geometria amostrada por cilindros em seu interior.

Se considerarmos o volume como uma soma de cilindros no espaço do olho, podemos aproximar a integral $\int_S \tau(P)dP$ calculando o volume ocludido cilindros, onde a posição de cada cilindro é dada por uma amostra disposta em um disco perpendicular ao eixo z e com o mesmo centro que a esfera tangente (Figura 2.5). Para que possamos calcular volume ocupado da esfera, cada cilindro deve corresponder a uma porção interna da esfera e a altura do cilindro é delimitada totalmente pela esfera ou pela superfície de separação.

Apresentamos na Figura 2.6 os três casos possíveis para a delimitação altura do cilindro. Tendo que z^* é o valor presente no *depth-buffer* e z_{in} e z_{out} sendo os pontos de interseção do eixo do cilindro com a esfera podemos enumeramos os possíveis casos:

- $\Delta z_i = 0$ quando $z_{out} < z^*$. Eixo central na Figura 2.6.
- $\Delta z_i = z_{out} - z_{in}$ quando $z^* < z_{in}$. Eixo à esquerda na Figura 2.6.
- $\Delta z_i = z_{out} - z^*$ quando $z_{in} < z^* < z_{out}$. Eixo à direita na Figura 2.6.

Podemos então aproximar computacionalmente a integral em 2.7 por:

$$\int_S \tau(P)dp \approx \frac{(\frac{r}{2})^2\pi}{n} \sum_{i=1}^{n} \Delta z_i \tag{2.8}$$

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

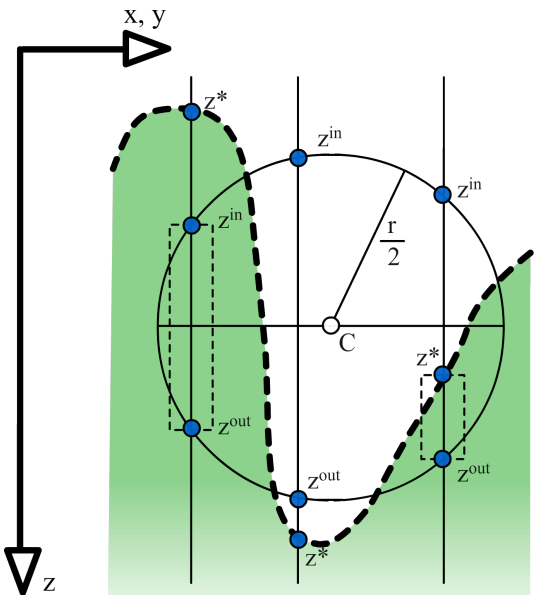


Figura 2.6 – Três casos possíveis para a delimitação da altura do cilindro.

Substituindo a equação 2.8 em 2.7 chegamos em:

$$V(P) \approx \frac{\frac{(\frac{r}{2})^2\pi}{n}}{|S|} \sum_{i=1}^{n} \Delta z_i \tag{2.9}$$

Contudo, podemos reduzir o erro da integração ao aproxima o seu volume usando os mesmos cilindros ao invés de utilizar o volume exato da esfera ($|S| = \frac{4}{3}\pi(\frac{r}{2})^3$) , com a altura do cilindro sendo $z_{out} - z_{in}$ no lugar de Δz e resolvendo analiticamente a expressão $\frac{(\frac{r}{2})^2\pi}{n} / |S|$ teremos a seguinte equação:

$$V(P) \approx \frac{\sum_{i=1}^{n} \Delta z_i}{\sum_{i=1}^{n} (z_{out} - z_{in})} \tag{2.10}$$

2.3.2 LineAO

Muitas abordagens evoluíram para solucionar o problema da compreensão das relações entre linhas dentro de um conjunto, através do uso de modelos de iluminação [2, 8] ou de renderização em forma de tubo [22]. Entretanto, eles não conseguem resolver o problema de relações espaciais complexas em conjuntos densos de linhas. Eichelbaum et al. [1] propõem um método de oclusão de ambiente que melhora a percepção espacial e estrutural de linhas, combinando iluminação global ambiente com as contribuições das reflexões das linhas na vizinhança, mantendo assim a espessura da linha que é necessária para garantir a consistência sobre interação no modelo.

LineAO [1] introduz um esquema de amostragem que resolve o problema da baixa resolução espacial em espaço de tela muito comum nas técnicas de *SSAO*. Através da extensão da Equação 1.3, pode-se amostrar as linhas que

estão perto e distante e classificá-las por *níveis de distância*.

Primeiramente, substitui-se o peso baseado por orientação das amostras para uma peso g que atenua o termo de oclusão $1 - V$, para permitir a avaliação da oclusão de ambiente em múltiplos hemisférios. Inclui-se também o parâmetro r que define o raio do hemisfério usado para amostrar os objetos ao redor do ponto P.

Entretanto, isto não é suficiente para classificar em *níveis de distância* e nem adicionar efeitos diferentes para oclusores locais e globais no ponto P, sendo necessário modificar as funções de visibilidade e peso para dependerem agora de um parâmetro l definindo o nível de distância. Quanto maior o l, mais as estruturas globais são importantes; quanto menor este valor for, mais ênfase se dará aos detalhes.

$$AO_{s,l}(P,r) = \frac{1}{s}\sum_{i=1}^{s}[(1 - V_l(r\omega_i, P))g_l(r\omega_i \cdot P)] \tag{2.11}$$

Temos então a Equação 2.11 capaz de avaliar a fração do valor da oclusão de ambiente para um hemisfério com raio r no ponto P com s_h amostras e l níveis de distância. Avaliando essa equação s_r vezes podemos calcular o efeito da oclusão de ambiente combinado, como pode ser visto abaixo:

$$LineAO_{s_r,s_h,r_0}(P) = \sum_{j=0}^{s_r-1} AO_{\frac{s_h}{j+1},j}(P, r_0 + jz(P)) \tag{2.12}$$

onde a primeira iteração $j=0$ representa o menor hemisfério que é responsável pelos detalhes locais com s_h amostras. Para as iterações seguintes o termo $\frac{s_h}{j+1}$ é o responsável por reduzir o número de amostras por hemisfério. O termo $r_0 + jz(P)$ define o raio do hemisfério de forma crescente para crescentes níveis de distância. Logo, temos que $j = 0$ para o primeiro nível e r_0 define o menor hemisfério para ser amostrado na vizinhança de uma linha. A função de zoom $z(P)$ é responsável por manter coerente o efeito da oclusão de ambiente, pois é responsável por manter a relação entre o volume suposto da linha no espaço do mundo e a grossura intrínseca da linha.

A função de visibilidade na Equação 2.11 detecta se certa parte do hemisfério ao redor do *pixel* P está ocludida ou não. A idéia é interpretar o mapa de profundidade como se fosse um campo de altura, checando se o pixel $P + r\omega_i$ no hemisfério de amostragem está mais alto que P. A visibilidade pode então ser definida pela seguinte função degrau:

$$V_l(\omega, P) = \begin{cases} 1, & \text{se } d_l(P) - d_l(P+\omega) < 0 \\ 0, & \text{caso contrário.} \end{cases}$$

para quando o valor de d_l for pequeno indicar que o objeto esta perto do observador. Usando níveis de distância, estruturas densas de linhas

se transformam em uma geometria sólida para grandes distâncias e linhas sozinhas desaparecem.

Precisa-se de um modelo mais sofisticado de pesos para as contribuições dos *pixels* que consiga lidar com estruturas locais e globais, e sua interação ao mesmo tempo. Para isso, dividi-se a função de peso em duas partes: a atenuação de visibilidade baseada em profundidade e a atenuação de oclusão pelas propriedade da iluminação das superfícies:

$$g_l(\omega, P) = g_l^{depth}(\omega, P) \cdot g_l^{light}(\omega, P). \tag{2.13}$$

Para a atenuação baseada em profundidade, usa-se a mesma diferença de profundidade usada na função de visibilidade, onde $\Delta d_l(\omega, P)$ para oclusores próximos equivale a intensidade da oclusão dentro de conjuntos de linhas densos, enquanto para oclusores distantes descreve o decaimento de sombras e a influência do espaço vazio entre conjuntos. Define-se uma função de decaimento $\delta(l)$ que atenua as diferenças de profundidade de acordo com que tipo de estrutura que está sendo amostrada, evitando assim que oclusores com grande diferença de profundidade se ocludam muito. Para definir um valor mínimo para a diferença de profundidade, necessária para aplicar a atenuação, utiliza-se um valor limiar $\delta_0 = 0.0001$. A seguir define-se a atenuação por profundidade:

$$g_l^{depth}(\omega, P) = \begin{cases} 0, & \text{se } \Delta d_l(\omega, P) > \delta(l) \\ 1, & \text{se } \Delta d_l(\omega, P) < \delta(0) \\ 1 - h(\frac{d_l(\omega,P)-\delta(0)}{\delta(l)-\delta(0)}), & \text{caso contrário.} \end{cases}$$

O valor de g_l^{depth} é 1 para oclusores próximos, os quais a diferença de profundidade é menor que o valor limiar, enfatizando assim as estruturas locais nas vizinhanças.Este valor é 0 para suprimir a oclusão se a diferença de profundidade exceder o máximo definido pela função de decaimento. Caso esteja entre $\delta(l)$ e $\delta(0)$ usa-se o Polinômio de Hermite:

$$h(x) = 3x^2 - 2x^3, \forall x \in [0, 1] : h(x) \in [0, 1] \tag{2.14}$$

Portanto, para oclusores próximos g_l^{depth} enfatiza linhas na vizinhança direta do ponto P, destacando estruturas locais em conjuntos de linhas.

Incorporam-se todas as fontes de luz da cena no cálculo para atenuar o efeito da oclusão de ambiente nas áreas iluminadas, evitando assim que áreas densas e muito ocludidas fiquem muito escuras.

A oclusão baseada em iluminação pode ser definida por:

$$g_l^{light}(\omega, P) = 1 - min(L_l(\omega, P), 1). \tag{2.15}$$

sendo $L_l(\omega, P)$ definido como somatório das intensidade de iluminação das fontes de luz da cena.

Combinando a atenuação de iluminação e de profundidade, a função de peso é capaz de enfatizar os detalhes locais sem exagerar no efeito de sombreamento enfatizando assim as relações espaciais.

2.4 Oclusão ambiente em espaço do volume

As técnicas conhecidas como oclusão ambiente em espaço do volume, são aquelas que utilizam uma representação baseada em *voxels* para estimar a geometria ao invés de somente a porção visível da cena pelo observador, de forma a se aproximar de um resultado fisicamente coerente.

Nas subseções a seguir são apresentados trabalhos que levam em conta a contribuição da luz em diversas direções e fazem o uso de *voxels* para discretizar a geometria buscando solucionar a oclusão ambiente.

2.4.1 *Ray-marching* em espaço do volume

Papaioannou et al. [17] propõem um método que visa produzir oclusão ambiente de aspecto global utilizando uma representação volumétrica para amostrar a visibilidade. Ao utilizar *ray-marching* em volumes pré-calculados ou dinamicamente gerados, a abordagem é independente da posição da câmera na cena e não necessita de uma representação da superfície nos cálculos de visibilidade.

A técnica *ray-marching* consiste no lançamento de raios a partir do ponto P em diversas direções. Esses raios são iterados a passos constantes em uma grade regular e a cada passo verifica-se a existência de geometria no *voxel* corrente. Caso o raio acerte a geometria, ou seja, o *voxel* corrente esteja preenchido, a iteração nesse raio é suspensa. A Figura 2.7 demonstra o algoritmo e a escolha de passos pequenos aproxima essa técnica da técnica de traçado de raios.

Utilizando a técnica de voxelização apresentada na Seção 2.2, podemos criar os volumes em pré-processamento ou dinamicamente a cada quadro.

2.4.2 Traçado de cones em *voxels*

O traçado de cones utilizado por Crassin et al.[4] busca aproximar os resultados de um pacote de raios partindo de um ponto, onde o eixo do cone é percorrido realizando acessos a uma estrutura hierárquica voxealizada em

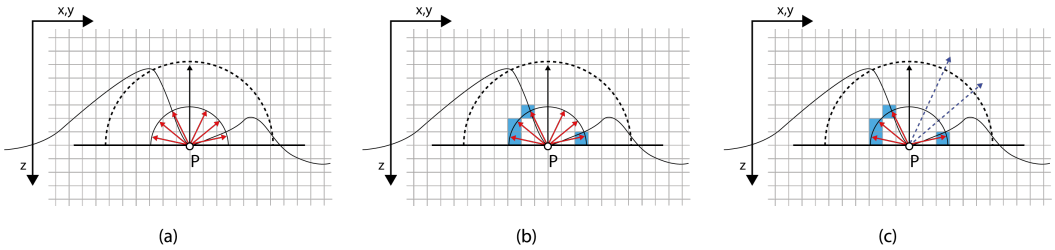


Figura 2.7 – Método de *ray-marching*. (a) Raios lançados a partir do ponto P para amostrar o volume. (b) Primeira iteração , os *voxels* marcados em azul foram atingidos interrompendo a iteração naqueles raios. (c) Estado final, os raios azuis ultrapassaram a distância do hemisfério sem atingir a geometria.

níveis diferentes de acordo com o raio do cone. Essa estrutura é mantida na forma de uma *octree* de *voxels* gerada a partir de malhas de triângulos na GPU. 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 dividido pelo número total de cones.

