UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

HADRYAN SALLES

ANÁLISE COMPARATIVA DE DIFERENTES TÉCNICAS DE ILUMINAÇÃO VOLUMÉTRICA

CURITIBA

2025

versão 11.0 (abr. 25)

HADRYAN SALLES

ANÁLISE COMPARATIVA DE DIFERENTES TÉCNICAS DE ILUMINAÇÃO VOLUMÉTRICA

Comparative Analysis of Different Volumetric Light Techniques

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia da Computação do Curso de Bacharelado em Engenharia da Computação da Universidade Tecnológica Federal do Paraná.
Orientador(a): Prof. Dr. Ricardo Dutra da Silva

CURITIBA

2025

HADRYAN SALLES

ANÁLISE COMPARATIVA DE DIFERENTES TÉCNICAS DE ILUMINAÇÃO VOLUMÉTRICA

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia da Computação do Curso de Bacharelado em Engenharia da Computação da Universidade Tecnológica Federal do Paraná.

Data de aprovação: 11 / julho / 2025

Bogdan Tomoyuki Nassu
Doutorado
Universidade Tecnológica Federal do Paraná

Leyza Elmeri Baldo Dorini
Doutorado
Universidade Tecnológica Federal do Paraná

Ricardo Dutra da Silva
Doutorado
Universidade Tecnológica Federal do Paraná

CURITIBA
2025

AGRADECIMENTOS

Agradeço ao professor Ricardo Dutra pela orientação atenta e dedicada durante o desenvolvimento deste trabalho, e aos professores Bogdan e Leyza pela atenção e participação na banca avaliadora. Também registro minha gratidão ao professor Francisco Ganacim, que me auxiliou em meus primeiros passos na área de computação gráfica.

Sou grato à minha família, à minha namorada Patrícia e aos amigos pelo apoio constante ao longo da minha trajetória acadêmica. Aos colegas de curso, agradeço pelos momentos compartilhados, entre desafios e aprendizados.

Por fim, agradeço aos talentosos artistas e programadores que dedicam suas vidas à criação de jogos, verdadeiras obras de arte que aprecio desde a infância e que despertaram meu interesse pela computação gráfica, mantendo viva minha inspiração até hoje.

Cyberspace. A consensual hallucination experienced daily by billions of legitimate operators, in every nation, by children being taught mathematical concepts... A graphic representation of data abstracted from banks of every computer in the human system. Unthinkable complexity. Lines of light ranged in the nonspace of the mind, clusters and constellations of data. Like city lights, receding... (Gibson, 1984)

RESUMO

Neste trabalho foram implementados, analisados e comparados três métodos de renderização de luz volumétrica em tempo real, baseados em abordagens distintas: pós-processamento de imagem (*Screen Space*), subdivisão do espaço (*Froxel*) e geração geométrica (*Polygonal*). A luz volumétrica é um efeito ótico observado quando a luz, no caminho desde sua fonte até o observador, interage com partículas presentes no meio de transmissão, como poeira ou neblina, gerando feixes visíveis. Para realizar a comparação, foi desenvolvido um motor gráfico em *C++* com a API Vulkan, capaz de renderizar cenas com malhas 3D e múltiplas luzes direcionais, aplicando os três métodos de iluminação volumétrica de forma intercambiável. A aplicação realiza automaticamente capturas de tela e registra métricas como tempo de renderização e uso de memória. O método baseado em pós-processamento de imagem simula o efeito com baixo custo computacional, mas apresenta limitações visuais, como falta de oclusão e artefatos de amostragem. O método *Froxel* divide o *frustum* da câmera em células 3D (*froxels*) e acumula luz ao longo do raio de visão usando texturas 3D e paralelismo em GPU, alcançando um bom equilíbrio entre desempenho e qualidade. O método *Polygonal* utiliza geometria gerada a partir do mapa de sombras para representar o volume de luz com alto detalhamento visual, mas com maior custo computacional e consumo de memória. Os resultados obtidos mostram que o método *Froxel* apresenta o melhor desempenho em tempo de execução e escalabilidade com a complexidade da cena, enquanto o método *Polygonal* alcança a melhor fidelidade visual em situações que permitem maior uso de recursos. Já o método de pós-processamento de imagem se destaca pela simplicidade de implementação e baixo uso de memória, sendo adequado para aplicações com restrições severas de hardware. Os três métodos foram avaliados de forma sistemática em múltiplas cenas, e o código-fonte completo, junto com os dados experimentais e cenas de teste, está disponível publicamente para reprodutibilidade e reutilização por outros pesquisadores.

Palavras-chave: luz volumétrica; renderização em tempo real; computação gráfica; análise comparativa.

ABSTRACT

This work implements, analyzes, and compares three real-time volumetric lighting rendering methods based on distinct approaches: image post-processing (Screen Space), space subdivision (Froxel), and geometric generation (Polygonal). Volumetric lighting is an optical effect that occurs when light interacts with particles in the transmission medium, such as dust or fog, producing visible light beams. A graphics engine was developed in *C++* using the Vulkan API, capable of rendering 3D mesh scenes with multiple directional lights and interchangeable volumetric lighting techniques. The application automatically captures screenshots and records metrics such as rendering time and memory usage. The image post-processing method offers low computational cost but suffers from visual limitations like lack of occlusion and sampling artifacts. The Froxel method subdivides the camera frustum into 3D cells (froxels), accumulating light along view rays using 3D textures and GPU parallelism, balancing performance and quality. The Polygonal method uses geometry generated from shadow maps to represent the light volume with high visual detail at a higher computational and memory cost. Results indicate that the Froxel method achieves the best runtime performance and scalability with scene complexity, while the Polygonal method delivers superior visual fidelity when resources allow. The image post-processing method stands out for implementation simplicity and low memory usage, suitable for hardware-constrained scenarios. All three methods were systematically evaluated across multiple scenes, and the full source code, experimental data, and test scenes are publicly available to ensure reproducibility and facilitate further research.

Keywords: volumetric light; real-time rendering; computer graphics; comparative analysis.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Simulações de efeitos luminosos são utilizadas em diferentes áreas de pesquisa quando deseja-se replicar, computacionalmente, características da luz. Essas simulações são fundamentadas por modelos matemáticos que descrevem o comportamento da luz ao interagir com as demais partículas do ambiente (Pharr; Jakob; Humphreys, 2023), podendo elas ser numéricas (Karlsson *et al.*, 2004) ou gráficas (Baranoski; Krishnaswamy, 2010).

Em aplicações interativas de renderização em tempo real, a qualidade da experiência é determinada por, entre outros fatores, a latência entre as ações do usuário e a resposta da aplicação (Ivkovic *et al.*, 2015). Além disso, tratando-se de aplicações visuais, a frequência com a qual as imagens são geradas e exibidas na tela, usualmente chamada de FPS (*frames per second*, ou quadros por segundo), é extremamente importante para manter a sensação de continuidade entre as imagens (Claypool; Claypool; Damaa, 2006)

Grande parte dos jogos digitais encontram-se na interseção entre aplicações de renderização em tempo real e simulações gráficas de efeitos luminosos. Nesse tipo de jogo, um mundo tridimensional, que possui objetos e luzes, é renderizado a partir de um ponto de vista, a fim de obter-se uma imagem e exibi-la ao jogador.

Na computação gráfica são feitas restrições à precisão dos modelos matemáticos utilizados para simular a luz, com o intuito de diminuir os tempos de renderização. Sendo assim, muitas vezes a luz não é simulada de forma holística (Pharr; Jakob; Humphreys, 2023) com uma equação geral, mas cada efeito é implementado com uma técnica separada. Por exemplo, o uso de mapas de sombra (Stamminger; Drettakis, 2002), que não solucionam o modelo de iluminação global, mas descrevem uma forma eficiente de determinar quais regiões da cena estão em sombra. Dessa forma, utilizando técnicas separadas para cada efeito, é possível maximizar a eficiência da simulação de acordo com a flexibilidade desejada para cada efeito visual.

Este trabalho concentra-se em diferentes técnicas utilizadas para simular graficamente uma das classes de efeitos da luz: a luz volumétrica. O termo luz volumétrica refere-se ao caso específico de dispersão da luz ao atravessar um meio que participa do seu transporte (Lambru *et al.*, 2021). Um exemplo prático seria um dia com neblina, nesse caso, ao interagir com partículas de água que estão suspensas no ar, a luz do sol forma um volume visível, como ilustra a fotografia da Figura 1. Isto ocorre pois parte da luz que interage com as partículas irá refratar e refletir em direção ao observador.

Figura 1 – Fotografia exemplificando o efeito de luz volumétrica do sol ao interagir com neblina. São visíveis cones de luz projetados a partir do sol entre os galhos de uma árvore.

Fonte: "Foggy Morning Sun" por Johan Neven, disponível em: https://www.flickr.com/photos/enneafive/45807625392, licenciado por CC BY 2.0 (https://creativecommons.org/licenses/by/2.0/)..

Dentro desta classe de efeitos ainda existem diversos parâmetros que podem variar de acordo com a situação, como, entre outros, o material e densidade das partículas e se o meio é homogêneo ou heterogêneo (Billeter; Sintorn; Assarsson, 2012). Considerando esses fatores, simular graficamente luzes volumétricas de forma eficiente (com alta taxa de quadros por segundo) e fisicamente apurada (compatível com modelos físicos que descrevem o comportamento da luz) é um problema desafiador na área de computação gráfica.

1.1 Objetivos

O objetivo deste trabalho é implementar diferentes técnicas utilizadas para renderização de luz volumétrica a fim de compará-las, medindo fatores quantitativos, como tempo de execução e uso de memória, qualitativos, como artefatos visuais produzidos por cada método, e analisar características gerais como limitações de cada abordagem.

1.2 Justificativa

Buscando demonstrar os pontos positivos e negativos de diferentes técnicas utilizadas na computação gráfica, este trabalho justifica-se como um ponto de partida para entender qual abordagem é mais apropriada de acordo com os requisitos de cada aplicação. Ao implementar diferentes técnicas e aplicá-las ao mesmo ambiente virtual composto por luzes e objetos, é possível obter uma comparação imparcial sobre a utilização de recursos computacionais (tempo e memória) de cada algoritmo.

Tratando-se do contexto de computação gráfica de tempo de real, características como tempo e uso de memória são de extrema importância para a experiência do usuário que interage com a aplicação. Porém, considerando a visão artística do responsável por criar a cena virtual,

a qualidade da simulação também deve ser considerada a fim de obter um equilíbrio entre qualidade gráfica e performance.

1.3 Estrutura do trabalho

O trabalho é dividido nos seguintes capítulos. O Capítulo 2 apresenta de forma sucinta os conceitos básicos e referências utilizadas para a implementação dos métodos de iluminação volumétrica. O Capítulo 3 descreve os detalhes técnicos de implementação dos métodos. Experimentos são apresentados no Capítulo 4, analisando de forma quantitativa e qualitativa os resultados obtidos. Por fim, são feitas as conclusões e sugestões de trabalhos futuros no Capítulo 5.

2 REFERENCIAL TEÓRICO E TRABALHOS RELACIONADOS

Esta seção tem como objetivo apresentar o arcabouço teórico utilizado para o desenvolvimento deste trabalho.

2.1 Conceitos

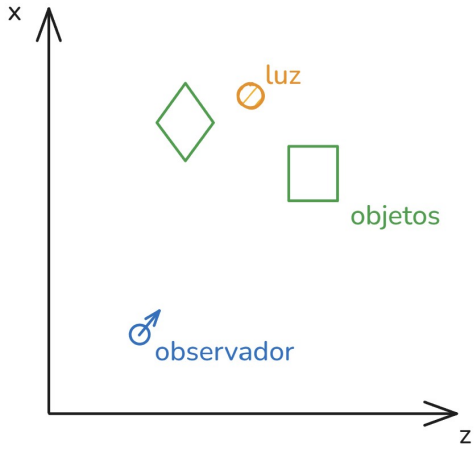
2.1.1 Fluxo geral de síntese de imagens

O processo de síntese de imagens pode ser descrito como uma sequência de operações realizadas sobre um conjunto de entradas, com o objetivo de gerar uma imagem de saída. As entradas formam a descrição da cena, que no contexto do trabalho, podem ser agrupadas em:

- **Objetos 3D**: Representados por malhas de triângulos, possuem atributos como posição, orientação, escala, materiais e texturas. Esses dados determinam a aparência e o comportamento dos objetos na cena.
- **Fontes de luz**: Responsáveis por definir a origem da iluminação que incide sobre os objetos, influenciando diretamente a aparência de cores, sombras e brilhos na imagem final. Entre os diferentes tipos possíveis, destaca-se a luz direcional, que simula uma fonte de iluminação distante, como o Sol. Esse tipo de luz é modelado por raios paralelos e sua intensidade não varia com a distância até os objetos. Uma luz direcional é definida por sua posição e direção no espaço, intensidade luminosa e cor.
- **Câmera**: Representa o ponto de vista do observador dentro da cena, sendo responsável por definir qual região do espaço tridimensional será visualizada e como essa visualização será projetada na imagem final. Seus principais parâmetros incluem a posição e orientação no espaço, o campo de visão e os planos de corte (*z near* e *z far*). A partir desses parâmetros, são derivadas duas matrizes fundamentais: a matriz de visualização (M_{view}), que transforma os objetos do mundo para o referencial da câmera, e a matriz de projeção (M_{proj}), que define a projeção dos objetos tridimensionais sobre o plano da imagem.

O processo de síntese de imagens tem início com a transformação dos objetos da cena, que estão originalmente definidos no *espaço do mundo*. Nesse espaço, como ilustrado na Figura 2, cada objeto possui sua posição e orientação em um sistema de coordenadas global.

Figura 2 – Objetos definidos no espaço do mundo.

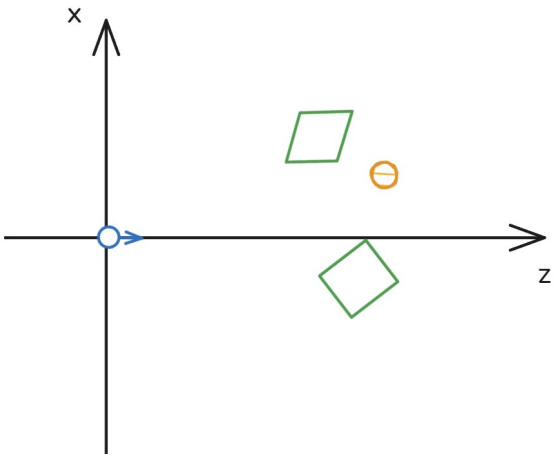


Fonte: Autoria própria (2025).

Cabe ressaltar que, por uma questão de clareza visual, tanto a Figura 2 quanto os próximos exemplos, representam cortes bidimensionais no plano horizontal da cena, omitindo o eixo vertical (Y). Essa simplificação foi adotada com o objetivo de facilitar a leitura e compreensão dos conceitos, sem prejuízo à generalidade tridimensional do processo.

A primeira transformação consiste em levar esses objetos para o *espaço do observador*, também chamado de *espaço da câmera*, por meio da aplicação da matriz de visualização M_{view}. Essa aplicação consiste em multiplicar cada vértice da malha, representado como um vetor em coordenadas homogêneas (isto é, um vetor 4D formado pela posição tridimensional com a componente 1 adicionada na quarta coordenada), pela matriz M_{view}, de dimensão 4×4. . Essa etapa é representada na Figura 3, em que o referencial da cena passa a estar alinhado com o ponto de vista da câmera.

Figura 3 – Objetos transformados para o espaço da câmera.

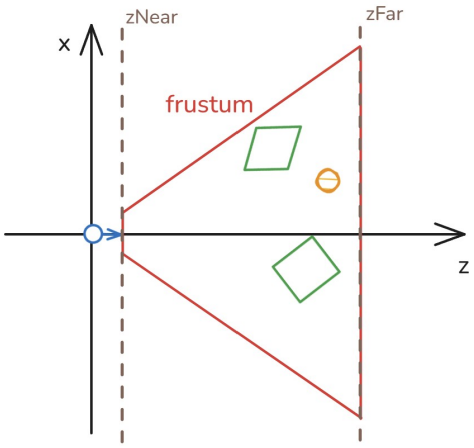


Fonte: Autoria própria (2025).

Na sequência, os dados são projetados para o *espaço de recorte* utilizando a matriz de projeção M_{proj}. Neste trabalho, foi utilizada projeção em perspectiva, que define o volume visível da câmera como um *frustum*. Esse volume corresponde a uma pirâmide truncada com vértice na posição da câmera e base voltada para frente, representando a região do espaço tridi-

mensional que será visualizada. A Figura 4 ilustra esse *frustum*, destacando os planos de corte próximo (*z near*) e distante (*z far*), que limitam a profundidade mínima e máxima considerada na renderização.

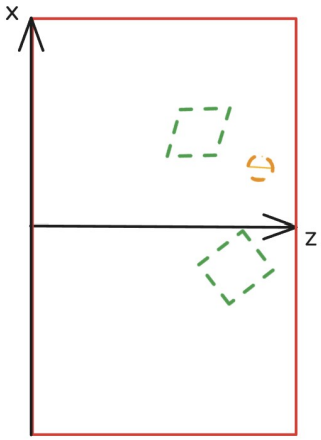
Figura 4 – Volume visível (frustum) delimitado pelos planos *z near* e *z far*.



Fonte: Autoria própria (2025).

A aplicação da matriz de projeção converte esse volume visível para uma região geométrica padronizada chamada de cubo de recorte canônico (*canonical clip cube*). Como ilustrado na Figura 5, esse cubo possui coordenadas normalizadas no intervalo $[-1, 1]$ nos eixos x e y, e $[0, 1]$ no eixo z. Objetos localizados fora desse volume são descartados durante a etapa de recorte. Ainda na Figura 5, os objetos são representados com linhas tracejadas para indicar que sua geometria pode ser alterada após a projeção. Isso ocorre devido à transformação não linear introduzida pela projeção em perspectiva, que afeta principalmente a distribuição dos valores de profundidade.

Figura 5 – Volume visível convertido para o espaço de recorte canônico (*canonical clip cube*).

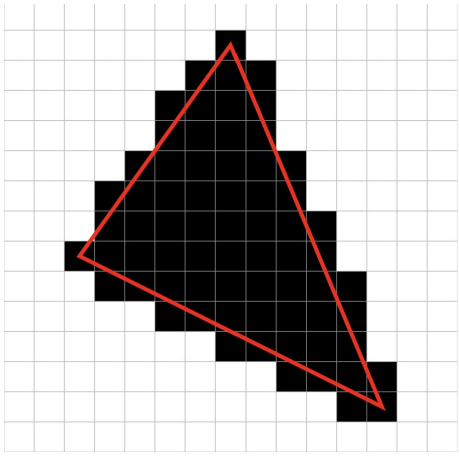


Fonte: Autoria própria (2025).

Após as transformações, os triângulos da cena são rasterizados, ou seja, convertidos em fragmentos, que são unidades básicas de processamento gráfico associadas aos *pixels* da imagem final. Durante essa etapa, o sistema percorre todos os triângulos visíveis e identifica quais *pixels* da tela são cobertos por cada um deles. A Figura 6 apresenta um exemplo de triângulo

(representado pelo contorno vermelho) e o resultado de sua rasterização (*pixels* pretos). Para cada *pixel* rasterizado, um fragmento é gerado e processado individualmente na etapa do *fragment shader*, em que são realizados os cálculos de texturas, iluminação base e sombreamento. O resultado desse processamento define o valor final de cor atribuído ao *pixel* correspondente na imagem, formando o espaço de tela, representando por coordenadas (i,j) com limites em $[(0,0),(W,H)]$, sendo W e H, respectivamente, a largura e altura da imagem renderizada.

Figura 6 – Fragmentos do triângulo representados por *pixels* pretos.



Fonte: Autoria própria (2025).

Os métodos de iluminação volumétrica analisados neste trabalho não substituem esse fluxo padrão de síntese de imagens, mas atuam como etapas adicionais, inseridas após o cálculo de iluminação convencional. Cada um dos métodos implementados utiliza os dados gerados anteriormente e adiciona contribuições específicas de luz volumétrica à imagem final.

2.1.2 Mapa de Profundidade

O mapa de profundidade, ou *depth map*, é uma imagem em que cada *pixel* armazena a distância entre a câmera e a superfície mais próxima da câmera naquele ponto específico da cena, conforme o exemplo apresentado na Figura 7. Os valores armazenados no mapa de profundidade são expressos no espaço de recorte, variando de 0 (mais próximo da câmera) até 1 (mais distante). Dessa forma, para um ponto $p = (i,j)$ no espaço de tela, pode-se definir a função Depth(p) como sendo o valor do mapa de profundidade em p.

Figura 7 – Exemplo de mapa de profundidade.

Fonte: Autoria própria (2025).