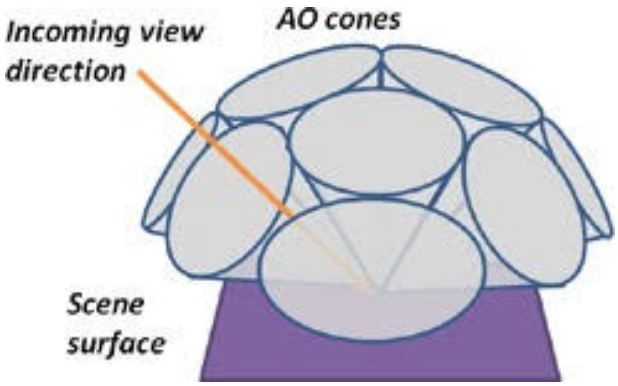


Figura 2.8 – Cone Tracing - lançamento de diversos cones no hemisfério em torno de um ponto. Retirado de [4]

O algoritmo proposto por Favera e Celes [5] utiliza uma estrutura de grade regular que discretiza a geometria em *voxels* facilitando o acesso as informações de geometria da cena, permitindo assim estimar o valor da oclusão ambiente ao redor de cada *pixel*. Pode-se estimar a oclusão ambiente em cada ponto através do lançamento de uma série de cones distribuídos no hemisfério de interesse. Cada cone representa um pacote de raios. O volume de cada cone é aproximado por um conjunto de esferas, usadas para determinar o volume ocluso. A Figura 2.9 ilustra o cálculo do volume ocluso em um dos cones.

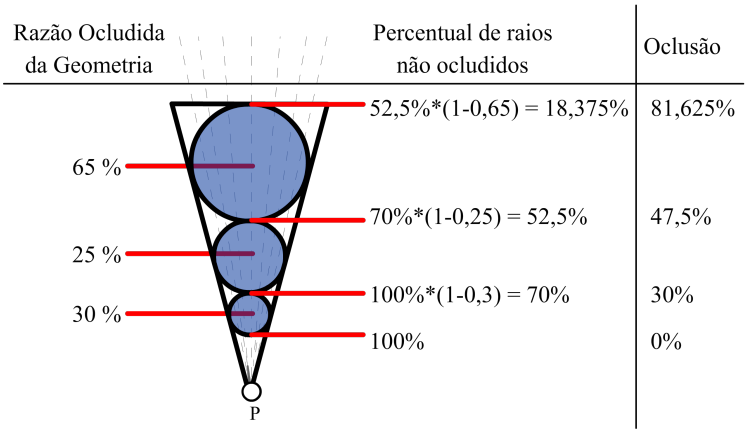


Figura 2.9 – Cálculo do volume ocluso aproximado por um conjunto de esferas, retirado de [5].

2.5 Considerações e proposta

A iluminação contribui de forma importante para a compreensão espacial de conjuntos de linhas, sendo o uso de tubos cilíndricos a base para muitas técnicas de renderização na literatura. As técnicas abordadas na Seção 2.1 são utilizadas em conjunto com o método proposto para o cálculo da iluminação final da cena.

As seções anteriores apresentam métodos que visam estimar o valor da oclusão para cada *pixel* visível na tela em tempo real. Os métodos apresentados na Seção 2.3 trabalham no espaço da imagem e obtêm resultados com eficiência pois independem da geometria da cena. Somente a utilização da informação presente na tela poderia produzir resultados incorretos e variáveis de acordo com o ponto de vista [17], sendo necessário o uso de estruturas de dados para complementar as informações. O trabalho *LineAO* [1] será utilizado como foco na comparação dos resultados, visto que apresenta bom desempenho e qualidade visual.

Papaioannou et al.[17] acabam com a dependência do ponto de vista utilizando o *ray-marching* em um volume; no entanto, o processo de *ray-marching* tem seu desempenho reduzido para produzir um resultado de qualidade, pois necessita de uma grande quantidade de raios e iterações.

Buscando um desempenho eficiente podemos referenciar a técnica proposta por Crassin et al. [4], que constrói uma estrutura de dados complexa capaz de tirar proveito da placa gráfica e lançar cones para amostrar a geome-

tria voxelizada em diversos níveis de uma *octree*. Favera e Celes [5] utilizam a mesma ideia de lançar uma série de cones, mas com uma estrutura de grade regular que discretiza a geometria em *voxels*, aproximando o cálculo da oclusão por esferas.

Técnicas para desenho de fios de cabelo [23, 24] também poderiam ser consideradas, mas devido à grande quantidade de simplificações para torná-las eficientes, elas não podem ser usadas em visualização científica.

Este trabalho tem o objetivo de desenvolver um método que aproxime a oclusão ambiente para linhas de forma a aumentar a percepção e compreensão espacial dos conjuntos de linhas. O uso de um método eficiente de voxelização em tempo real e a utilização de uma amostragem da geometria de baixo custo para calcular a integral de oclusão tornam o método proposto uma solução interessante para aplicações que busquem desempenho e qualidade em tempo real.

3
Método Proposto

O método desenvolvido neste trabalho foi inspirado nas técnicas dos trabalhos apresentados no capítulo anterior de Papaioannou et al.[17], Szirmay-Kalos et al.[16] e Favera e Celes [5]. A voxealização da geometria em uma grade regular foi baseada no trabalho de Papaioannou et al.[17], no qual se constrói uma representação voxelizada da cena em um buffer de textura. Modificou-se a técnica de obtenção do volume ocluso de uma esfera no espaço voxelizado proposto por Szirmay-Kalos et al.[16] para utilizar prismas que correspondem aos voxels no *depth-buffer*, da mesma forma que Favera e Celes [5] calculam o quanto do volume do cone está ocluso.

Esta dissertação propõe uma técnica que amostra o espaço voxelizado da cena ao redor de um ponto para avaliar a quantidade de geometria presente e obter a oclusão de cada *pixel* visível pelo observador. A amostragem do volume por prismas almeja substituir o custoso traçado de raios fazendo uso de operações baratas em máscaras de *bits*. O algoritmo proposto pode ser decomposto nas seguintes passadas:

- Obtenção das informações geométricas das linhas.
- Voxelização da geometria.
- Cálculo da oclusão.

A Figura 3.1 ilustra o *pipeline* do algoritmo:

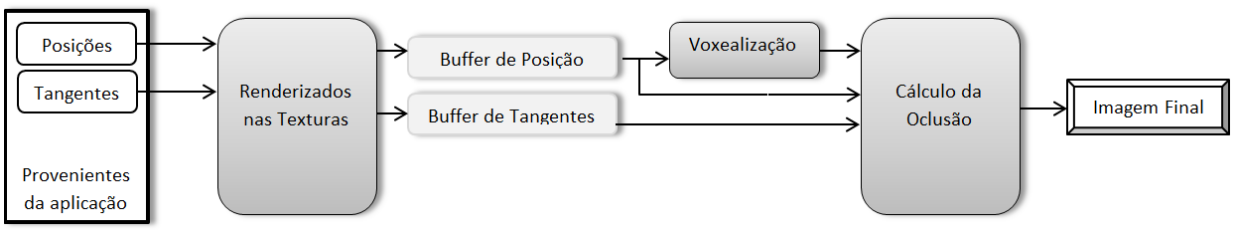


Figura 3.1 – *Pipeline* do algoritmo proposto neste trabalho.

3.1 Obtenção das informações geométricas

O método utiliza o espaço de tela para o cálculo da oclusão de ambiente, sendo assim dependente da posição corrente da câmera para realizar o cálculo do fator de oclusão para os *pixels* visíveis. Logo, são construídos *buffers* de textura contendo dados geométricos da cena para estender esse cálculo, de modo que este leve em conta toda cena renderizada.

Os *buffers* de geometria contém a posição e a tangente no espaço do olho de cada fragmento. Esses dados são utilizados diretamente na construção da estrutura de grid que representa a cena (voxealização) e no cálculo da oclusão.

3.2 Voxelização

Na segunda passada de *render* do método, utiliza-se as informações de geometria para a criação de uma estrutura de dados que reduz o custo computacional do cálculo da oclusão da passada subsequente. Para dar suporte a cenas dinâmicas, essa estrutura é construída a cada quadro renderizado.

A diferença para o algoritmo de voxelização original proposto por Eisemann e Décoret [3]reside em: ao invés de utilizar a operação lógica XOR para marcar o interior dos objetos da cena, fazemos o uso da operação lógica OR, porque não possuímos objetos sólidos, mas sim linhas. Utiliza-se a projeção ortogonal para a construção do grid criando-se então uma voxealização, onde somente estão ativos os *bits* que representam no espaço 3D a presença de uma geometria(Figura 3.2).

Utilizamos uma textura 2D com largura e comprimento iguais à do canvas e 128 *bits* de profundidade, divididos em 4 canais de cor *RBGA* (32 *bits* por canal). Cada *bit* de profundidade representa um *voxel* na direção z, ou seja, normalizando o espaço da cena entre o *near plane* e o *far plane* da câmera convertemos a distância do fragmento ($z_{distance}$) no índice do *bit* correspondente:

$$\begin{aligned} z_{norm} &= \frac{z_{distance} - z_{near}}{z_{far} - z_{near}} \\ z_{GridIndex} &= z_{norm} * 128.0 \end{aligned} \tag{3.1}$$

No entanto, analisando a voxealização proveniente da operação lógica OR com a máscara de *bits*, nota-se que esta não capta as linhas que são perpendiculares ao plano de projeção. Isto se deve ao fato que linhas na

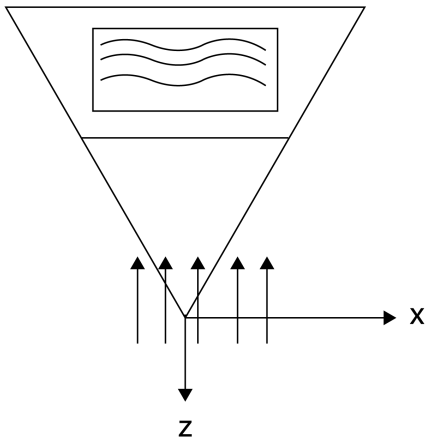


Figura 3.2 – Construção do grid de voxelização utilizando a projeção ortogonal.

direção do raio de projeção são rasterizadas em apenas um fragmento, o que não é suficiente para representar toda uma linha na voxealização.

Logo, propõe-se uma alteração no método de voxealização, onde agora será voxealizada cada linha de uma vez, sendo necessário o uso do *geometry shader* que terá como entrada dois vértices usando um *layout line_strip* para passar informações relativas aos extremos da linha para o *fragment shader*, e assim avaliar de forma correta os índices z que serão ligados na voxealização.

No *geometry shader* então são calculados para cada linha o valor de z_0 e z_1, que são os valores de profundidade no espaço do olho do começo e do fim da linha, d_x e d_y que são a distância em x e em y dos pontos do começo e do fim da linha. A Figura 3.3 apresenta esquema com esses valores.

$$\begin{aligned} x_i &= ((p_i.x/p_i.w + 1)/2.0) * screen\_width \\ y_i &= ((p_i.y/p_i.w + 1)/2.0) * screen\_height \\ dx &= abs(x_1 - x_0) \\ dy &= abs(y_1 - y_0) \\ z_i &= abs(\check{z}_i * 128.0) \end{aligned} \tag{3.2}$$

sendo i igual a 0 ou 1, referente ao ponto inicial ou final da linha, p o próprio ponto no espaço do olho e $\check{z}_i$ o valor da profundidade no espaço do olho normalizado pela Equação 3.1.

Essas informações são passadas para o *fragment shader* e usadas no cálculo dos limites de profundidade da linha, ligando na textura, através da

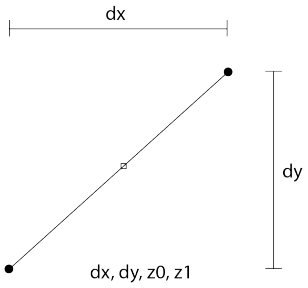


Figura 3.3 – Variáveis calculadas no *geometry shader* que serão utilizadas no *fragment shader* na voxealização.

operação binária OR, todos os *pixels* nesse intervalo. O cálculo dos limites é dado a seguir:

$$\begin{aligned} value &= z_{EyeNorm} * 128.0 \\ \Delta &= abs((z_1 - z_0)/(max(dx, dy) + \epsilon)); \\ z_{start} &= (value - (\Delta/2.0f)); \\ z_{end} &= (z_{start} + \Delta); \end{aligned} \tag{3.3}$$

onde ϵ é um valor pequeno (0.001) para evitar a divisão por zero, $z_{EyeNorm}$ o valor da profundidade no espaço do olho normalizado pela Equação 3.1 e z_{start} e z_{end} restritos a estarem entre z_0 e z_1.

3.3 Cálculo da oclusão

O cálculo da oclusão pode ser avaliado como o volume ocludido do hemisfério ao redor de um ponto, sendo computacionalmente avaliado através da simplificação da integral do hemisfério de interesse em uma Soma de Riemman, na qual o volume ocludido do hemisfério é sub-amostrado por um conjunto de prismas (Figura 3.4) e a contribuição de cada prisma é acumulada.

A integral da oclusão pode então ser aproximada computacionalmente por:

$$AO = \frac{\sum \text{ Volume ocludido dos prismas amostrais}}{\text{Volume total dos prismas amostrais}} \tag{3.4}$$

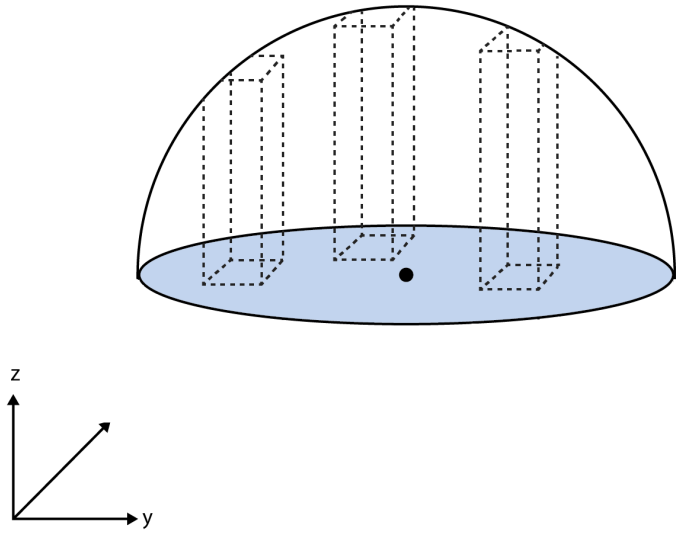


Figura 3.4 – Hemisfério pode ser simplificado por uma soma de prismas.