Para uma câmera definida pelas matrizes de visão $\mathbf{M}_{\text{view}}$ e projeção $\mathbf{M}_{\text{proj}}$, associada a uma imagem com resolução W por H, a transformação de um ponto p_{world} (ponto no espaço do mundo) para o ponto p_{clip} no espaço de recorte é dada pela Equação 1. Além disso, a transformação para espaço de tela p_{screen} ocorre de acordo com a Equação 2. Nota-se que o espaço de tela é 2D, portanto a componente z é descartada.

$$p_{\text{clip}} = \mathbf{M}_{\text{proj}} \cdot \mathbf{M}_{\text{view}} \cdot p_{\text{world}}. \tag{1}$$

$$p_{\text{screen}} = \lfloor p_{\text{clip}} \times [W, H, 0] \rfloor \tag{2}$$

A partir disso, o teste de visibilidade de um ponto p_{world} pode ser realizado comparando o componente z de p_{clip} com o valor correspondente armazenado no mapa de profundidade $\text{Depth}(p_{\text{screen}})$. Caso o valor armazenado seja menor, isso indica que existe um objeto mais próximo da câmera, e portanto o ponto p_{world} está ocluído.

Outra grande utilidade do mapa de profundidade é realizar a reprojeção de coordenadas de tela em coordenadas do mundo. Essa reprojeção, para determinado *pixel* p_{screen} pode ser calculada de acordo com a Equação 3:

$$p_{\text{world}} = \mathbf{M}_{\text{view}}^{-1} \cdot \left(\mathbf{M}_{\text{proj}}^{-1} \cdot p_{\text{clip}}\right), \quad \text{com} \quad p_{\text{clip}} = \left(2\frac{p_{\text{screen}}}{(W, H)} - 1, \text{Depth}(p_{\text{screen}}), 1\right) \tag{3}$$

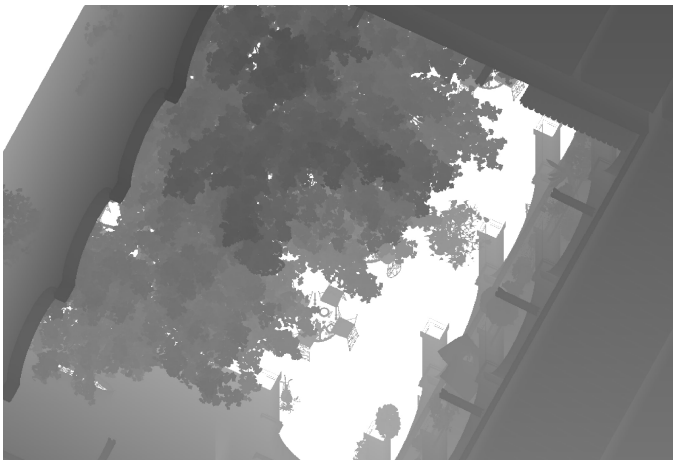
, tal que

- $\mathbf{M}_{\text{proj}}^{-1}$: matriz inversa de projeção;
- $\mathbf{M}_{\text{view}}^{-1}$: matriz inversa da câmera;

2.1.3 Mapa de Sombra

O mapa de sombras é um mapa de profundidade gerado a partir de uma fonte de luz, portanto, possui as mesmas propriedades do mapa de profundidade. Esse tipo de mapa é renderizado utilizando matrizes de câmera e de projeção criadas a partir da posição e direção da fonte de luz. No caso de luzes direcionais, é utilizada uma matriz de projeção ortográfica para renderizar todos os objetos da cena, a partir do ponto de vista da luz. A Figura 8 apresenta um exemplo desse tipo de mapa, cuja principal utilidade é, por meio de teste de visibilidade (Stamminger; Drettakis, 2002), definir quais pontos da cena estão em sombra.

Figura 8 – Exemplo de mapa de sombra.



Fonte: Autoria própria (2025).

A partir do mapa de sombras de uma luz L, é possível definir a função de sombra da Equação 4, que utiliza as Equações 1 e 3 para converter um ponto p no espaço do mundo em seus respectivos pontos no espaço de recorte da luz p_{clip} e no espaço de tela da luz p_{screen}.

$$\text{Shadow}_L(p) = \begin{cases} 0, & \text{se } p_{\text{clip}}[z] > \text{Depth}_L(p_{\text{screen}}) + \epsilon \\ 1, & \text{caso contrário} \end{cases} \tag{4}$$

2.1.4 Função de Fase

A função de fase é um modelo matemático utilizado para descrever como ocorre o espalhamento da luz devido a partículas presentes no meio pela qual ela é transmitida. Essa função define a direção predominante do espalhamento, indicando, por exemplo, se a luz tende a continuar na mesma direção (espalhamento frontal), ser refletida (espalhamento retroativo) ou ser distribuída uniformemente (espalhamento isotrópico).

Neste trabalho foi utilizada a função de fase de Henyey e Greenstein (1941), descrita pela Equação 5. Esta função utiliza o parâmetro g (constante anisotrópica) para calcular a quantidade de luz que, vinda da direção ω_l, irá ser espalhada na direção ω_v.

$$\rho_m(\omega_v, \omega_l) = \frac{1 - g^2}{4\pi \left(1 + g^2 - 2g(\omega_v \cdot \omega_l)\right)^{3/2}} \tag{5}$$

2.2 Trabalhos Relacionados

A simulação gráfica de luz volumétrica tem sido objeto de diversas pesquisas, que exploram diferentes estratégias para equilibrar fidelidade visual e desempenho em tempo real. Mitchell (2007) propôs uma abordagem baseada em pós-processamento de imagem, que se destaca pela simplicidade de implementação e eficiência computacional, embora apresente limitações perceptíveis, como a ausência de oclusão e contornos imprecisos. Billeter, Sintorn e Assarsson (2010) introduziram um método geométrico utilizando malhas auxiliares para representar o volume de luz com alta precisão visual, mas com elevado custo de processamento. Posteriormente, em Billeter, Sintorn e Assarsson (2012), os mesmos autores exploraram técnicas baseadas em *light propagation volumes*.

A partir de 2014, aparecem métodos que fazem uso da ideia de *voxels* para subdividir o espaço e criar uma textura 3D que armazena a luz volumétrica na cena, com Wrónski (2015) popularizando a abordagem. Abordagens baseadas em *voxels* são amplamente utilizadas nos dias de hoje, aparecendo em motores gráficos de renome como *Unreal Engine* (Epic Games, 2024). Ainda no contexto de *voxels*, Kovalovs (2020) descreve como foi feita a implementação de efeitos volumétricos no jogo *The Last of Us: Part Two*[1], com grande ênfase na eficiência e otimização das técnicas.

Além das abordagens implementadas, outras técnicas relevantes para a simulação de luz volumétrica em tempo real não foram exploradas neste trabalho. Uma delas é o *ray marching*, utilizado por Zhou *et al.* (2008), em que são traçados raios a partir da câmera, e ao longo de cada raio são feitas diversas amostras de iluminação. Cada amostra contribui para o acúmulo de iluminação volumétrica final observada no pixel correspondente. Essa técnica permite representar feixes de luz com boa fidelidade visual, mas possui alto custo computacional devido à grande quantidade de amostras necessárias. Outra estratégia amplamente utilizada é o *path tracing*, como descrita por Kajiya (1986), que simula as múltiplas interações da luz com superfícies (e meios de transmissão) da cena, para capturar efeitos de iluminação com alto grau de realismo. Embora essas abordagens ofereçam resultados visuais superiores, sua aplicação em contextos interativos ainda é limitada devido ao custo de processamento envolvido, sendo mais utilizadas em renderizadores *offline* como o caso de produções cinematográficas.

[1] *The Last of Us: Part Two*™ é um jogo eletrônico desenvolvido pela *Naughty Dog* e publicado em 2020 pela *Sony Interactive Entertainment* para o console *PlayStation 4*. Marca registrada de seus respectivos detentores.

3 METODOLOGIA

Neste capítulo são descritas as estratégias adotadas para o desenvolvimento e avaliação dos métodos de iluminação volumétrica analisados. Inicialmente, apresenta-se a arquitetura geral do motor gráfico implementado, que fornece as funcionalidades básicas de renderização e manipulação de cenas tridimensionais. Em seguida, são detalhadas as implementações específicas de cada um dos três métodos comparados, abordando suas estruturas, entradas, processamentos e resultados.

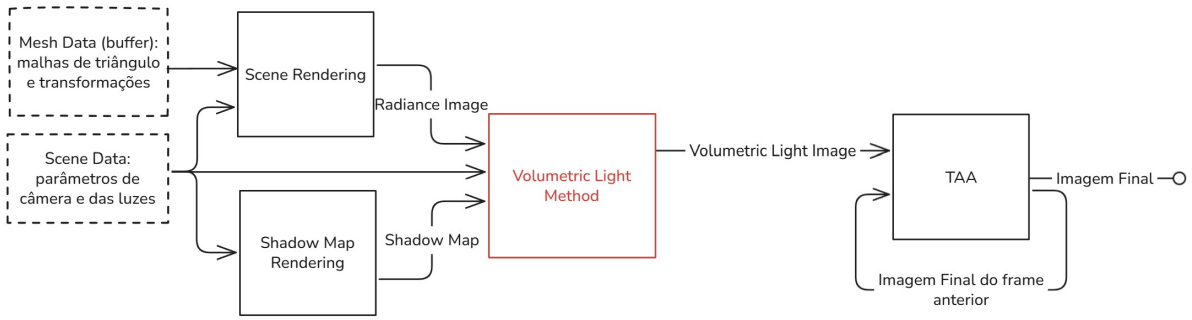
3.1 Motor gráfico

Para realizar as análises e comparações entre os diferentes métodos de iluminação volumétrica, além de implementá-los, foi desenvolvido um motor gráfico base. Esta aplicação, nomeada *Luz Engine*, foi implementada em C++ e utiliza a API gráfica Vulkan, sendo compatível com sistemas operacionais Windows e Linux. O código da aplicação, assim como de todos os demais métodos implementados durante o trabalho estão disponibilizados em repositório online[1], possuindo código aberto sob licença MIT[2].

As principais funcionalidades da aplicação base incluem carregar malhas de triângulos e adicionar/manipular luzes direcionais. Além disso, para permitir a reprodutibilidade dos testes, todos os projetos, contendo malhas 3D, luzes e câmera, podem ser salvos, permitindo recarregar a cena posteriormente com os exatos mesmos parâmetros.

A Figura 9 apresenta um diagrama com as etapas envolvidas na renderização de um *frame* na aplicação *Luz Engine*. Nesta figura, os retângulos de borda sólida representam diferentes *shaders* (programas executados na placa de vídeo, em paralelo para triângulos da cena, ou para *pixels* da imagem, a depender do contexto), enquanto os quadrados tracejados e linhas conectando os *shaders* representam os diferentes tipos de dados utilizados.

Figura 9 – Diagrama de renderização do motor gráfico.



Fonte: Autoria própria (2025).

Descrição dos dados:

[1] https://github.com/hadryansalles/Luz
[2] https://opensource.org/license/mit

- `Mesh Data`: um *buffer* contendo as malhas de triângulo de todos os objetos da cena, assim como suas texturas e transformações.
- `Scene Data`: parâmetros gerais da cena como: matrizes de câmera, lista de luzes (posição, direção, intensidade, cor), coeficientes dos métodos de iluminação volumétrica (*scattering*, densidade, anisotrópico).
- `Radiance Image`: contém a imagem renderizada com iluminação aplicada (não volumétrica), incluindo sombras, texturas, materiais e mapa de profundidade (exemplo apresentado na Figura 10).
- `Shadow Map`: mapas de sombra renderizados para cada luz da cena.
- `Volumetric Light Image`: imagem contendo a iluminação volumétrica renderizada a partir de cada método, combinada com a `Radiance Image`.
- `Imagem Final`: imagem apresentada ao usuário ao fim do processo de renderização.

Figura 10 – Exemplo de *Radiance Image* obtido a partir da renderização base.
(a) Mapa de radiância **(b) Mapa de profundidade**

Fonte: Autoria própria (2025).

Descrição dos *shaders*:

- `Scene Rendering`: renderiza as malhas de triângulos e aplica efeitos de luz, sombra e textura na cena, utilizando traçado de raios.
- `Shadow Map Rendering`: renderiza os mapas de sombra de acordo com os parâmetros de luz armazenados em `Scene Data`.
- `TAA`: técnica de acumulação temporal, necessário para melhorar o resultado visual do traçado de raios aplicado pelo `shader Mesh Rendering`.

Na interface do motor gráfico, foi adicionado um menu para que o usuário possa escolher qual técnica de iluminação volumétrica será executada, também permitindo a opção de

desativar as luzes volumétricas. Independente da técnica selecionada, a renderização de luzes volumétricas ocorre onde está o retângulo vermelho `Volumetric Light Method` na Figura 9.

A renderização do mapa de sombras, que ocorre no *shader* `Shadow Map Rendering` foi implementada percorrendo todas as luzes e, para cada uma delas, rasterizando todos os objetos da cena. Nesta etapa foi utilizado o descarte das faces frontais (*front face culling*) da API gráfica, garantindo que apenas as superfícies voltadas para a luz sejam consideradas no teste de visibilidade (teste para definir objetos visíveis na cena). Essa abordagem é útil para evitar artefatos de auto-sombreamento, como o *shadow acne*, em que, devido à precisão limitada do mapa de profundidade, uma superfície iluminada acaba projetando sombra sobre si mesma (Scherzer; Wimmer; Purgathofer, 2011).

3.2 Métodos de Iluminação Volumétrica

As técnicas de iluminação volumétrica implementadas neste trabalho foram selecionadas buscando cobrir uma diversidade de estratégias de solução para o problema. Neste sentido, foram escolhidos: um método baseado em pós-processamento de imagens (Mitchell, 2007), um baseado em rasterização de geometria (Billeter; Sintorn; Assarsson, 2010), e um baseado em subdivisão do espaço (Wrónski, 2015).

Um fator comum a todas os métodos implementados é como é feita a combinação dos valores de radiância (renderização tradicional, obtido na etapa `Scene Rendering`) com os valores de irradiância (referentes à luz volumétrica). Independente do método selecionado, para um *pixel* $p = (i,j)$, o valor final de cor $L_f(p)$ é obtido somando a radiância base $L_b(p)$ à irradiância volumétrica $L_v(p)$, sendo assim, $L_f(p) = L_b(p) + L_v(p)$.

3.2.1 *Screen Space*

Esse método foi baseada no trabalho de Mitchell (2007), que utiliza aplica uma técnica pós-processamento para simular o efeito da luz volumétrica. A técnica é aplicada após a renderização da cena virtual, recebendo como entrada a imagem de radiância já renderizada e a descrição da cena, contendo os parâmetros de câmera e das luzes.

A implementação deste método foi feita em apenas um *shader* que é executado em paralelo para todos os *pixels* da imagem de radiância. O objetivo do *shader* é calcular L_v para cada pixel, e para isso é utilizado o modelo proposto por (Mitchell, 2007), de onde foi derivada a Equação 6, em que cada parâmetro é explicado a seguir.

$$L_v(p) = \text{exposure} \times \sum_{i}^{N} \text{decay}^i \times \text{weight} \times \frac{L_d(p_i)}{N} \tag{6}$$

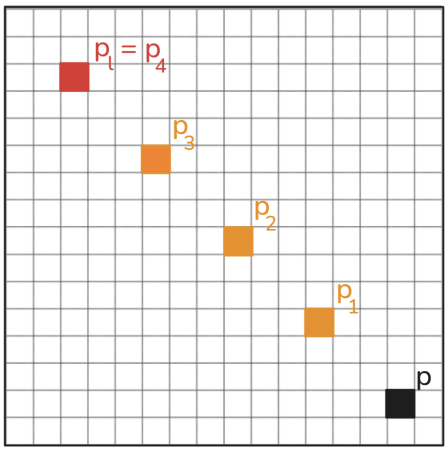
Dada uma determinada fonte de luz L, com posição p_l no espaço de tela, o cálculo da Equação 6 para determinado pixel p realiza um somatório de N amostras de iluminação

volumétrica. Essas amostras são dispostas uniformemente espaçadas entre p e p_l, sendo a última amostra $p_N = p_l$. Dessa forma, para uma amostra i, $p_i = p + i\Delta p$, em que Δp é obtido através da Equação 7.

$$\Delta p = \frac{p_l - p}{N} \tag{7}$$

A Figura 11 apresenta um exemplo de distribuição de amostras no caso de $N = 4$, em que o *pixel* vermelho representa a posição da luz, os laranjas representam as amostras, e o preto a posição onde a iluminação volumétrica está sendo calculada.

Figura 11 – Exemplo de posicionamento de amostras do método *Screen Space*.



Fonte: Autoria própria (2025).

A contribuição luminosa de uma amostra p_i, representada na Equação 6 por $L_d(p_i) = L_c \times V(p_i)$ é calculada a partir da intensidade da luz L_c e da visibilidade do ponto de amostragem $V(p_i)$. A visibilidade $V(p_i)$ é obtida a partir do mapa de profundidade, de acordo com a Equação 8.

$$V(p) = \begin{cases} 1, & \text{se } Depth(p) = 1 \text{ e } (0,0) \leq p \leq (W,H) \\ 0, & \text{caso contrário} \end{cases} \tag{8}$$

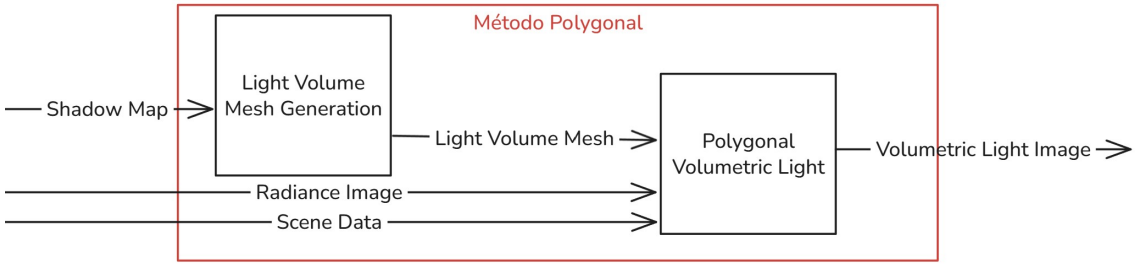
Portanto, a contribuição da amostra p_i é somada apenas se o pixel não estiver ocluído por nenhum outro objeto e estiver dentro dos limites do tamanho da imagem. Esse segundo teste é importante para casos em que a fonte de luz não está posicionada dentro do espaço de tela, por exemplo, quando a câmera está de costas para a luz.

Os demais parâmetros da Equação 6 foram propostos por (Mitchell, 2007), sendo eles: *exposure* é a intensidade geral do efeito, *decay* é o fator que dissipa a contribuição de cada amostra ao longo do somatório e *weight* controla a intensidade de cada amostra. Por fim, caso a cena seja constituída por mais que uma fonte de luz, o mesmo processo é repetido, calculando e acumulando L_v para cada uma das luzes.

3.2.2 *Polygonal*

Esse método foi baseado nos trabalhos de Hoobler (2016) e Billeter, Sintorn e Assarsson (2010), que utilizam o mapa de sombras para gerar uma malha de triângulos que encobre todo volume da luz. Um diagrama das etapas envolvidas na execução deste método está apresentado na Figura 12, fazendo referência ao diagrama apresentado anteriormente (Figura 9).

Figura 12 – Diagrama de etapas para execução do método *Polygonal*.

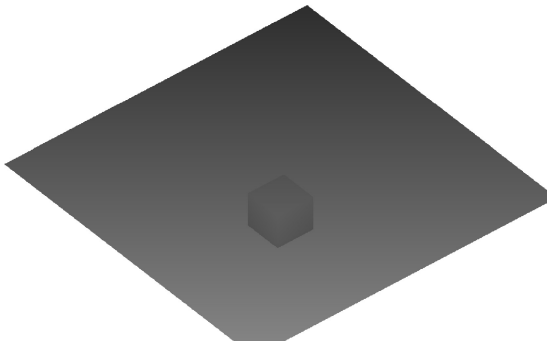


Fonte: Autoria própria (2025).

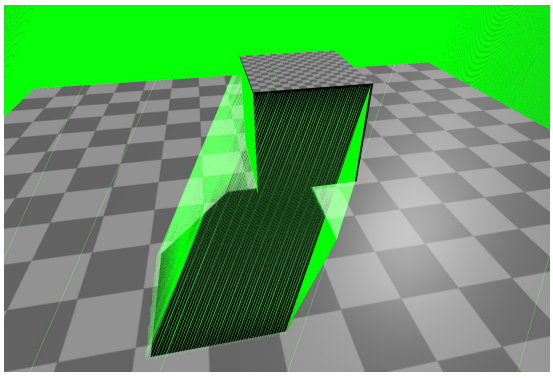
Sendo L_S o mapa de sombras de uma determinada luz L, com resolução W_S por H_S, inicialmente, aloca-se memória da placa de vídeo para armazenar uma malha com $W_S \times H_S$ *vértices*, em que cada vértice possui um pixel correspondente no mapa de sombras. Essa malha, representada no diagrama da Figura 12 `Light Volume Mesh` é uma malha de triângulos que encobre todo o volume iluminado por L. A Figura 13 apresenta um exemplo desta malha renderizado em modo *wireframe* (apenas as arestas) verde, assim como o mapa de sombras correspondente.

Figura 13 – Exemplo de malha gerada a partir do mapa de sombras.