3.3.1 Orientação do hemisfério

A base do hemisfério de amostragem é tangente à linha, tendo sua orientação normal voltada para câmera o quanto possível. Logo, é necessário definir uma nova base ortogonal:

$$\hat{u} = \hat{tangente}_{eye} \tag{3.5}$$

$$\hat{v} = normalize(\hat{tangente}_{eye} \times \hat{posição}_{eye}) \tag{3.6}$$

$$\hat{w} = \hat{u} \times \hat{v} \tag{3.7}$$

3.3.2 Prismas amostrais

Para aproximar a integral de oclusão, são obtidos prismas através do lançamento de amostras na base circular do hemisfério. Essas amostras, depois de geradas, são definidas no plano de projeção e influenciam diretamente na forma que é captada a presença de oclusores que contribuirão para o escurecimento da cor final de cada ponto. Por isso, busca-se escolher amostras bem distribuídas em um círculo de raio unitário que representa a base do hemisfério.

Na proposta, foram escolhidas amostras de *Poisson* porque esta é uma distribuição de probabilidade de variável aleatória discreta e independente. Com o objetivo de evitar a formação de faixas devido à amostragem regular foram utilizados vários conjuntos distintos gerados em pré-processamento e

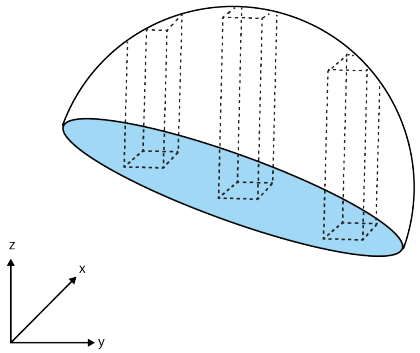


Figura 3.5 – Hemisfério fora do plano normal.

armazenados em uma textura. Variando os conjuntos de amostragem evita-se a formação de um padrão visual de amostragem na imagem.

Com o objetivo de ressaltar os detalhes entre linhas, foi utilizado uma amostragem com uma concentração maior de amostras no centro do hemisfério circular, geradas a partir da soma de três conjuntos de amostras de *Poisson* na qual o primeiro é gerado com raio 0.2, o segundo com o raio 0.5 e o último com raio 1.0. As amostras são armazenadas em uma textura 2D. A Figura 3.6 apresenta um exemplo de conjunto de pontos de *Poisson* gerados em um circulo unitário de intervalos $[-1, 1]$.

A quantidade de amostras usadas influencia diretamente o desempenho do algoritmo e a qualidade final da imagem gerada. No algoritmo proposto são usadas 150 amostras por ponto, pois apresentaram uma relação *qualidade* × *desempenho* superior a outras configurações. A Seção 4.1.2 apresenta uma avaliação do desempenho variando o número de amostras utilizadas.

Pode-se computar a oclusão de ambiente em um ponto P se considerarmos o volume do hemisfério ao redor do ponto como uma soma de prismas no espaço do olho. A posição de cada prisma é determinada por uma amostra e os prismas são alinhados no eixo z. Cada prisma deve corresponder à porção interna do hemisfério e pode ser subdividido em seções que equivalem a um *voxel* na textura de voxealização gerada na passada anterior.

O método para o cálculo do intervalo do prisma dado uma amostra é detalhado a seguir (Figura 3.7):

– Escolhe-se um ponto c, que é uma amostra na base do hemisfério, utilizando uma amostra de *Poisson* s e um raio de amostragem R:

$$\overrightarrow{c} = \overrightarrow{p} + R \begin{bmatrix} s_x \\ s_y \\ 0 \end{bmatrix} \tag{3.8}$$

– Encontra-se então $\overrightarrow{c_0}$ e $\overrightarrow{c_1}$ através do cálculo da interseção raio-esfera, onde a origem do raio é a câmera:

$$at^2 + bt + c = 0 \tag{3.9}$$

t é a projeção de $\overrightarrow{c}$ no near plane e define-se $\hat{k} = \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix}$.

$$\begin{aligned} a &= \hat{k} \cdot \hat{k} \\ b &= 0 \\ c &= (\overrightarrow{c} - \overrightarrow{p}) \cdot (\overrightarrow{c} - \overrightarrow{p}) - R^2 \end{aligned} \tag{3.10}$$

onde temos duas soluções analíticas:

$$t = \pm\sqrt{R^2 - d^2} \tag{3.11}$$

dado que $d = R * \sqrt{(s_x * s_x) + (s_y * s_y)}$.

os dois pontos de interseção são:

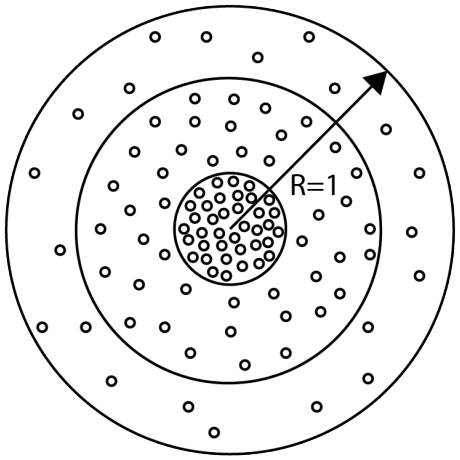


Figura 3.6 – Conjunto de amostras com maior concentração no interior do circulo de raio unitário.

$$\begin{aligned} \overrightarrow{c_0} &= \overrightarrow{c} - t\hat{k} \\ \overrightarrow{c_1} &= \overrightarrow{c} + t\hat{k} \end{aligned} \tag{3.12}$$

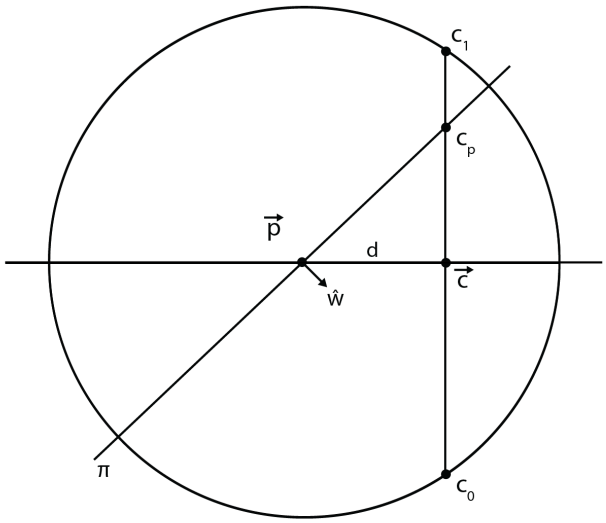


Figura 3.7 – Cálculo dos limites do prisma.

– Calcula-se também a interseção entre o plano dado pela base cartesiana e o raio. Onde as equações do plano e do raio podem ser dadas por:

$$raio : \overrightarrow{c} + t\hat{k} \tag{3.13}$$

$$plano : \hat{w}\,\overrightarrow{x} + d \tag{3.14}$$

$$d = -\hat{w}\,\overrightarrow{p} \tag{3.15}$$

$$f = \frac{-(\overrightarrow{c} \cdot \hat{w} + d)}{\hat{k} \cdot \hat{w}} \tag{3.16}$$

$$\overrightarrow{c_p} = \overrightarrow{c} - f\hat{k} \tag{3.17}$$

A Figura 3.8 ilustra os três casos possíveis para o cálculo dos limites dos prismas:

- Se $c_p.z > c_0.z$, o prisma está fora do hemisfério avaliado e não é considerado no cálculo.
- Se $c_p.z < c_1.z$, o prisma avaliado tem seus limites sendo c_0 à c_1.
- Senão, o prisma avaliado tem como limites limites sendo c_0 à c_p ($c_1 = c_p$).

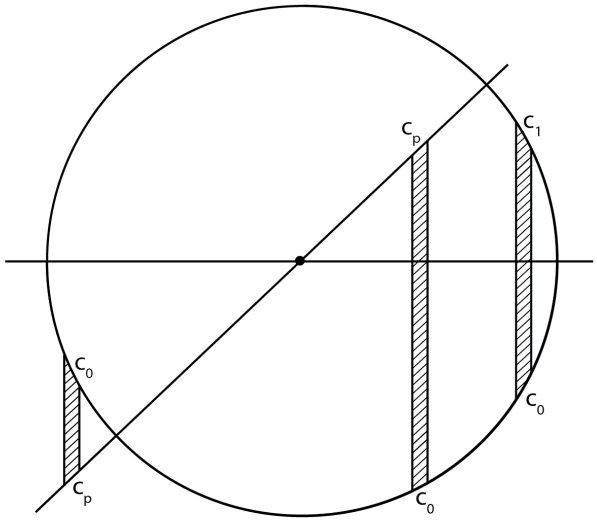


Figura 3.8 – Configurações possíveis para os prismas amostrais do hemisfério avaliado.

– Calcula-se então a posição do ponto no espaço de clipping, que é o simples cálculo do ponto multiplicado pela matriz de projeção, já que o ponto estava em coordenadas do espaço do olho:

$$posClip = ProjectionMatrix * \begin{pmatrix} c.x \\ c.y \\ c.z \\ 1.0 \end{pmatrix} \tag{3.18}$$

em seguida encontra-se a posição do *textel* que se irá acessar na textura de voxelização:

$$posVoxel = \begin{pmatrix} \frac{(\frac{posClip.x}{posClip.w})}{2.0} + 0.5 \\ \frac{(\frac{posClip.y}{posClip.w})}{2.0} + 0.5 \end{pmatrix} \tag{3.19}$$

– Normaliza-se os pontos c_0 e c_1 e multiplica-se por 128, por causa dos 4 canais de 32 bits da textura, para se achar os limites do prisma:

$$\begin{aligned} c\_normalizado_i &= \frac{-c_i.z - z_{near}}{z_{far} - z_{near}} \\ val_i &= c\_normalizado_i * 128 \end{aligned} \tag{3.20}$$

onde i é 0 ou 1. Basta dividir o val_α por 32 para encontrar o canal de cor da textura que está o limite e fazer a operação *mod* 32 para descobrir qual o *bit* daquele canal.

3.3.3 Volume Ocludido

O algoritmo proposto basea-se na idéia que pode-se calcular a oclusão em um ponto através da soma dos volumes que possuem geometria nos prismas amostrais dividido pela soma dos volumes de todos primas amostrais. Sendo assim, para cada prisma, tendo encontrado seus limites inferior e superior, basta agora contabilizar quantos bits estão ativos naquele intervalo, indicando quantos *voxels* possuem geometria e dividir pelo número total de bits do intervalo. Somando a contribuição de cada prisma e dividindo pelo número de prismas temos a oclusão no ponto P.

A linguagem de *Shader GLSL* disponibiliza funções para operações com *bits*. A função *bitfieldextract* recebe um intervalo de um número inteiro e retorna os bits daquele intervalo. A função *bitcount* recebe um intervalo e retorna o número de *bits* acesos no mesmo. Logo, fornecendo os limites para a função *bitfieldextract* e utilizando sua saída na função *bitcount* conseguimos contabilizar o número de *voxels* que possuem geometria no prisma. A Figura 3.9 ilustra esse processo.

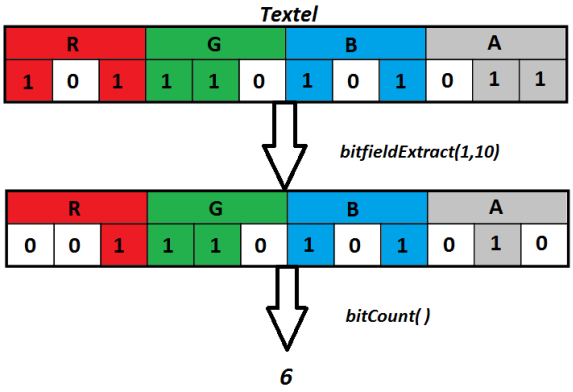


Figura 3.9 – Uso das funções *bitfieldextract* e *bitcout*.

Dividindo o valor obtido pelo número total de bits no intervalo, temos a fração de quanto da geometria está ocludida.

Para calcular o volume ocluso no hemisfério basta somar a fração da geometria oclusa de cada prisma e dividir pelo número total de prismas.

3.3.4 Cálculo da atenuação

Após calcular corretamente os limites do prisma, podemos determinar o número de *bits* ligados entre eles. Entretanto, cada prisma pode estar fora do plano normal, sendo necessário a interpolação do ângulo entre $\overrightarrow{c_i}$ (que é relativo à um bit) e o plano normal.

Temos que $\overrightarrow{c_i}$ pode ser dado por uma interpolação de $\overrightarrow{c_0}$ e $\overrightarrow{c_1}$:

$$\overrightarrow{c_i} = \overrightarrow{c_0}(1 - \frac{i}{n-1}) + \overrightarrow{c_1}(\frac{i}{n-1}) \tag{3.21}$$

A Figura 3.10 ilustra o processo de interpolação, onde temos $\overrightarrow{t}$ como sendo o vetor unitário tangente, α sendo o ângulo formado entre $\overrightarrow{t}$ e $\overrightarrow{c_0}$ e θ sendo o ângulo formado entre o plano normal à linha e $\overrightarrow{c_0}$.

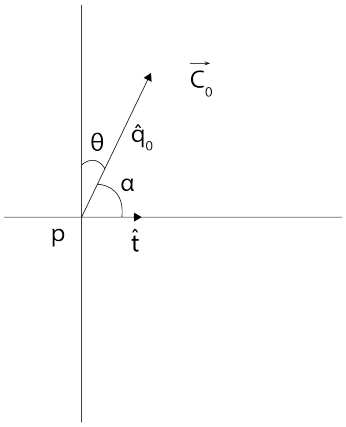


Figura 3.10 – Interpolação do ângulo entre c_i e o plano normal.