(a) *Mapa de sombras*



(b) *Wireframe* da malha de volume de luz sobreposta à cena

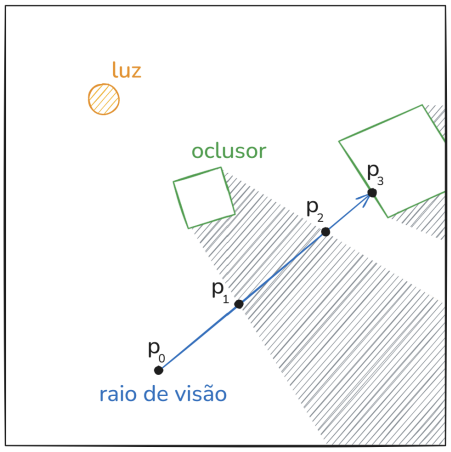


Fonte: Autoria própria (2025).

O *shader* `Light Volume Mesh Generation` é responsável por gerar essa malha, sendo executado em paralelo para cada vértice i da malha, a fim de calcular suas posições p_{iw} no espaço de mundo. Dessa forma, dado um vértice i, com pixel correspondente no mapa de sombras p_{is}, a posição p_{iw} é calculada através da reprojeção de coordenadas de tela em coordenadas de mundo (Equação 3).

Para explicar como é feito o cálculo da luz volumétrica a partir da malha gerada na etapa anterior, a Figura 14 apresenta um exemplo de cena, com pontos de interesse marcados em p_0, p_1, p_2 e p_3. Nessa cena, existe uma fonte de luz e um raio de visão, partindo do ponto p_0 (posição do observador) na direção de p_3. Os pontos p_1 e p_2 aparecem nas interseções entre a direção de visão e as regiões sombreadas da cena, representadas pelas partes hachuradas.

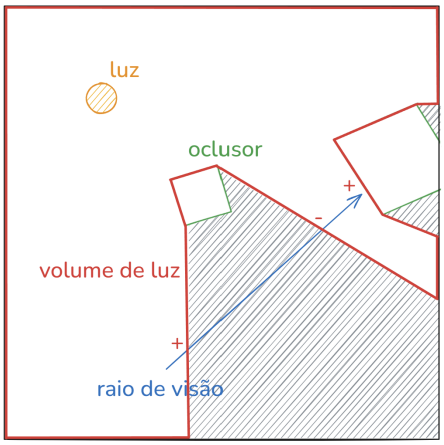
Figura 14 – Diagrama de exemplo para o método *Polygonal*.



Fonte: Autoria própria (2025).

A malha do volume de luz é gerada com base no mapa de sombras e delimita toda a região da cena que é iluminada pela fonte de luz em questão. No caso ilustrado pela Figura 14, todas as superfícies iluminadas, bem como as interfaces entre áreas iluminadas e sombreadas, são representadas por triângulos nessa malha. Esse volume de luz, que define toda a região iluminada da cena está destacada pelo contorno vermelho na Figura 15.

Figura 15 – Diagrama do volume de luz para o exemplo do método *Polygonal*.



Fonte: Autoria própria (2025).

Billeter, Sintorn e Assarsson (2010) apresentam um modelo para calcular a iluminação volumétrica $L_p(a,b)$ partindo de um ponto a até um ponto b, contabilizando apenas os segmentos de $\vec{ab}$ que estão dentro do volume de luz. Sendo assim, no caso do exemplo da Figura 14, $L_p(p_0, p_3) = L_p(p_0, p_1) + L_p(p_2, p_3)$.

A partir disso, Billeter, Sintorn e Assarsson (2010) derivam uma propriedade do modelo que permite simplificar o caso genérico de $L_p(a,b)$ como uma sequência de somas e subtrações de $L_p(a,p)$, para os diferentes pontos de interesse p ao longo de $\vec{ab}$. No caso $L_p(p_0,p_3)$ da Figura 14, passa a ser calculado como $L_p(p_0,p_3) = L_p(p_0,p_1) + L_p(p_0,p_3) - L_p(p_0,p_2)$. A decisão entre somar ou subtrair $L_p(a,b)$ para um determinado trecho depende se o ponto de interesse demarca uma região em que $\vec{ab}$ está entrando no volume de luz (subtração), ou saindo (soma), como exemplificado na Figura 15.

Utilizando essa propriedade, é possível rasterizar a malha de triângulos do volume de luz com um *fragment shader* (*shader* que executa para cada fragmento gerado a partir da malha), responsável por calcular $L_p(a,b)$. Nesse caso, o ponto de origem a é mantido fixo na posição da câmera p_v, para todos os fragmentos rasterizados. A contribuição luminosa $L_p(p)$, de cada fragmento em posição p, foi implementada através da equação 9, definida por Hoobler (2016).

$$L_p(p) = L_c \times \rho_m(\vec{d},\vec{l})\,\frac{1 - e^{-\tau_{ex}\|\vec{d}\|}}{\tau_{ex}} \tag{9}$$

Em que $\vec{d}$ é a direção de visão, definida por $(p - p_v)$, $\vec{l}$ a direção da luz, τ_{ex} o coeficiente de extinção do meio e ρ_m é a função de fase (Seção 2.1.4). Essa implementação também foi baseada em (Hoobler, 2016) que utiliza a mesma abordagem de (Billeter; Sintorn; Assarsson, 2010), e apresenta um modelo específico para o caso de luzes direcionais.

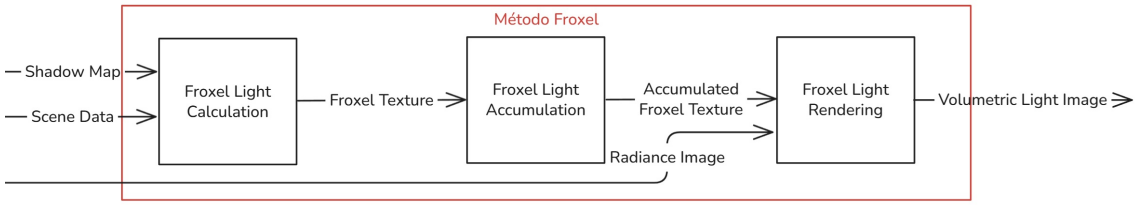
Dessa forma, o *shader* `Polygonal Volumetric Light` é responsável por rasterizar todos os triângulos do volume de luz, e acumular a contribuição luminosa L_p de cada fragmento, a fim de constituir o valor final de luz volumétrica L_v. Essa acumulação ocorre de acordo com a direção normal $\vec{n}$ de cada fragmento. No caso de $\vec{n} \cdot \vec{d} > 0$, então a contribuição é subtraída, pois representa um ponto em que o raio de visão está adentrando o volume de luz. Caso contrário, a contribuição é adicionada.

Um detalhe importante na implementação deste método foi desativar o teste de profundidade antecipado (*early depth test*) na API gráfica, pois é essencial que os triângulos sobrepostos sejam contabilizados. Por fim, assim como no método *Screen Space*, todas as etapas são repetidas no caso de múltiplas fontes de luz, desde a geração do volume de luz, até a combinação do valor final de L_v com a radiância base L_b.

3.2.3 *Froxel*

Esse método, baseado no trabalho de Wrónski (2015), utiliza como ideia principal subdividir o espaço da cena em uma grade 3D de elementos. Nesse caso, o espaço subdividido é o *frustum* da câmera, portanto, cada elemento adquire o nome de *froxel*. As etapas envolvidas na execução deste método estão apresentadas pelo diagrama na Figura 16. Nessa figura, ambas as setas `Froxel Texture` e `Accumulated Froxel Texture` são texturas 3D alocadas na placa de vídeo, com resolução (W_f, H_f, D_f).

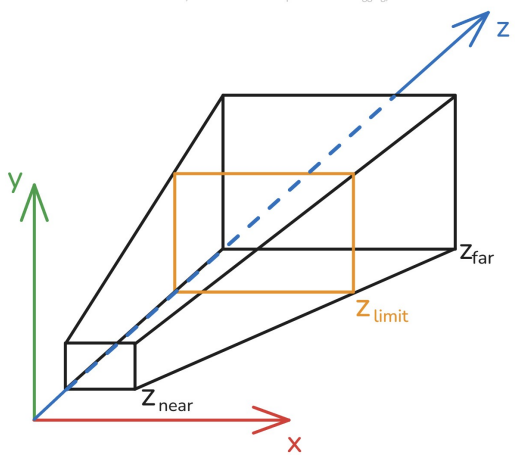
Figura 16 – Diagrama de etapas para execução do método *Froxel*.



Fonte: Autoria própria (2025).

Inicialmente é feita a subdivisão de uma região do *frustum* da câmera em uma grade de *froxels*, em que cada *froxel* possui um índice (i,j,k) de (0,0,0) até (W_f,H_f,D_f). Os *froxels* que compartilham um mesmo índice k, são chamados de camada k do volume de *froxels*. A região do *frustum* da câmera a ser subdividido parte de z_{near} (parâmetro de câmera) e é delimitada pelo parâmetro z_{limit}, assim como demonstrado na Figura 17.

Figura 17 – Diagrama contendo seção do *frustum* subdividida em *froxels*.



Fonte: Autoria própria (2025).

Nos eixos X e Y, os *froxel* são distribuídos uniformemente ao longo da largura e altura do *frustum*. Enquanto no eixo Z, os *froxels* são divididos uniformemente em valores de profundidade no espaço de recorte, ou seja, a coordenada z_k no espaço de câmera de uma camada k é calculada através da Equação 10.

$$z_k = \mathbf{M}_{\text{view}}^{-1} \cdot \mathbf{M}_{\text{proj}}^{-1} \cdot \left(k \frac{\mathbf{M}_{\text{proj}} \cdot \mathbf{M}_{\text{view}} \cdot (0,0,z_{\text{limit}})}{D_f} \right) \tag{10}$$

Então, o primeiro *shader* (`Froxel Light Calculation`) é executado em paralelo para todos os *froxels* da grade. O objetivo desta etapa é calcular a contribuição de luz volumétrica de cada *froxel* individualmente. Para tanto, dado um *froxel* f_{ijk} centrado na posição p_f no espaço de mundo, e uma fonte de luz L da cena, a luz volumétrica $L_l(p_f)$ é calculada através da Equação 11 e acumulada na textura de *froxels*. Essa equação foi implementada a partir do modelo proposto por Wrónski (2015).

$$L_l(p_f) = L_d(p_l,p_f)\rho_m(\vec{d},\vec{l})C_s\rho_l D(k) \tag{11}$$

Na Equação 11, $L_d(p_l,p_f)$ é a luz direta incidente no ponto p_f, calculada como a intensidade da luz L_c multiplicado pelo fator de sombra Shadow$_L(p_f)$. Além de $\vec{d} = (p_f - p_v)$ como direção de visão e $\vec{l}$ direção da luz, os demais termos utilizados são:

- $\rho_m(\vec{d},\vec{l})$ calculado através função de fase;
- C_s constante de *scattering* da partícula participante no meio de transmissão da luz;
- ρ_l é a densidade do meio;
- $D(k) = z_{k+1} - z_k$ é a profundidade da camada k, calculada a partir da equação 10;

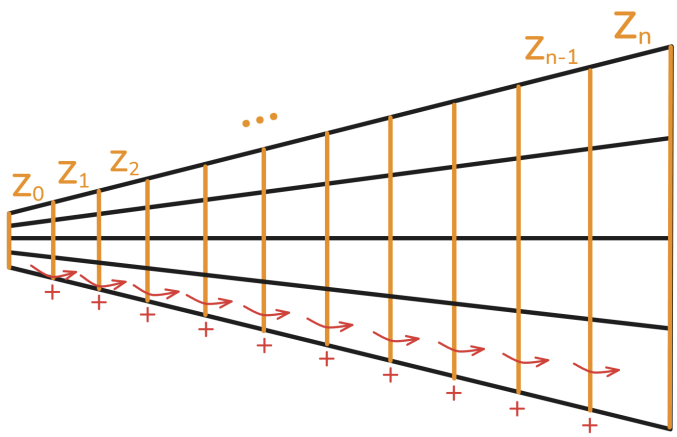
Dessa forma, o *shader* `Froxel Light Calculation` realiza o mesmo processo para todas as luzes da cena, a fim de calcular a contribuição total $L_T(p_{ijk})$ do *froxel* em f_{ijk} e armazená-la na textura `Froxel Texture`.

Após o término dessa etapa, o *shader* `Froxel Light Accumulation` é executado em paralelo para os $W_f \times H_f$ da primeira camada do volume. Esse *shader* percorre todas as D_f camadas, partindo da primeira até a última, acumulando os valores calculados na etapa anterior e armazenando-os na textura 3D `Accumulated Froxel Texture`. Esse processo pode ser representado pela Equação 12, que utiliza recursão para descrever o valor acumulado de L_{acc} a partir de um *froxel* f_{ijk} com centro em p_{ijk}.

$$L_{\text{acc}}(f_{ij}, k) = \begin{cases} L_T(p_{ijk}), & \text{se } k = 0 \\ L_{\text{acc}}(f_{ij}, k-1) + L_T(p_{ijk}), & \text{caso contrário} \end{cases} \tag{12}$$

A Figura 18 apresenta um diagrama (com visão lateral do *frustum*) que exemplifica como é feita a acumulação ao longo das camadas, partindo da camada Z_0 até a camada Z_n, *froxels* com mesmos índices i e j são acumulados em sequência.

Figura 18 – Diagrama exemplificando acumulação de camadas de *froxels* do método *Polygonal*.



Fonte: Autoria própria (2025).

Por fim, o shader `Froxel Light Rendering` é executado em todos os *pixels* da imagem de saída (que possui mesma resolução que `Radiance Image`). Cada *pixel* tem

suas coordenadas reprojetadas no espaço do mundo (Equação 3), identificando o *froxel* mais próximo correspondente. A contribuição acumulada daquele *froxel* é então amostrada e somada à `Radiance Image`, compondo o resultado L_v do método.

4 RESULTADOS E DISCUSSÃO

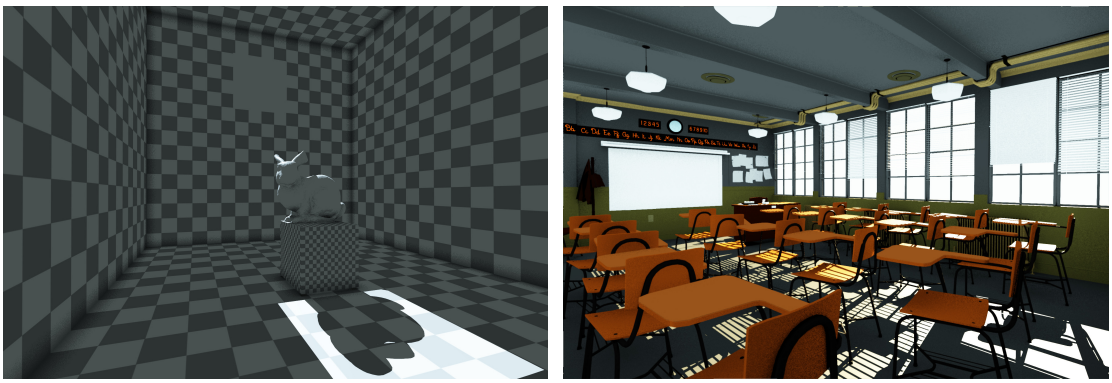
Neste capítulo, são apresentados parâmetros e cenas usadas para avaliar os métodos de iluminação volumétrica implementados. Após isso, resultados quantitativos são apresentados e discutidos, seguidos por uma análise qualitativa dos métodos. Todos os experimentos quantitativos foram realizados realizados em um computador com processador AMD Ryzen 9 7940HS, 16 GB de memória RAM e placa de vídeo NVIDIA RTX 4060 (Laptop).

4.1 Cenas

Os métodos de iluminação volumétrica foram analisados usando três cenas de teste. A Figura 19 apresenta as cenas com todos os efeitos visuais calculados pelo motor gráfico, excetuando a iluminação volumétrica. Para cada uma das cenas, uma câmera foi posicionada manualmente, permitindo replicar os testes automaticamente com cada um dos métodos de iluminação volumétrica.

Figura 19 – Cenas utilizadas para as análises experimentais.

(a) *Bunny* **(b) *Classroom***



(c) *San Miguel (exposição aumentada para melhorar visualização)*

Fonte: Autoria própria (2025).

A Tabela 1 apresenta informações sobre cada uma das cenas utilizadas, detalhando a complexidade das cenas conforme o número de triângulos. Todas as cenas são compostas por múltiplas malhas 3D texturizadas.

Tabela 1 – Detalhes das cenas de teste.

Cena	Número de triângulos	Fonte
Bunny Box	4968	(Laboratory, 1994)
Classroom	607402	(Seux, 2019)
San Miguel	5608441	(Llaguno *et al.*, 2017)

Fonte: Autoria própria (2025).

4.2 Parâmetros utilizados

Os parâmetros técnicos utilizados durante os testes estão listados na Tabela 2, se mantendo constantes para todas as cenas e testes realizados. Os parâmetros visuais, específicos de cada método implementado, estão listados nas tabelas 3, 4 e 5, respectivamente para os métodos *Screen Space*, *Polygonal* e *Froxel*. Nota-se que apenas os parâmetros puramente estéticos variam de acordo com a cena, pois foram ajustados manualmente a fim de aproximar os resultados visuais de cada método.

Tabela 2 – Parâmetros técnicos utilizados nos testes.

Parâmetro	Valor
Resolução da aplicação	2134 x 1459
Resolução dos mapas de sombras	1024 x 1024
Resolução das texturas 3D do método *Froxel*	190 x 90 x 128
Quantidade de amostras do método *Screen Space*	64

Fonte: Autoria própria (2025).

Tabela 3 – Parâmetros visuais utilizados no método *Screen Space* para diferentes cenas de teste.

Parâmetro	Bunny	Classroom	San Miguel
L_C (intensidade da luz)	10	50	4
`exposure`	1	1	1
`decay`	0.94	0.9	0.94
`weight`	0.01	0.002	0.01

Fonte: Autoria própria (2025).

Tabela 4 – Parâmetros visuais utilizados no método *Polygonal* para diferentes cenas de teste.

Parâmetro	Bunny	Classroom	San Miguel
L_C (intensidade da luz)	10	50	4
g (constante anisotrópica)	0.4	0.5	0
τ_{ex} (coeficiente de extinção)	0.3	$1.99\text{e}{-5}$	$3.99\text{e}{-5}$

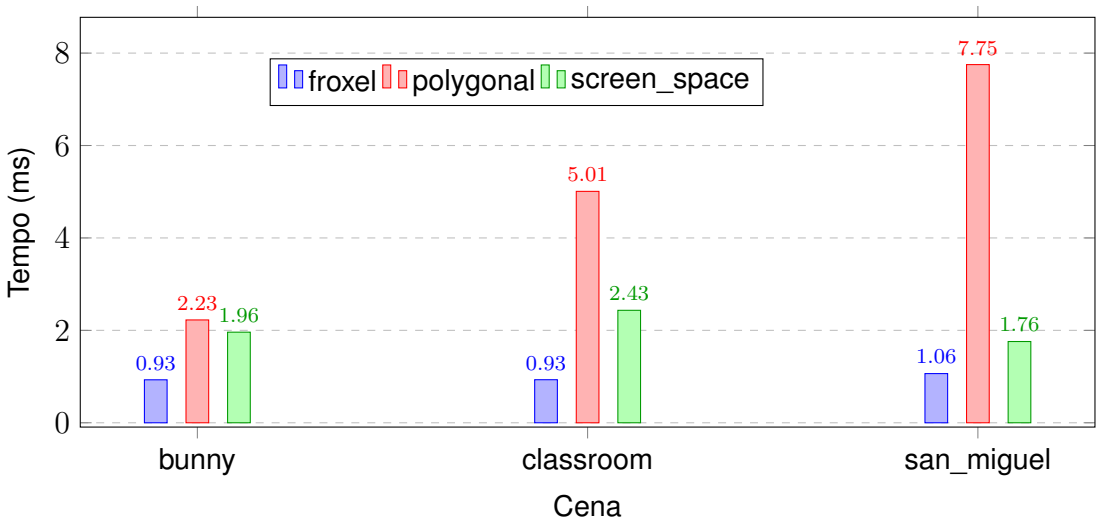
Fonte: Autoria própria (2025).

Tabela 5 – Parâmetros visuais utilizados no método *Froxel* para diferentes cenas de teste.

Parâmetro	Bunny	Classroom	San Miguel
L_C (intensidade da luz)	10	50	4
g (constante anisotrópica)	0.4	0.5	0.25
z_{limit}	17.6	11	40
ρ (densidade do meio)	1	0.1	0.02
C_s (constante de *scattering*)	0.1	0.18	2

Fonte: Autoria própria (2025).

Figura 20 – Gráfico de tempo de execução de cada algoritmo nas diferentes cenas testadas, considerando uma única fonte de luz.



Fonte: Autoria própria (2025).

4.3 Análise de tempo com uma fonte de luz

Primeiramente, os algoritmos foram testados em cenas com apenas uma fonte de luz, para analisar seus devidos comportamentos ao variar a complexidade geométrica das cenas. Nesses testes, cada algoritmo teve seu tempo de execução aferido para a execução de apenas um *frame*. Os resultados podem ser vistos no gráfico da Figura 20.

Analisando os resultados exibidos na Figura 20, é possível observar que o algoritmo *Froxel* apresentou os melhores tempos de execução em todas as cenas, sendo pouco influenciado pelo número de triângulos. O algoritmo *Polygonal* obteve a pior performance, demonstrando-se mais sensível às variações de número de triângulos. Quanto ao algoritmo *Screen Space*, por não depender da geometria da cena, não apresentou correlação direta com o tamanho das cenas.

A Tabela 6 utiliza os resultados da Figura 20 para demonstrar de forma prática quanto cada método impactou a taxa de quadros por segundo da aplicação. Pode-se observar que o pico de tempo de execução apresentado pelo método *Polygonal* na cena *San Miguel* representaria uma taxa de 34 FPS, o que estaria muito próximo do limiar aceitável (30 FPS) para uma boa experiência visual. Ainda na cena *San Miguel*, por sua maior complexidade geométrica,

os demais métodos, *Froxel* e *Screen Space*, não ocasionaram grandes quedas de FPS, pois a maior parte do custo de processamento dessa cena diz respeito à renderização base (luzes, sombras e texturas).

Tabela 6 – Impacto em quadros por segundo causado por cada método.

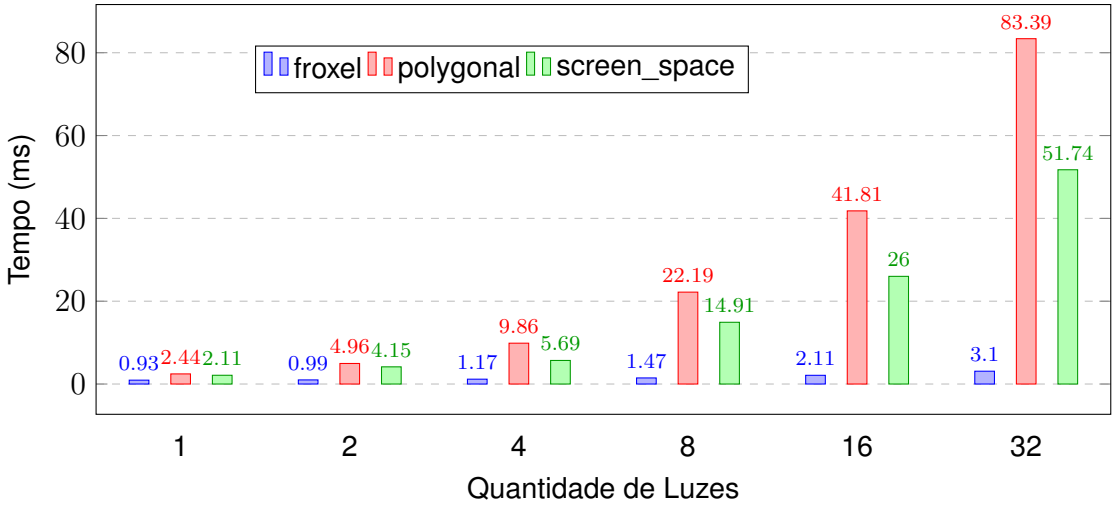
Cena	Desativado	Método *Froxel*	Método *Screen Space*	Método *Polygonal*
Bunny Box	156	136	119	116
Classroom	103	94	83	68
San Miguel	46	44	43	34

Fonte: Autoria própria (2025).

4.4 Análise de tempo e memória com múltiplas fontes de luz

O segundo conjunto de testes foi composto apenas pela cena *Bunny*, porém variando a quantidade de luzes na cena. Foram feitos testes para 1, 2, 4, 8, 16 e 32 fontes de luz, posicionadas proceduralmente na cena. O resultado pode ser visto nas Figura 21 e 22, que apresentam, respectivamente, o tempo de execução e o consumo de memória para cada algoritmo.

Figura 21 – Gráfico de tempo de execução de cada algoritmo variando a quantidade de luzes na cena.

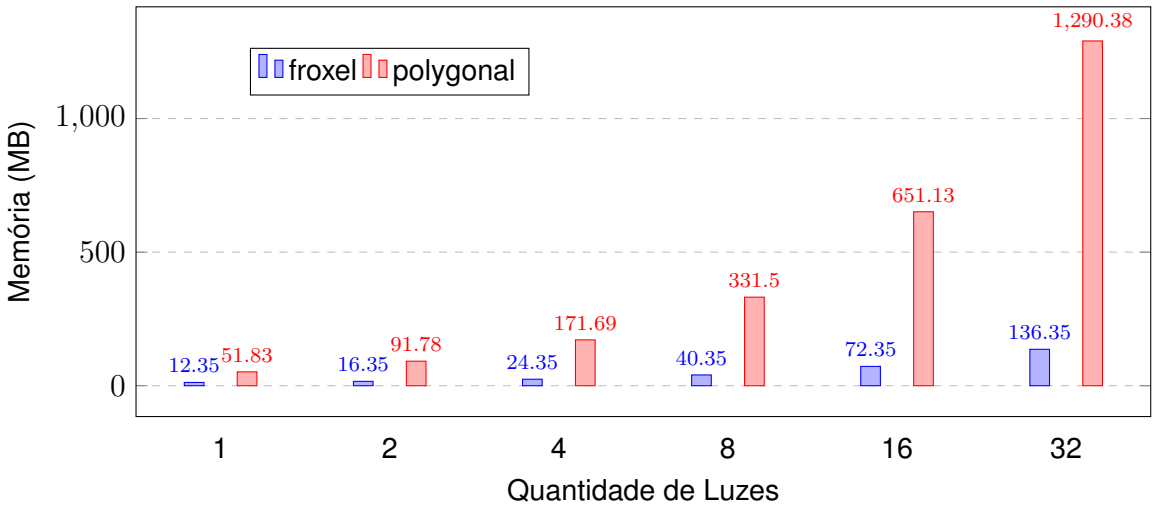


Fonte: Autoria própria (2025).

Ao analisar os resultados exibidos nas Figuras 21 e 22, pode-se perceber que, tanto no tempo de execução quanto no consumo de memória, todos os métodos respondem linearmente ao número de luzes.

O método *Screen Space* não é apresentado nos teste de memória da Figura 22, pois não requer a alocação de memória para nenhuma imagem ou estrutura de dados, realizando os cálculos na própria imagem da cena, *in-place*. Enquanto isso, o método *Polygonal* necessita alocar memória para os mapas de sombras e malhas de volume de luz, sendo um par para cada luz na cena. De modo similar, o método *Froxel* também utiliza memória referente aos mapas de sombra, porém o volume 3D onde são armazenados os *froxels* é único para a cena inteira.

Figura 22 – Gráfico de uso de memória de cada algoritmo variando a quantidade de luzes na cena.



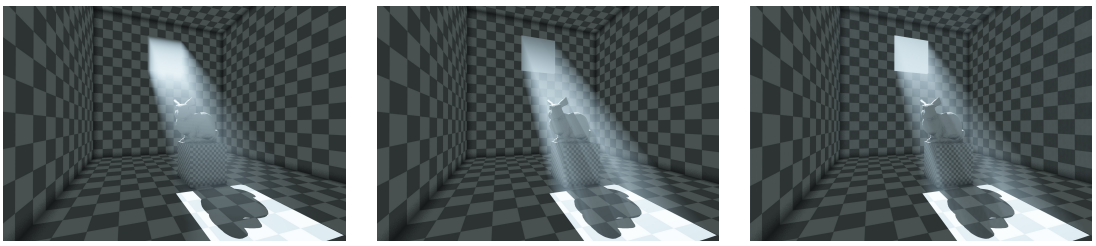
Fonte: Autoria própria (2025).

4.5 Análise qualitativa

Para realizar a análise qualitativa, as Figuras 23 a 25 apresentam as imagens renderizadas das cenas de teste utilizando cada método de iluminação volumétrica. Todas as cenas são renderizadas fixando a fonte de luz e a câmera. Desta forma, é possível analisar comparativamente os resultados produzidos por cada método de iluminação volumétrica.

Figura 23 – Render da cena *Bunny* com diferentes métodos de iluminação volumétrica

(a) Método *Screen Space* **(b) Método *Polygonal*** **(c) Método *Froxel***



Fonte: Autoria própria (2025).

Figura 24 – Render da cena *Classroom* com diferentes métodos de iluminação volumétrica

(a) Método *Screen Space* **(b) Método *Polygonal*** **(c) Método *Froxel***

Fonte: Autoria própria (2025).

Figura 25 – Render da cena *San Miguel* com diferentes métodos de iluminação volumétrica

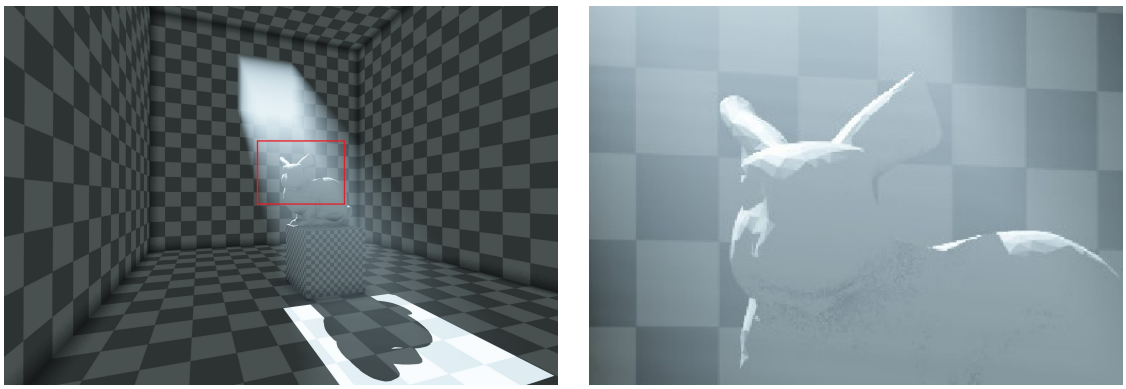
(a) Método *Screen Space* **(b) Método *Polygonal*** **(c) Método *Froxel***

Fonte: Autoria própria (2025).

O método *Screen Space* apresenta uma inconsistência visual evidente, pois, devido à sua natureza indiferente à geometria da cena, acaba sendo incapaz de calcular as áreas da luz volumétrica que deveriam ser ocluídas por objetos. Este efeito é bastante notável ao comparar os resultados da Figura 23. Enquanto os métodos *Polygonal* e *Froxel* representam o objeto obstruindo a passagem da iluminação (Figuras 23b e 23c), o método *Screen Space* não produz o mesmo efeito. A Figura 26 mostra em detalhe que o caminho da luz volumétrica não sofre a interferência do coelho na cena.