Temos que $\theta = \frac{\pi}{2} - \alpha$ e aplicando a regra do cosseno:

$$\cos\theta = \cos\frac{\pi}{2}\cos\alpha + \sin\frac{\pi}{2}\sin\alpha \tag{3.22}$$

usando o valor conhecido do seno e cosseno de $\frac{\pi}{2}$:

$$\cos\theta = \sin\alpha \rightarrow \sin\alpha = ||\hat{t} \times \hat{c_i}|| \tag{3.23}$$

Dessa forma podemos substituir o cálculo da luz incidente por:

$$L = \frac{\sum \cos\theta bits_i}{\sum \cos\theta total\_bits} \tag{3.24}$$

e podemos simplificar o cálculo de 3.21, por razões de desempenho, para utilizar a média aritmética do seno:

$$L = \frac{\sum \sin(\alpha_{average}) bits_i}{\sum \sin(\alpha) total\_bits} \tag{3.25}$$

Por fim, o resultado da oclusão deve levar em conta um peso de acordo com a distancia da amostra para o ponto P. Dessa forma amostras que estão mais próximas do ponto terão uma contribuição maior na cor final do que amostras que estão mais afastadas.

Escolheu-se utilizar como peso a distância média entre o ponto P e os limites c_0 e c_1 do prisma:

$$\begin{aligned} dist_{c_0} &= |c_0 - P| \\ dist_{c_1} &= |c_1 - P| \\ dm &= (dist_{c_1} + dist_{c_0})/2; \end{aligned} \tag{3.26}$$

fazendo que o cálculo final da luz incidente seja dado por:

$$L = \frac{\sum \sin(\alpha_{average}) bits_i}{\sum \sin(\alpha) \text{ total\_bits } dm^2} \tag{3.27}$$

3.3.5 Algoritmo

O algoritmo proposto é executado para todos os *pixels* visíveis, amostrando o espaço ao redor dos mesmos através de prismas e calculando o volume ocluso destes por meio de operações de *bits*, estimando assim a oclusão em cada ponto.

Cada aplicação possui requisitos diferentes e para que esse método possa ser configurado da maneira mais adequada algumas variáveis podem ser alteradas modificando a relação qualidade e desempenho. Podemos destacar as seguintes variáveis:

- Número de amostras
- Raio de influência das amostras
- Fator de oclusão

O valor da oclusão de um ponto depende diretamente da variável *número de amostras*. Após o cálculo da oclusão para cada amostra, calcula-se o valor da atenuação dessa amostra, atribuindo-se pesos menores para amostras mais distantes. Assim, a contribuição de cada amostra é estimada e acumulada bastando agora dividir pelo número total de amostras para obter-se a oclusão de cada fragmento.

O algoritmo proposto realiza a leitura dos valores das amostra de *Poisson* de uma textura, os quais ajudam na determinação da posição e direção de cada prisma. Se utilizarmos os mesmos valores para todos os *pixels*, notaria-se o surgimento de padrões visuais. Para diminuir este efeito, são gerados diversos conjuntos de amostras em pré-processamento e armazenados em uma textura que é enviada à placa gráfica. Quando os fragmentos são gerados, usa-se a sua posição no espaço da tela para determinar qual a distribuição será utilizada, eliminando esse problema.

O posicionamento de cada amostra, além de variar com a posição do fragmento, depende da variável *raio de influência*, o qual é responsável por

variar o raio do círculo que contem as amostras de *Poisson*. Ao utilizarmos raios menores estaremos favorecendo detalhes locais, enquanto aumentando o raio favoreceremos detalhes globais.

O *fator de oclusão* serve para dar um controle fino de ajuste ao usuário da aplicação sobre a intensidade da oclusão na imagem resultante.

A Tabela 3.1 apresenta o pseudocódigo do algoritmo (*fragment shader*) para o cálculo da oclusão.

Tabela 3.1 – Pseudocódigo do funcionamento do algoritmo.

```
position, tangent, depth, radius ← readInputData()
ao, weight ← 0
u, v, w ← getBase()
for i = 0, numSamples do
  sectionBits, totalBits, onBits ← 0
  currentSample ← texelFetch(PoissonSample)
  c ← position+radius∗vec3(currentSample.x, currentSample.y, 0)
  c0, c1 ← calcRaySphereIntersec()
  cp ← calcRayPlaneIntersec()
  c1 ← testPrismLimits()
  q0, q1, bit0, bit1 ← calcSectionsAndBits()
  posTex ← calcPositionTexture(c)
  voxelData ← texture(VoxelTex)
  medSin ← calcMedSin()
  dm ← calcMedDist()
  if q0! = q1 then
    sectionBits ← bitfieldExtract(voxelData[q0], bit0, 32 − bit0)
    OnBits+ = bitCount(sectionBits)
    totalBits = +32 − bit0
    if q1 − q0 == 2 then
      sec ← q0 + 1
      sectionBits ← bitfieldExtract(voxelData[sec], 0, 32)
      OnBits+ = bitCount(sectionBits)
      totalBits = +32
    else if q1 − q0 == 3 then
      sec ← q0 + 1
      sectionBits ← bitfieldExtract(voxelData[sec], 0, 32)
      OnBits+ = bitCount(sectionBits)
      sec + +
      sectionBits ← bitfieldExtract(voxelData[sec], 0, 32)
      OnBits+ = bitCount(sectionBits)
      totalBits = +64
    end if
    sectionBits ← bitfieldExtract(voxelData[q1], 0, bit1 + 1)
    OnBits+ = bitCount(sectionBits)
    totalBits = +bit1 + 1
  else
    sectionBits ← bitfieldExtract(voxelData[q1], bit0, (bit1 −
    bit1 + 1))
    OnBits+ = bitCount(sectionBits)
    totalBits = +(bit1 − bit1 + 1)
  end if
  ao+ = (onBits ∗ medSin)/(totalBits ∗ sinAlpha ∗ dm ∗ dm)
  weight+ = 1.0
end for
ao ← ao/weight
finalAo ← clamp(ao ∗ AO_Factor, 0.0, 1.0)
```

4
Resultados

O método proposto foi implementado utilizando a linguagem de programação *C++*, usando a biblioteca *OpenGL* e a linguagem de *Shader GLSL*. Foram realizados diversos testes visando analisar a qualidade dos resultados obtidos e avaliar o desempenho do método.

Todos os testes foram efetuados em um processador *Intel i5* de *3.1 GHz* com *8Gb* de memória *RAM* utilizando uma placa de vídeo *Nvidia GeForce GTX 560Ti* com 1024MB de memória de vídeo. Entretanto, o desempenho do algoritmo não depende da *CPU* nem da memória, pois ele é executado unicamente na *GPU*. As medições de tempo presentes nas próximas seções são dos tempos médios das execuções dos algoritmos.

Os resultados alcançados são discutidos nas seções a seguir: uma análise do desempenho de cada etapa do algoritmo e o impacto na variação dos parâmetros deste são analisados na Seção 4.1. A Seção 4.2 analisa o uso de mais de um *buffer* de textura para a voxealização da cena. A Seção 4.3 compara o efeito produzido pela oclusão ambiente juntamente com a utilização de iluminação difusa. A Seção 4.4 compara o método proposto com outro algoritmo considerado como o estado da arte que foi implementado durante o desenvolvimento deste trabalho. Uma comparação com a ferramenta *Optix* desenvolvida pela *Nvidia* com uma aplicação de oclusão ambiente no espaço do mundo é feita na Seção 4.5. Finalmente, é discutido na Seção 4.6 o comportamento do algoritmo com a variação da complexidade da geometria da cena e da resolução da tela.

4.1
Análise de Desempenho

Para analisar o desempenho do algoritmo proposto foram utilizados modelos com diferentes números de linhas e diferentes orientações destas. A Tabela 4.1 apresenta uma descrição dos modelos escolhidos e serve como referência para as seções seguintes.

As linhas são carregadas no programa através da leitura de um arquivo de texto contendo o número de linhas, o número de vértices por linha e os

Tabela 4.1 – Modelos utilizados na análise do algoritmo proposto.

Modelo	Vértices	Número de linhas
Dipolo	804.780	900
Pom-Pom	1.502.088	1330
Espiral	2.712.880	1330
Reservatório 2000	664.616	1993
Reservatório 5000	1.599.928	4989
Reservatório 20000	6.485.554	19934

vértices de cada linha. Através desses dados são calculadas as tangentes de cada vértice de cada linha. As tangentes e os vértices de todas as linhas são colocados em um único *VBO - Vertex Buffer Object* que é alocado na memória da placa gráfica. Para desenhar as linhas utiliza-se apenas uma chamada da função *glDrawElements* pois é utilizado o modo *GL_PRIMITIVE_RESTART*, o qual para um conjunto de *strip* de linhas coloca-se um identificador (*primitive restart index*) entre duas linhas consecutivas e a cada vez que o comando de desenho encontrar esse identificador ele começa o desenho de um novo *strip*. A Figura 4.1 ilustra esse processo:

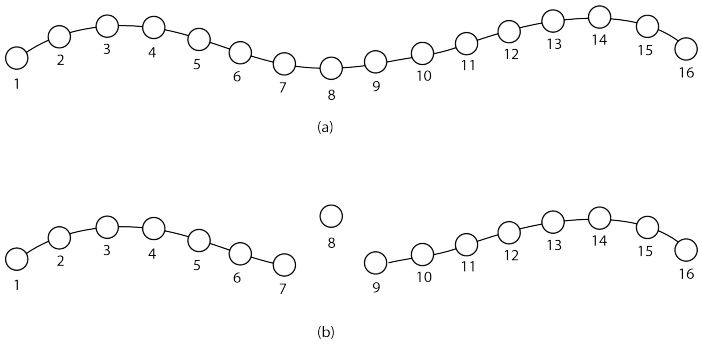


Figura 4.1 – O desenho de uma *Line Strip* (a) sem *primitive restart* (b) com *primitive restart*.

4.1.1 Custo de cada passada do algoritmo

O algoritmo proposto é composto por três passadas distintas: a construção dos *buffers* de geometria, a voxelização do volume da cena e o cálculo da oclusão. A Tabela 4.2 apresenta a medida de tempo que cada etapa necessita para a sua execução do modelo Reservatório 2000 utilizando uma resolução de tela de 1024×768 e 150 amostras para avaliar a oclusão. A coluna da direita

da tabela apresenta o percentual do quanto cada passada consome do tempo total de execução do algoritmo.

Tabela 4.2 – Resultados de tempo para a execução de cada uma das passadas.

Etapa	Tempo (ms)	Percentual
Obtenção dos dados de geometria	0.825	5 %
Voxelização	1.913	12 %
Cálculo da Oclusão	13.251	83 %
Total	9.275	100 %

Analisando a tabela verifica-se que a etapa do cálculo da oclusão consome quase todo o tempo de execução de um quadro. Como as etapas anteriores do cálculo da oclusão dependem da geometria nota-se que o algoritmo exibe um grande esforço por *pixel* mas ainda apresenta uma dependência da geometria, como pode ser visto na Figura 4.2 do *profiler Nsight da Nvidia*.

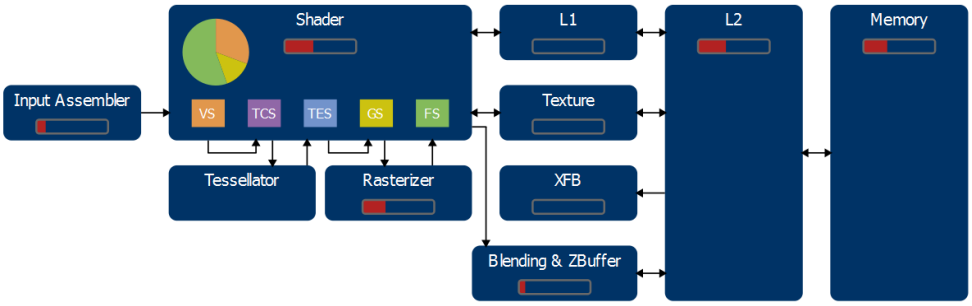


Figura 4.2 – Captura da tela do *profiler* mostrando que no geral o algoritmo tem um maior esforço do *Fragment Shader*, embora o uso do *Vertex Shader* representa mais de 25% do esforço total.

4.1.2 Influência dos parâmetros

Algumas variáveis podem ser alteradas no algoritmo para modificar a qualidade da imagem final gerada. O raio de amostragem (isto é, o raio do hemisfério) foi fixado em 13% do tamanho da cena, pois este se mostrou suficiente e adequado para a captura de detalhes tanto perto quanto distantes dos modelos, podendo ser variado de forma que se adapte melhor em outra aplicação, já que a partir de testes verificou-se que não afetar de forma significativa o desempenho do algoritmo.

O número de amostras por *pixel* é um parâmetro que influencia diretamente na qualidade do resultado gerado e pode ser variado dinamicamente de forma a alcançar a melhor relação de custo e benefício, pois esse afeta diretamente o desempenho, e à medida que o valor cresce, pode chegar a eliminar a

interatividade do algoritmo. As Tabelas 4.3 a 4.8 a seguir demostram como a variação do número de amostras afetam o desempenho.

Tabela 4.3 – Resultados para o modelo Dipolo com diferentes números de amostras.

Número de Amostras	Tempo (ms)	FPS
8	5.64	176.1
16	6.60	151.5
32	8.38	119.1
64	12.05	82.8
96	15.61	64.1
128	19.35	51.7
150	21.83	45.8
256	34.51	29.0

Tabela 4.4 – Resultados para o modelo Pom-Pom com diferentes números de amostras.

Número de Amostras	Tempo (ms)	FPS
8	8.76	114.2
16	9.80	102.0
32	11.89	84.1
64	16.05	62.3
96	20.08	49.8
128	24.31	41.1
150	27.20	36.8
256	41.44	24.1

Analisando esses dados, nota-se que à medida que o número de amostras aumenta o desempenho decai; contudo, com a evolução do *hardware* a capacidade de processamento das placas gráficas tende a aumentar mantendo assim um bom nível de interatividade e qualidade da imagem. Ainda assim, a aproximação proposta nesta dissertação possui a capacidade de gerar resultados próximos do cálculo fisicamente correto, visto que o aumento do número de amostras gera uma convergência do resultado.

A partir desta análise optou-se pela utilização da configuração de 150 amostras por *pixel* nos testes subsequentes devido à sua boa relação entre qualidade e desempenho.

4.2 Uso de múltiplas texturas para a voxealização

Uma restrição do método proposto se encontra na resolução da dimensão do eixo z da grade usada na voxealização. Na implementação desse trabalho ela é de 128 *voxels* devido ao número máximo de *bits* por *texel*. Esta restrição faz com que detalhes pequenos possam sejam perdidos. A resolução da grade

Tabela 4.5 – Resultados para o modelo Espiral com diferentes números de amostras.

Número de Amostras	Tempo (ms)	FPS
8	13.88	72.1
16	15.06	66.5
32	17.11	58.4
64	21.45	46.6
96	25.64	39.0
128	30.00	33.3
150	32.98	30.3
256	46.70	21.4

Tabela 4.6 – Resultados para o modelo Reservatório 2000 com diferentes números de amostras.

Número de Amostras	Tempo (ms)	FPS
8	4.96	201.5
16	5.76	174.1
32	7.34	136.2
64	10.52	95.1
96	13.74	72.8
128	17.01	58.8
150	19.25	52.0
256	30.23	33.1

pode ser aumentada ao se utilizar mais texturas para construí-la. Entretanto, isto pode impactar no desempenho geral da aplicação, havendo uma maior necessidade de tempo para construir a voxealização e mais acessos à textura na etapa do cálculo da oclusão;

Foi então investigado o uso de um segundo *buffer* para aumentar a resolução da grade. Esta agora passando a ter 256 *voxels* por *texel*. As Figuras 4.3 e 4.4 apresentam poses diferentes do modelo Reservatório 5000 e do modelo Espiral para o algoritmo proposto usando um e dois *buffers de textura*.

Esses modelos foram escolhidos porque foram os que apresentaram uma melhora na qualidade visual do resultado obtido. Pode-se notar uma melhor definição no conjunto de linhas do modelo Reservatório 5000 nas regiões mais densas do modelo, onde as linhas estão muito próximas. Já no modelo Espiral o ganho visual é muito sutil, podendo ser observado no centro da espiral.

A partir dos resultados obtidos na Tabela 4.9 pode-se perceber que o desempenho do algoritmo com *buffer* duplo foi um pouco inferior, sendo necessário uma melhor avaliação de quando seu uso gera uma melhora qualidade visual.

Tabela 4.7 – Resultados para o modelo Reservatório 5000 com diferentes números de amostras.

Número de Amostras	Tempo (ms)	FPS
8	8.97	111.6
16	9.80	102.2
32	11.46	87.6
64	14.77	67.7
96	18.07	55.3
128	21.42	46.7
150	23.69	42.2
256	34.80	28.7

Tabela 4.8 – Resultados para o modelo Reservatório 20000 com diferentes números de amostras.

Número de Amostras	Tempo (ms)	FPS
8	29.27	34.2
16	30.08	33.2
32	31.68	31.6
64	34.92	28.6
96	37.94	26.4
128	41.20	24.3
150	43.44	23.0
256	54.26	18.4

Tabela 4.9 – Comparação entre a média dos os tempos de execução obtidos para as poses apresentadas utilizando o método proposto com *buffer* simples e *buffer* duplo para a voxealização da cena.

Modelo	*Buffer* Simples (FPS)	*Buffer* Duplo (FPS)
Reservatório 5000	42.2	39.0
Espiral	30.3	27.2

4.3 Iluminação difusa

A oclusão ambiente tem como objetivo melhorar a qualidade da iluminação em ambientes tridimensionais, aumentando a percepção de detalhes da geometria da cena. Para ilustrar este ganho de qualidade foi comparado uma cena iluminada usando iluminação ambiente constante com outra usando iluminação ambiente calculada pelo método proposto. Ambas as cenas utilizam a iluminação difusa proposta por Mallo et al. [2]. A Figura 4.5 ilustra essa comparação utilizando o modelo Reservatório 5000.

Pode-se notar o aumento na percepção da forma e dos detalhes da cena, principalmente nas regiões não atingidas diretamente pela iluminação difusa. Além disso, a posição das linhas se torna mais evidente devido ao escurecimento

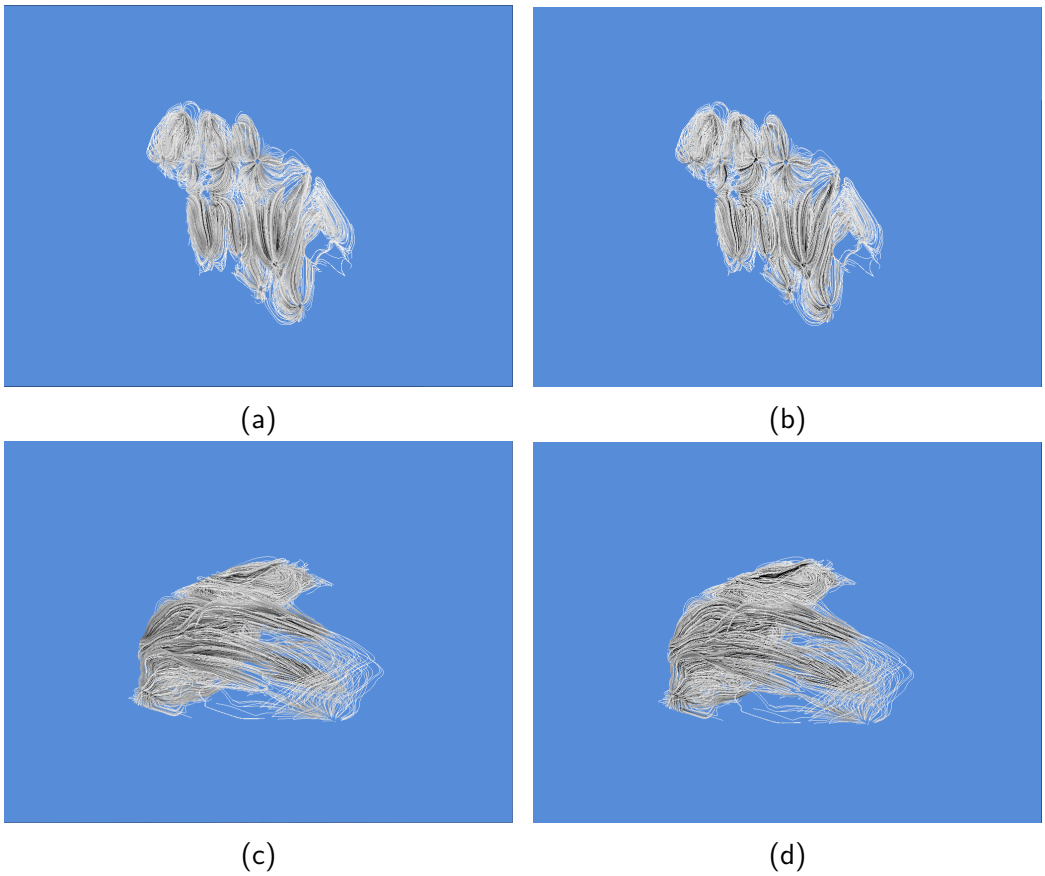


Figura 4.3 – Comparação do modelo Reservatório 5000 usando 1 ou 2 buffers para a voxealização. As Figuras 4.3a e 4.3a apresentam poses que utilizam apenas um buffer na voxealização da cena, enquanto as Figuras 4.3b e 4.3d utilizam 2 buffers para a voxealização.

das regiões em que há um conjuntos de linhas.

4.4 Comparação entre algoritmos

Os resultados a seguir apresentam imagens e o tempo de execução do algoritmo proposto e do algoritmo considerado como estado da arte para a renderização de linhas utilizando oclusão de ambiente. Este foi implementado juntamente com o método proposto nessa dissertação e corresponde a implementação LineAO por proposta Eichelbaum et al. [1], o qual faz o uso de uma amostragem no espaço de tela baseada em níveis de distância e utiliza uma função de visibilidade associada a uma função de peso para avaliar a contribuição de cada amostra.

A implementação constitui-se em adaptar o mesmo programa do algoritmo proposto para utilizar o código do *fragment shader* disponibilizado no site do artigo LineAO [1]. Para isso, foi necessário fazer modificações dos dados armazenados na placa gráfica. Criou-se uma nova textura de ruido branco

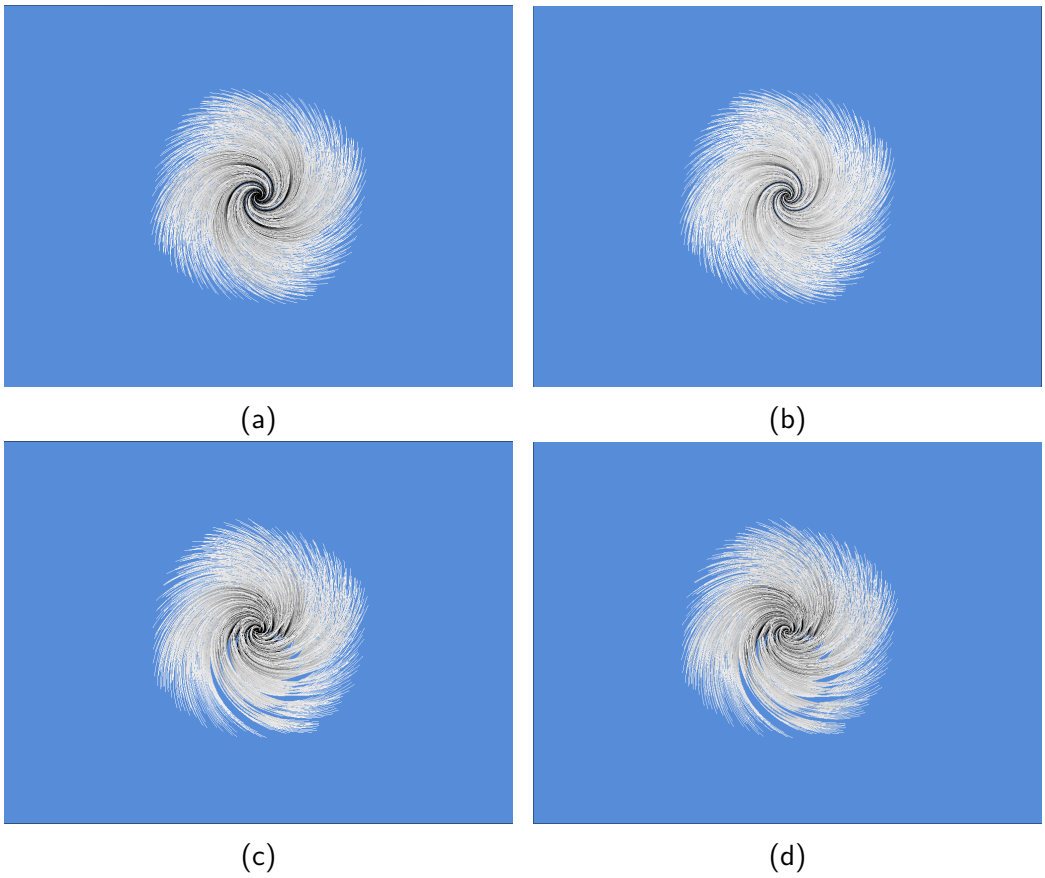


Figura 4.4 – Comparação do modelo Espiral usando 1 ou 2 buffers para a voxealização. As Figuras 4.4a e 4.4c apresentam poses que utilizam apenas um buffer na voxealização da cena, enquanto as Figuras 4.4b e 4.4d utilizam 2 buffers para a voxealização.

e calculou-se as tangentes da linha da mesma forma que é proposto no artigo.

O algoritmo LineAO [1] leva em consideração em seus cálculo as contribuições da iluminação difusa e especular. Portanto, para efeitos de comparação, foram utilizados junto com o método proposto os cálculos da iluminação difusa proposta por Mallo et al. [2] e da iluminação especular do trabalho de Banks [8].

A Figura 4.6 apresenta poses diferentes do modelo Reservatório 5000 para cada um dos métodos. A Tabela 4.10 apresenta tanto para o algoritmo proposto quanto para o método proposto a quantidade de quadros por segundo que foi medido para cada pose de câmera.

A partir dos resultados obtidos para o modelo Reservatório 5000 pode-se perceber que o algoritmo proposto apresenta desempenho um pouco superior ao LineAO [1]. A oclusão nas bordas dos modelos não é estimada corretamente para o LineAO [1], tornando as linhas de fora escuras. Por outro lado, a percepção das linhas, quando estão em conjuntos densos, está mais nítida nas imagens geradas pelo algoritmo LineAO [1].

Já a Figura 4.7 apresenta poses diferentes do modelo Dipolo para cada

Tabela 4.10 – Comparação entre os tempos obtidos utilizando o método proposto e o algoritmo LineAO [1] para o modelo Reservatório 5000.

Pose	LineAO (FPS)	Método proposto (FPS)
1	30.0	42.0
2	29.5	42.3
3	33.3	45.6

um dos métodos, tendo a Tabela 4.11 apresentando as medições de tempo para os dois algoritmos.

Tabela 4.11 – Comparação entre os tempos obtidos utilizando o método proposto e o algoritmo LineAO [1] para o modelo Dipolo.

Pose	LineAO (FPS)	Método proposto (FPS)
1	39.0	47.0
2	17.3	23.3

Pode-se notar que o algoritmo proposto escurece somente onde existe a oclusão, enquanto o LineAO [1] continua a apresentar um escurecimento incorreto das linhas mais externas.

Por último, a Figura 4.8 apresenta poses diferentes do modelo Espiral para os dois métodos com a Tabela 4.12 apresentando os dados coletados para os dois algoritmos.

Tabela 4.12 – Comparação entre os tempos obtidos utilizando o método proposto e o algoritmo LineAO [1] para o Espiral.

Pose	LineAO	Método proposto (FPS)
1	25.5	30.8
2	25.4	30.8
3	25.7	31.0
4	27.5	31.6

Ambos algoritmos apresentaram um escurecimento das áreas que são esperadas que se haja oclusão, entretanto o LineAO tenha o problema dos escurecimento das linhas mais externas, mas apresenta uma maior nitidez dos conjuntos de linhas.

4.5 Comparação com aplicativo de oclusão em espaço de mundo da *Nvidia*

Com o objetivo de analisar o resultado gráfico do algoritmo proposto foi utilizado o aplicativo de traçado de raios Optix [25] desenvolvida pela *Nvidia* que realiza o cálculo da oclusão ambiente em espaço de tela. A aplicação faz parte dos exemplos do *SDK* de *CUDA* disponíveis pela *Nvidia* [26]. Este

software utiliza *CUDA* sendo executado então na GPU e produz resultados de grande qualidade mas não em tempo real. Nenhum dos parâmetros do algoritmo implementado ou informação sobre detalhes da execução do mesmo (tal como número de raios, raio do hemisfério, uso de *multi-sampling*, etc.) é disponibilizado, apenas pode-se obter a posição corrente da câmera, o que facilita na comparação das poses de cada cena.

Para realizar a comparação foi necessário transformar as linhas em malhas de cilindros, pois a aplicação somente aceitava como entrada modelos no formato *Wavefront OBJ*. Este formato é bastante simples e serve para representar uma geometria 3D fornecendo as posições de cada vértice, as normais dos mesmos, coordenadas de textura e uma lista de vertices que forma as faces dos triângulos.

A resolução padrão das imagens utilizadas nos testes foi escolhida como 640x480 para facilitar a comparação com esta referência. O posicionamento de câmera foi aproximado e a comparação somente pôde ser feita utilizando até 1000 linhas, pois a aplicação não permite o carregamento modelos muito grandes.

A Figura 4.9 apresenta imagens comparando algumas poses de câmera do modelo Reservatório com 1000 linhas entre a aplicação em espaço de tela da *Nvidia* e o algoritmo proposto nesta dissertação.

Já a Figura 4.10 apresenta poses diferentes do modelo Dipolo para cada um dos métodos.

Por último, a Figura 4.11 apresenta poses diferentes do modelo Espiral para os dois métodos.

Os resultados apresentados nas figuras anteriores permitem perceber que as imagens produzidas pelo algoritmo proposto tendem a se aproximar do Optix à medida que temos um número de amostras que façam o método convergir. Como o resultado do programa da *Nvidia* é usado como *ground truth* em termos de áreas escurecidas, podemos dizer que o algorítimo proposto é fisicamente correto. Variando a quantidade de amostras podemos notar que algumas cenas chegam a produzir resultados muito próximos mesmo com uma quantidade menor de amostras, mostrando assim que o método proposto nesta dissertação apresenta qualidade comparável ao traçado de raios.

4.6 Escalabilidade

Buscando compreender o desempenho do método proposto, realizou-se a análise do algoritmo variando a complexidade da geometria da cena e variando o número de *pixels* através do uso de diferentes tamanhos de canvas.

A Tabela 4.13 apresenta os tempos medidos com a execução do mesmo modelo com complexidade geométrica variável devido à variação do numero de linhas em um canvas de resoluções , utilizando 150 amostras. A geometria de cada cena também é apresentada na tabela.

A Figura 4.12 apresenta o gráfico dos tempos obtidos. Nota-se uma tendência de um crescimento linear do tempo com o aumento da complexidade da geometria, o que é esperado, devido às etapas inicias dependerem da geometria.

A Tabela 4.14 apresenta as medições de tempo obtidas ao executar a cena Reservatório 5000 composta por 4989 linhas, 1.599.928 vértices em diversas resoluções de tela utilizando 150 amostras raio 0.13 e fator de oclusão 4.0.

A Figura 4.13 exibe um gráfico destes tempos, demonstrando que o algoritmo tem desempenho linearmente proporcional ao número de *pixels*, tal como esperado.

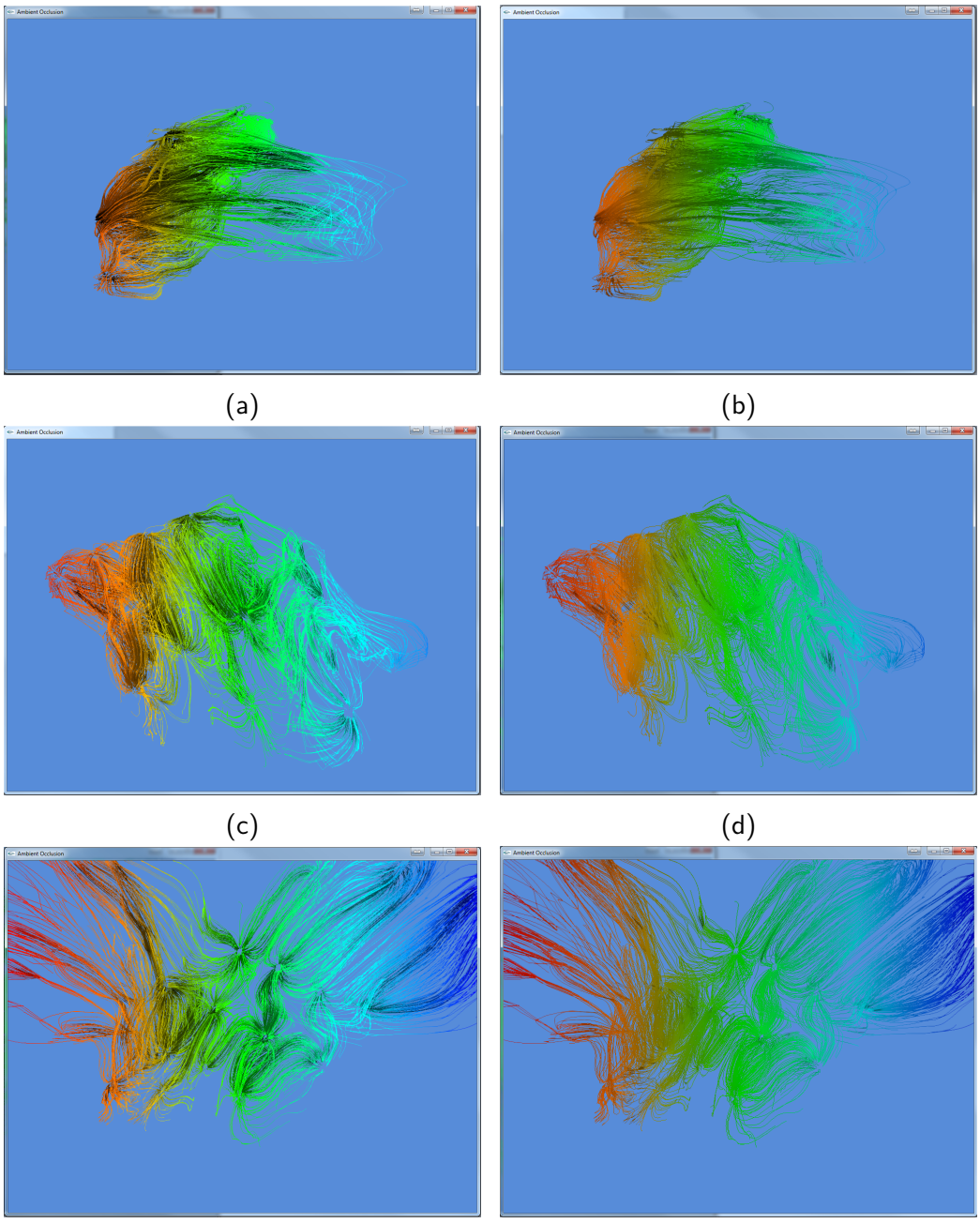


Figura 4.5 – Cena Reservatório 5000 com iluminação difusa. As Figuras 4.5a, 4.5c e 4.5e utilizam o fator de oclusão de ambiente calculado pelo algoritmo proposto, enquanto as Figuras 4.5b, 4.5d e 4.5f utilizam iluminação ambiente constante.

Tabela 4.13 – Resultados de tempo para a execução de cenas de diversas complexidades geométricas.

Modelo	Vértices	Número de linhas	Tempo (ms)	FPS
Reservatório 2000	664.616	1993	19.25	52.0
Reservatório 5000	1.599.928	4989	26.69	42.2
Reservatório 10000	3.085.503	9987	28.96	34.5
Reservatório 15000	4.366.567	14954	34.51	29.0
Reservatório 20000	6.485.554	19934	43.44	23.0

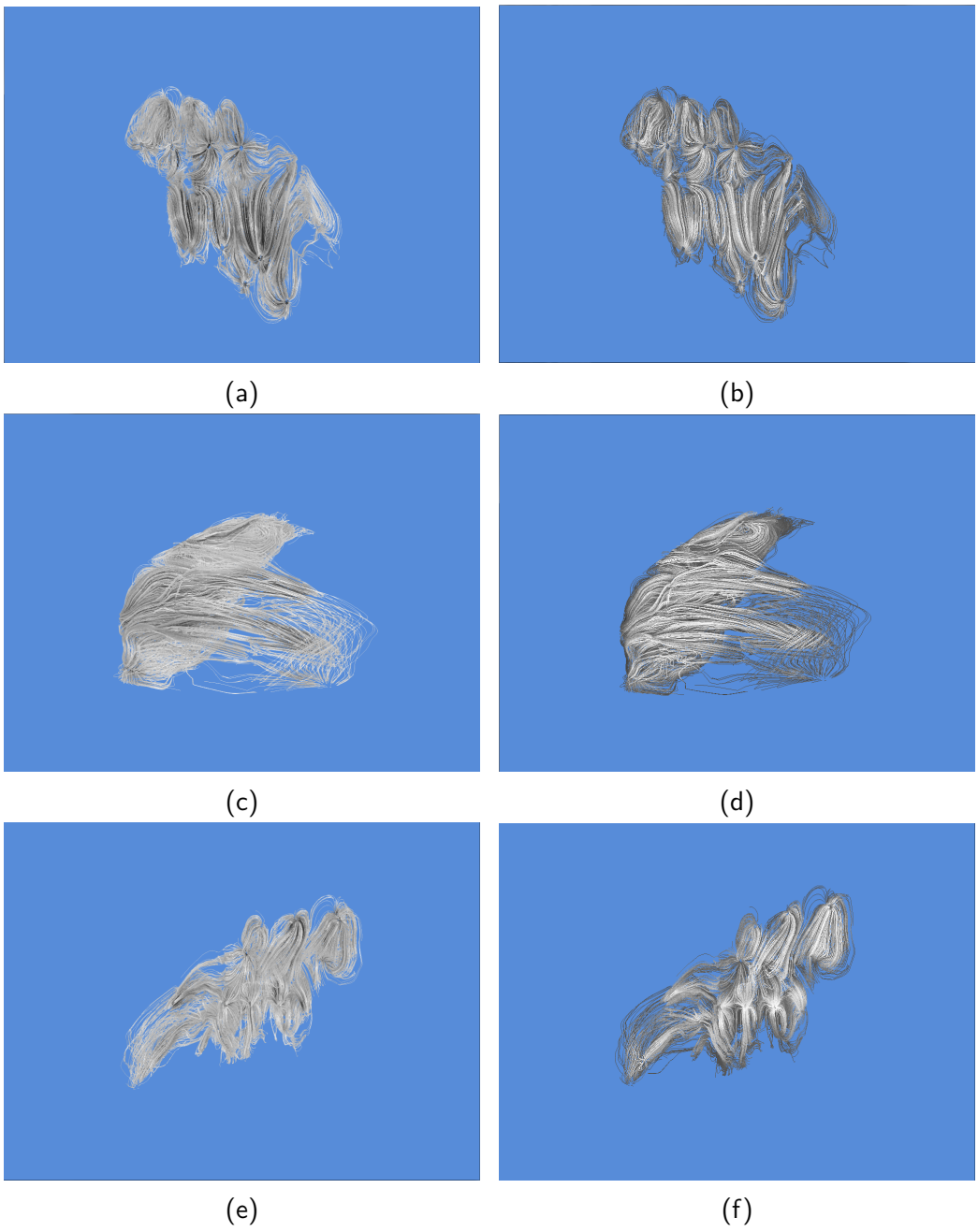
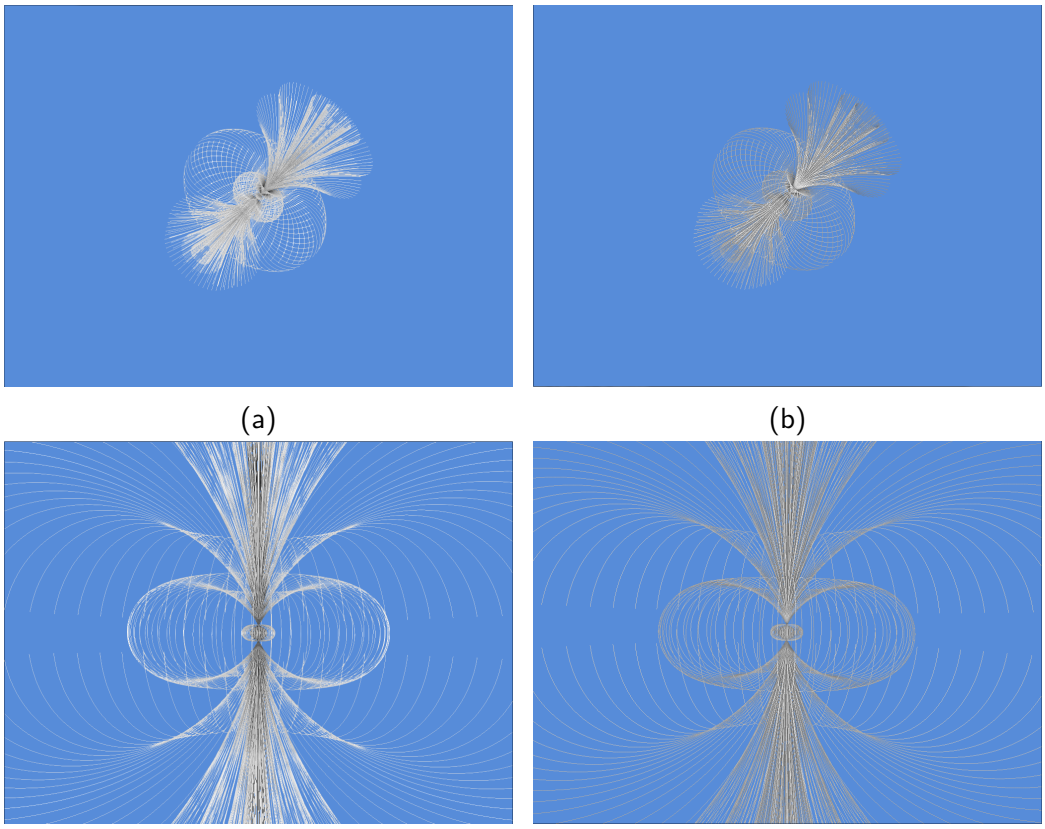


Figura 4.6 – Cena Reservatório 5000. As Figuras 4.6a, 4.6c e 4.6e apresentam poses com o cálculo da oclusão realizada pelo algoritmo proposto, enquanto as Figuras 4.6b, 4.6d e 4.6f utilizam o algoritmo LineAO [1].

Tabela 4.14 – Resultados de tempo para a execução em diversas resoluções do modelo Reservatório 5000.

Resolução	Tempo (ms)	FPS
800 x 600	18.11	55.2
1024 x 768	23.69	42.2
1280 x 960	32.37	30.9
1600 x 1024	35.76	28.0
1920 x 1080	37.71	26.5



(a) (b) (c) (d)

Figura 4.7 – Cena Dipolo. As Figuras 4.7a e 4.7c apresentam poses com o cálculo da oclusão realizada pelo algoritmo proposto, enquanto as Figuras 4.7b e 4.7d utilizam o algoritmo LineAO.

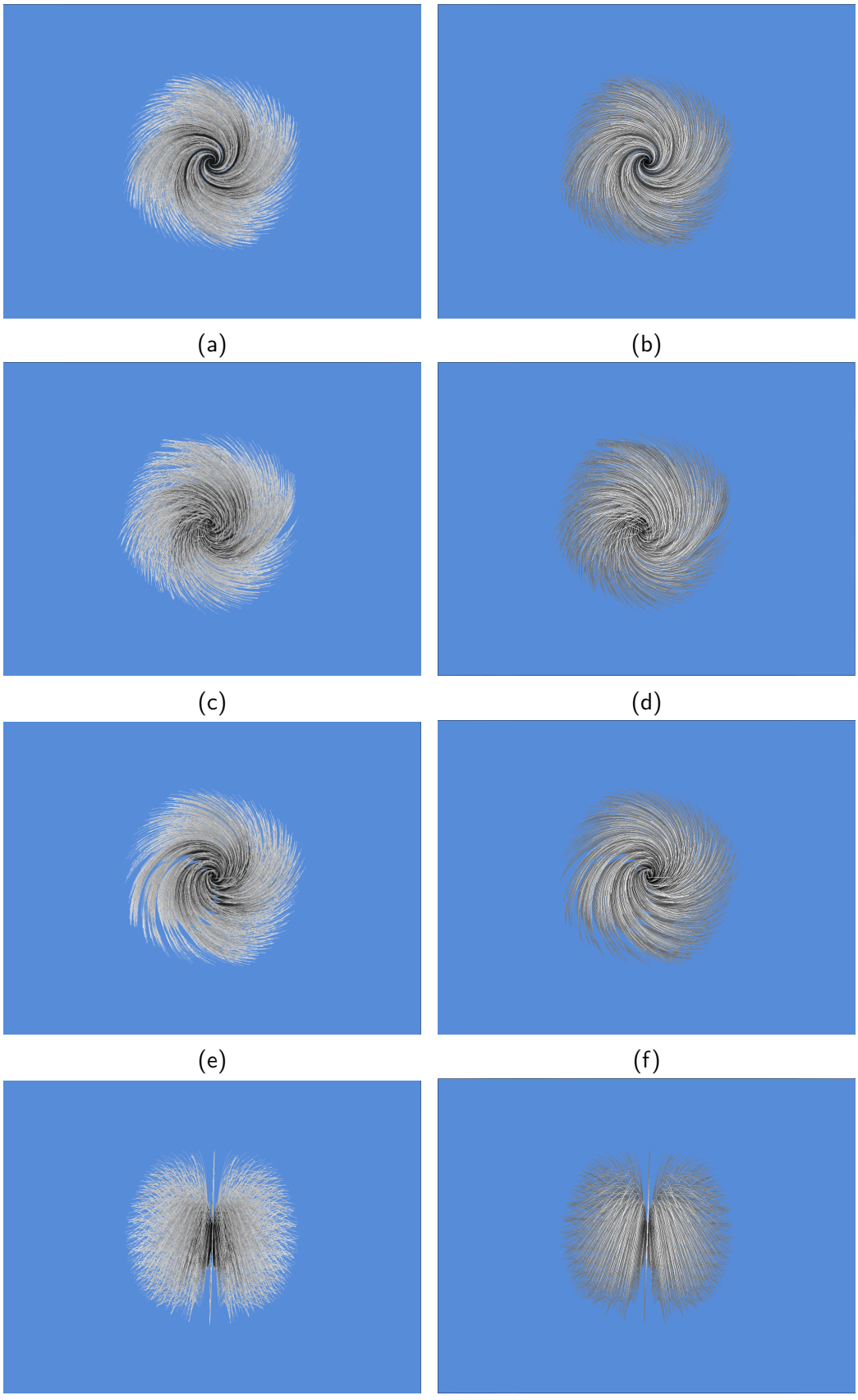
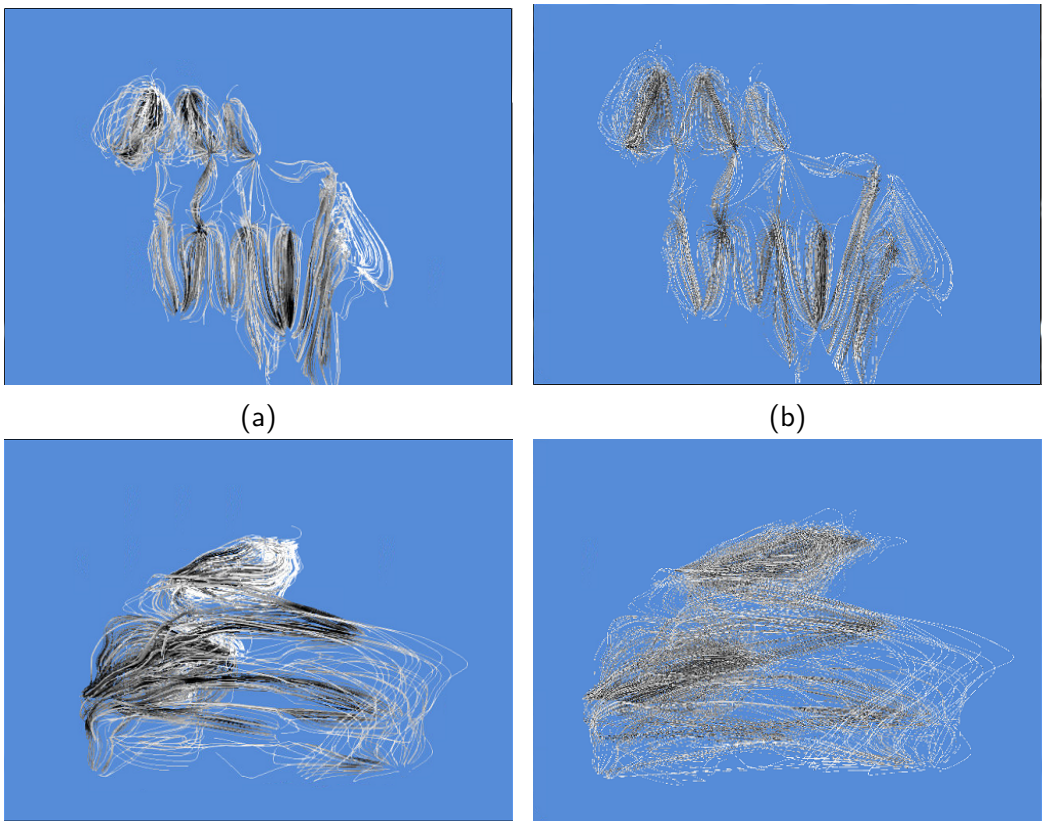


Figura 4.8 – Cena Espiral. As Figuras 4.8a, 4.8c, 4.8e e 4.8g apresentam poses com o cálculo da oclusão realizada pelo algoritmo proposto, enquanto as Figuras 4.8b, 4.8d, 4.8f e 4.8h utilizam o algoritmo LineAO.



(a) (b)

(c) (d)

Figura 4.9 – Comparação entre resultados obtidos com a cena Reservatório 1000, utilizando o método proposto (Figuras 4.9a e 4.9c) e a aplicação de oclusão ambiente em espaço de tela da *Nvidia* (Figuras 4.9b e 4.9d).

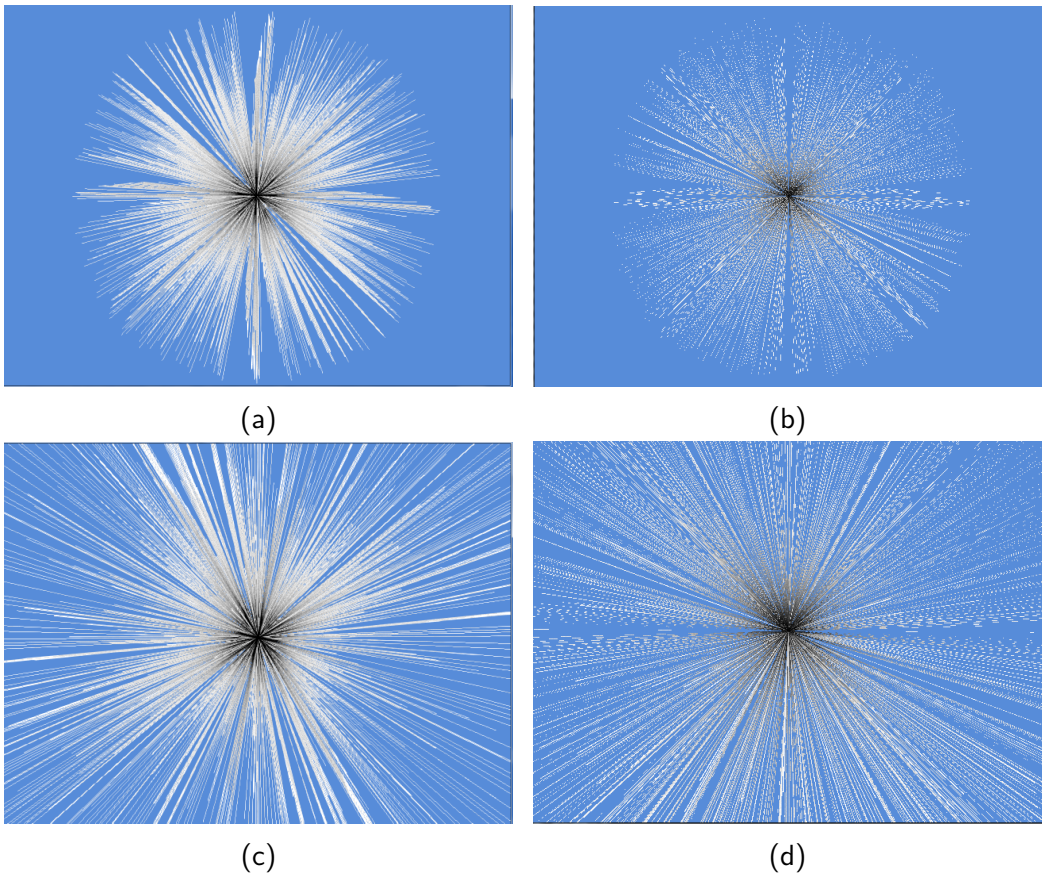


Figura 4.10 – Comparação entre resultados obtidos com a cena Pom-Pom 1000, utilizando o método proposto (Figuras 4.10a e 4.10c) e a aplicação de oclusão ambiente em espaço de tela da *Nvidia* (Figuras 4.10b e 4.10d).

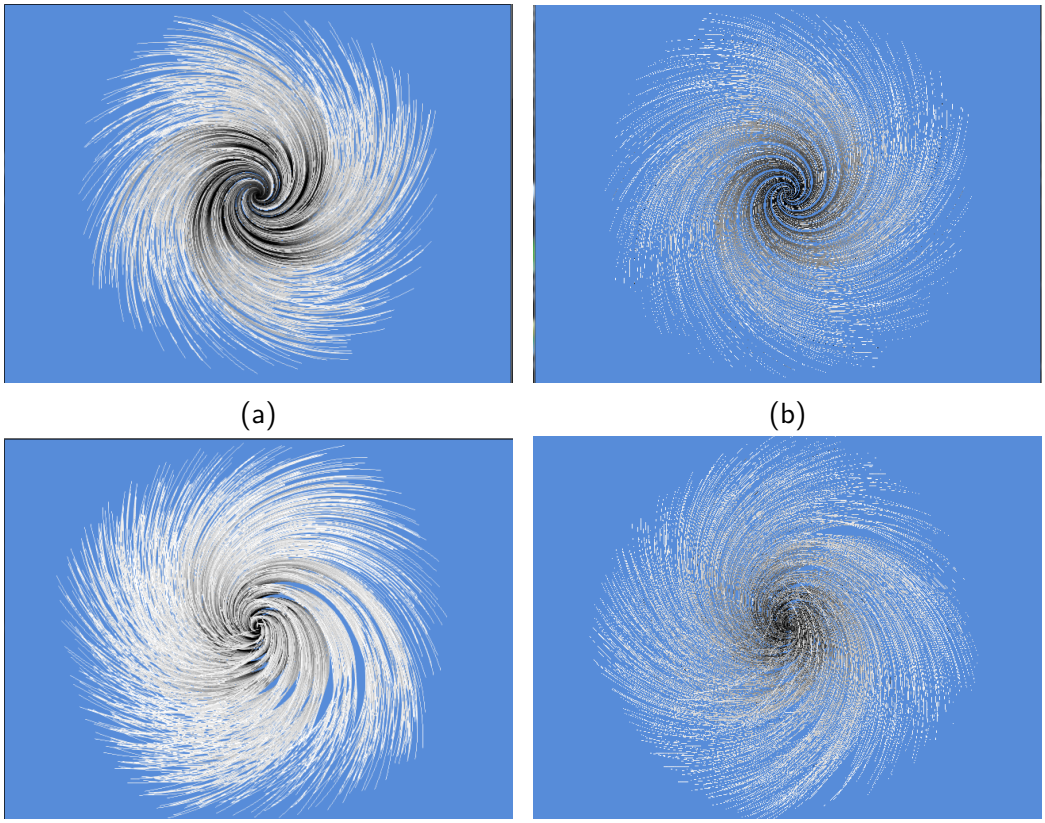


Figura 4.11 – Comparação entre resultados obtidos com a cena Espiral 900, utilizando o método proposto (Figuras 4.10a e 4.10c) e a aplicação de oclusão ambiente em espaço de tela da *Nvidia* (Figuras 4.11b e 4.11d).

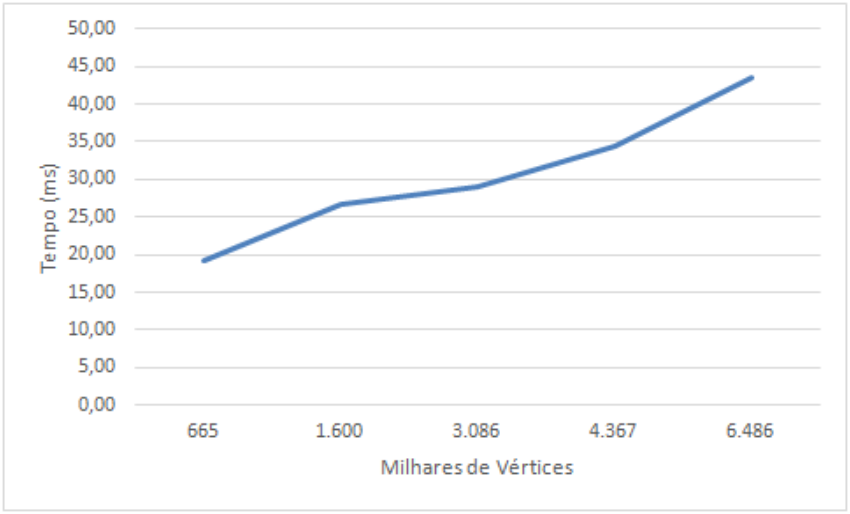


Figura 4.12 – Gráfico exibindo o tempo em milissegundos para a execução de cenas de complexidades geométricas diferentes.

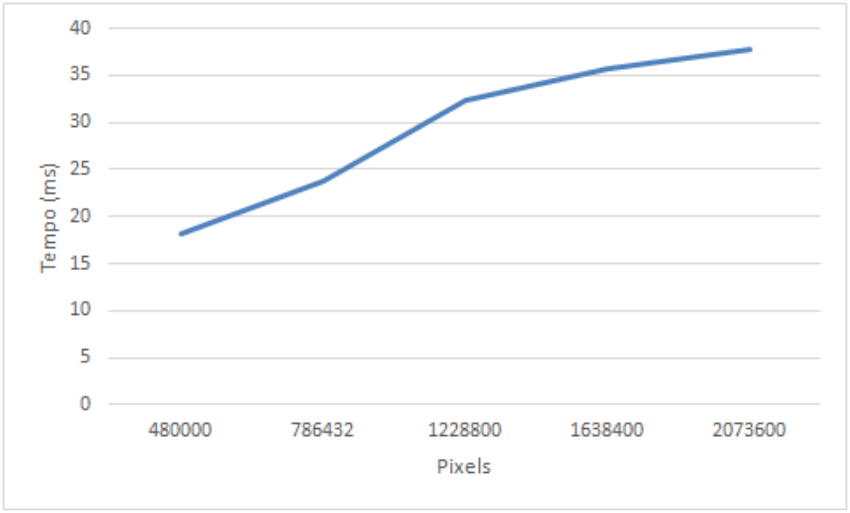


Figura 4.13 – Gráfico exibindo o tempo em milissegundos para a execução de uma mesma cena em diversas resoluções.

5
Conclusão

Este trabalho apresentou uma técnica para estimar a oclusão ambiente para a renderização de linhas em tempo real. Para isso, foi utilizada uma voxelização da cena para disponibilizar a informação da geometria de forma eficiente, de modo a permitir o cálculo da oclusão de pontos que tem sua vizinhança não visível pelo observador no espaço de tela, fazendo com que se tenha uma melhor compreensão da morfologia dos conjuntos de linhas. Foi proposto substituir o *ray-tracing* por um método de amostragem do hemisfério utilizando prismas a partir de pontos no disco do hemisfério. Para obter a oclusão de cada prisma basta utilizar as funções de manipulação de *bits* disponíveis na biblioteca gráfica. Somando as contribuições atenuadas de cada prisma e dividindo pelo número de prismas temos a oclusão no ponto.

O método proposto nesta dissertação consegue produzir imagens em tempo real com qualidade semelhante ao traçado de raios em termos das regiões ocludidas. O método permite também um controle fino de ajuste por parte do usuário através da variação de parâmetros, permitindo assim aumentar cada vez mais a qualidade final.

Considera-se avaliar como trabalhos futuros a utilização de uma grade regular com maior resolução através do uso de mais de uma textura ou o uso de uma textura 3D para representar a voxealização e assim, analisar a relação ganho de qualidade *versus* desempenho. Além disso, pretende-se investigar a utilização de oclusão ambiente para a renderização de fios de cabelo [23, 24].

Referências Bibliográficas

[1] EICHELBAUM, S.; HLAWITSCHKA, M. ; SCHEUERMANN, G.. **Lineao;improved three-dimensional line rendering**. IEEE Transactions on Visualization and Computer Graphics, 19(3):433–445, Mar. 2013.

[2] MALLO, O.; PEIKERT, R.; SIGG, C. ; SADLO, F.. **Illuminated lines revisited**. In: PROCEEDINGS OF IEEE VISUALIZATION (VIS), p. 19–26, 2005.

[3] EISEMANN, E.; DÉCORET, X.. **Single-pass GPU solid voxelization for real-time applications**. In: GRAPHICS INTERFACE 2008, GI '08, MAY, 2008, p. 73–80, Windsor, Canada, May 2008. Canadian Information Processing Society.

[4] CRASSIN, C.; NEYRET, F.; SAINZ, M.; GREEN, S. ; EISEMANN, E.. **Interactive indirect illumination using voxel cone tracing: An insight**. Technical Talk at SIGGRAPH, aug 2011.

[5] FAVERA, E. C. D.; CELES, W.. **Ambient occlusion using cone tracing with scene voxelization**. In: PROCEEDINGS OF THE 2012 25TH SIBGRAPI CONFERENCE ON GRAPHICS, PATTERNS AND IMAGES, SIBGRAPI '12, p. 142–149, Washington, DC, USA, 2012. IEEE Computer Society.

[6] WANGER, L. C.; FERWERDA, J. A. ; GREENBERG, D. P.. **Perceiving spatial relationships in computer-generated images**. IEEE Comput. Graph. Appl., 12(3):44–51, 54–58, May 1992.

[7] WANGER, L.. **The effect of shadow quality on the perception of spatial relationships in computer generated imagery**. In: PROCEEDINGS OF THE 1992 SYMPOSIUM ON INTERACTIVE 3D GRAPHICS, I3D '92, p. 39–42, New York, NY, USA, 1992. ACM.

[8] BANKS, D. C.. **Illumination in diverse codimensions**. In: PROCEEDINGS OF THE 21ST ANNUAL CONFERENCE ON COMPUTER GRAPHICS AND INTERACTIVE TECHNIQUES, SIGGRAPH '94, p. 327–334, New York, NY, USA, 1994. ACM.

[9] ZÖCKLER, M.; STALLING, D. ; HEGE, H.-C.. **Interactive visualization of 3d-vector fields using illuminated stream lines**. In: PROCEEDINGS OF THE 7TH CONFERENCE ON VISUALIZATION '96, VIS '96, p. 107–ff., Los Alamitos, CA, USA, 1996. IEEE Computer Society Press.

[10] MERHOF, D.; SONNTAG, M.; ENDERS, F.; NIMSKY, C.; HASTREITER, P. ; GREINER, G.. **Hybrid visualization for white matter tracts using triangle strips and point sprites**. IEEE Transactions on Visualization and Computer Graphics, 12(5):1181–1188, Sept. 2006.

[11] BLINN, J. F.. **Seminal graphics**. chapter Models of Light Reflection for Computer Synthesized Pictures, p. 103–109. ACM, New York, NY, USA, 1998.

[12] PHONG, B. T.. **Illumination for computer generated pictures**. Commun. ACM, 18(6):311–317, June 1975.

[13] SATTLER, M.; SARLETTE, R.; ZACHMANN, G. ; KLEIN, R.. **Hardware-accelerated ambient occlusion computation**. In: Girod, B.; Magnor, M. ; Seidel, H.-P., editors, VISION, MODELING, AND VISUALIZATION 2004, p. 331–338. Akademische Verlagsgesellschaft Aka GmbH, Berlin, Nov. 2004.

[14] SHANMUGAM, P.; ARIKAN, O.. **Hardware accelerated ambient occlusion techniques on gpus**. In: IN I3D 07: PROCEEDINGS OF THE 2007 SYMPOSIUM ON INTERACTIVE 3D GRAPHICS AND GAMES, ACM. Press.

[15] BAVOIL, L.; SAINZ, M. ; DIMITROV, R.. **Image-space horizon-based ambient occlusion**. In: SIGGRAPH '08: ACM SIGGRAPH 2008 TALKS, p. 1–1, New York, NY, USA, 2008. ACM.

[16] SZIRMAY-KALOS, L.; UMENHOFFER, T.; TÓTH, B.; SZÉCSI, L. ; SBERT, M.. **Volumetric ambient occlusion for real-time rendering and games.** IEEE Computer Graphics and Applications, 30(1):70–79, 2010.

[17] PAPAIOANNOU, G.; MENEXI, M. L. ; PAPADOPOULOS, C.. **Real-time volume-based ambient occlusion**. IEEE Transactions on Visualization and Computer Graphics, 16:752–762, 2010.

[18] NOORUDDIN, F. S.; TURK, G.. **Simplification and repair of polygonal models using volumetric techniques**. IEEE Transactions on Visualization and Computer Graphics, 9:191–205, 2003.

[19] DEERING, M.; WINNER, S.; SCHEDIWY, B.; DUFFY, C. ; HUNT, N.. **The triangle processor and normal vector shader: a vlsi system for high performance graphics**. In: SIGGRAPH, p. 21–30, 1988.

[20] MITTRING, M.. **Finding next gen: Cryengine 2**. In: SIGGRAPH '07: ACM SIGGRAPH 2007 COURSES, p. 97–121, New York, NY, USA, 2007. ACM.

[21] DIMITROV, R.; BAVOIL, L. ; SAINZ, M.. **Horizon-split ambient occlusion**. In: PROCEEDINGS OF THE 2008 SYMPOSIUM ON INTERACTIVE 3D GRAPHICS AND GAMES, I3D '08, p. 5:1–5:1, New York, NY, USA, 2008. ACM.

[22] SCHIRSKI, M.; KUHLEN, T.; HOPP, M.; ADOMEIT, P.; PISCHINGER, S. ; BISCHOF, C.. **Efficient visualization of large amounts of particle trajectories in virtual environments using virtual tubelets**. In: PROCEEDINGS OF THE 2004 ACM SIGGRAPH INTERNATIONAL CONFERENCE ON VIRTUAL REALITY CONTINUUM AND ITS APPLICATIONS IN INDUSTRY, VRCAI '04, p. 141–147, New York, NY, USA, 2004. ACM.

[23] YUKSEL, C.; TARIQ, S.. **Advanced techniques in real-time hair rendering and simulation**. In: ACM SIGGRAPH 2010 COURSES, SIGGRAPH '10, p. 1:1–1:168, New York, NY, USA, 2010. ACM.

[24] WARD, K.; BERTAILS, F.; KIM, T.-Y.; MARSCHNER, S. R.; CANI, M.-P. ; LIN, M. C.. **A survey on hair modeling: Styling, simulation, and rendering**. IEEE Transactions on Visualization and Computer Graphics, 13(2):213–234, Mar. 2007.

[25] NVIDIA. **Nvidia® optix 2 ray tracing engine examples**, June 2011. `http://developer.nvidia.com/optix-interactive-examples`.

[26] NVIDIA. **Nvidia direct3d sdk 10 code samples**, June 2011. `http://developer.download.nvidia.com/SDK/10.5/direct3d/samples.html`.