Figura 26 – Artefato visual do método *Screen Space* em que a luz volumétrica sobrepõem o objeto oclusor.

(a) Resultado do método *Screen Space* na cena Bunny **(b) Recorte da região com artefato visual**



Fonte: Autoria própria (2025).

Um segundo artefato visual presente no método *Screen Space* ocorre devido à amostragem de *pixels* utilizada no somatório da técnica, que acumula valores ao longo do trajeto da luz. Esse processo pode introduzir padrões artificiais, especialmente em regiões com geometrias finas ocluindo o céu. Um exemplo desse artefato aparece na Figura 27, onde é visível um padrão de *aliasing* na luz volumétrica que atravessa os pequenos vãos entre as folhas do topo da árvore.

Figura 27 – Artefato visual do método *Screen Space* em que a luz volumétrica apresenta descontinuidades.

(a) Resultado do método *Screen Space* na cena San Miguel **(b) Recorte da região com artefato visual**

Fonte: Autoria própria (2025).

Mitchell (2007) sugere algumas abordagens para mitigar esse problema, como amostragem estocástica e redução da resolução antes da aplicação da técnica. No entanto, essas abordagens não são definitivas, pois o artefato decorre de limitações fundamentais da amostragem discreta, conforme descrito pelo teorema de Nyquist-Shannon, que estabelece que a frequência de amostragem deve ser ao menos o dobro da frequência do sinal para evitar *aliasing* (Shannon, 1949). No caso do exemplo da Figura 27, a frequência do sinal é definida pelo tamanho do vão entre as folhas, e a frequência de amostragem é definida pelo espaçamento entre as amostras (Equação 7).

Para fins de comparação com uma aplicação comercial, a Figura 28 apresenta uma captura de tela do jogo *Bloodborne*[1], exibindo um artefato visual muito semelhante ao observado na técnica *Screen Space*. Ressalta-se essa imagem foi equalizada digitalmente para melhorar a visibilidade do efeito.

[1] *Bloodborne*™ é um jogo eletrônico desenvolvido pela *FromSoftware* e publicado em 2015 pela *Sony Computer Entertainment* para o console *PlayStation 4*. Marca registrada de seus respectivos detentores.

Figura 28 – Captura de tela do jogo *Bloodborne* apresentando artefato visual. Brilho e exposição ajustados digitalmente para melhor visualização.

(a) Captura de tela **(b) Recorte da região com artefato visual**

Fonte: *Bloodborne* (2015).

O método *Polygonal* apresentou os resultados mais consistentes, sem grandes artefatos visíveis. Até mesmo na cena *Classroom*, apresentada na Figura 29, a luz passando por entre as frestas da persiana na janela cria um padrão de detalhes pequenos na luz volumétrica, que foram corretamente representados pelo método *Polygonal*.

Figura 29 – Detalhes da persiana na luz volumétrica do método *Polygonal*.

(a) Resultado do método *Polygonal* na cena Classroom. **(b) Recorte da região com detalhes da persiana.**

Fonte: Autoria própria (2025).

Porém, mesmo com esses bons resultados visuais, o método *Polygonal* ainda depende do mapa de sombras, que possui uma resolução limitada. No caso de detalhes de sombra menores que um pixel, ou algum padrão que cause *aliasing*, esses artefatos estariam presentes na luz volumétrica gerada pelo método. A mesma limitação relacionada ao mapas de sombra ocorre com o método *Froxel*, porém, os artefatos são amplificados pela resolução da textura 3D utilizada. Ao calcular a luz volumétrica para cada *texel* dessa textura, que cobre uma região do *frustum*, também pode haver *aliasing*. Além disso, por se tratar de uma textura 3D, a resolução é ainda mais restrita, devido ao maior uso de memória. Esses artefatos, devido à resolução limi-

tada e ao *aliasing*, são bem evidentes na Figura 24c, e estão ampliados no recorte apresentado na Figura 30.

Figura 30 – Artefato de baixa resolução visível no método *Froxel*.

(a) Resultado do método *Froxel* na cena Classroom **(b) Recorte da região com artefato visual**

Fonte: Autoria própria (2025).

Esse mesmo tipo de artefato, relacionado à baixa resolução da textura 3D, pode ser observado em diversos jogos recentes desenvolvidos com a *Unreal Engine*, que também utiliza um método de iluminação volumétrica baseado em *voxel* (Epic Games, 2024). Um exemplo está presente na Figura 31, que exibe uma captura de tela do jogo *Black Myth: Wukong*[2], feito na *Unreal Engine*.

Figura 31 – Captura de tela do jogo *Black Myth: Wukong* apresentando artefato visual de iluminação volumétrica

(a) Captura de tela **(b) Recorte da região com artefato visual**

Fonte: *Black Myth: Wukong* (2024).

[2] *Black Myth: Wukong*™ é um jogo eletrônico, publicado em 2024, desenvolvido pelo estúdio *Game Science*, utilizando a *Unreal Engine 5*. Marca registrada de seus respectivos detentores.

5 CONCLUSÃO

Neste trabalho foram analisadas três abordagens distintas para simulação de luz volumétrica em tempo real: o método baseado em pós-processamento de imagem (*Screen Space*), o método geométrico (*Polygonal*) e o método de voxelização do *frustum* (*Froxel*). A implementação e experimentação de cada técnica permitiram uma análise comparativa considerando critérios como desempenho, uso de memória, fidelidade visual e complexidade de implementação.

Os resultados indicam que o método *Froxel* apresenta o melhor desempenho em termos de tempo de execução, sendo pouco impactado pela complexidade geométrica da cena. O método *Screen Space* possui implementação simples e uso reduzido de memória, mas apresenta limitações visuais perceptíveis, como artefatos de oclusão e aliasing. Por outro lado, o método *Polygonal* demonstrou a maior fidelidade visual, sendo capaz de representar detalhes finos da iluminação volumétrica, embora com custo computacional elevado e maior uso de memória.

Cada método possui características próprias que devem ser consideradas conforme os requisitos da aplicação. Em situações que exigem alta qualidade visual e toleram maior custo de processamento, o método *Polygonal* é mais apropriado. Para aplicações que priorizam desempenho e escalabilidade, o método *Froxel* apresenta uma solução eficiente e equilibrada. Já o método *Screen Space* pode ser útil em contextos onde os efeitos visuais são discretos e os recursos computacionais são limitados.

Como possibilidade de continuação deste trabalho, sugere-se a investigação de abordagens híbridas que combinem os pontos fortes das técnicas analisadas. Também é recomendada a aplicação de algoritmos de amostragem adaptativa, reprojeção temporal e técnicas de *upsampling* para redução de artefatos visuais e otimização da performance.

REFERÊNCIAS

BARANOSKI, G. V.; KRISHNASWAMY, A. **Light and skin interactions: simulations for computer graphics applications**. [*S.l.*]: Morgan Kaufmann, 2010.

BILLETER, M.; SINTORN, E.; ASSARSSON, U. Real time volumetric shadows using polygonal light volumes. *In*: HIGH PERFORMANCE GRAPHICS. 2010. **Anais [...]** [*S.l.*: *s.n.*], 2010. p. 39–45.

BILLETER, M.; SINTORN, E.; ASSARSSON, U. Real-time multiple scattering using light propagation volumes. *In*: PROCEEDINGS OF THE ACM SIGGRAPH SYMPOSIUM ON INTERACTIVE 3D GRAPHICS AND GAMES (I3D). 2012. **Anais [...]** [*S.l.*]: ACM, 2012. p. 119–126.

CLAYPOOL, M.; CLAYPOOL, K.; DAMAA, F. The effects of frame rate and resolution on users playing first person shooter games. *In*: SPIE. MULTIMEDIA COMPUTING AND NETWORKING 2006. 6071., 2006. **Anais [...]** [*S.l.*], 2006. p. 607101.

Epic Games. **Volumetric Fog in Unreal Engine**. [*S.l.*]: , 2024. https://dev.epicgames.com/documentation/en-us/unreal-engine/volumetric-fog-in-unreal-engine. Acesso em: 25 jun. 2025.

GIBSON, W. **Neuromancer**. New York: Ace Books, 1984. ISBN 9780441569595.

HENYEY, L. G.; GREENSTEIN, J. L. Diffuse radiation in the galaxy. **The Astrophysical Journal**, The University of Chicago Press v. 93„ p. 70–83, 1941.

HOOBLER, N. **Fast, Flexible, Physically-Based Volumetric Light Scattering**. [*S.l.*]: , 2016. Game Developers Conference.

IVKOVIC, Z. *et al.* Quantifying and mitigating the negative effects of local latencies on aiming in 3d shooter games. *In*: PROCEEDINGS OF THE 33RD ANNUAL ACM CONFERENCE ON HUMAN FACTORS IN COMPUTING SYSTEMS. 2015. **Anais [...]** [*S.l.*: *s.n.*], 2015. p. 135–144.

KAJIYA, J. T. The rendering equation. *In*: PROCEEDINGS OF THE 13TH ANNUAL CONFERENCE ON COMPUTER GRAPHICS AND INTERACTIVE TECHNIQUES. 1986. **Anais [...]** [*S.l.*: *s.n.*], 1986. p. 143–150.

KARLSSON, A. *et al.* Numerical simulations of light scattering by red blood cells. **IEEE Transactions on Biomedical Engineering**, IEEE v. 52, n. 1, p. 13–18, 2004.

KOVALOVS, A. Volumetric effects of the last of us: Part two. *In*: ACM SIGGRAPH 2020 TALKS. 2020, New York, NY, USA. **Anais [...]** New York, NY, USA: Association for Computing Machinery, 2020. ISBN 9781450379717. Disponível em: https://doi.org/10.1145/3388767.3407393.

LABORATORY, S. U. C. G. **The Stanford 3D Scanning Repository – Stanford Bunny**. [*S.l.*]: , 1994. Uso acadêmico livre. Acesso em: 23 jun. 2025. Disponível em: https://graphics.stanford.edu/data/3Dscanrep/.

LAMBRU, C. *et al.* Comparative analysis of real-time global illumination techniques in current game engines. **IEEE Access**, IEEE v. 9„ p. 125158–125183, 2021.

LLAGUNO, G. M. L. *et al.* **San Miguel 2.0 Scene**. [*S.l.*]: , 2017. Licença: Creative Commons Attribution 3.0 (CC BY 3.0). Acesso em: 23 jun. 2025. Disponível em: https://casual-effects.com/data/.

MITCHELL, K. **Volumetric Light Scattering as a Post-Process**. USA: Addison-Wesley Professional, 2007. (GPU Gems 3).

PHARR, M.; JAKOB, W.; HUMPHREYS, G. **Physically Based Rendering**: From thoery to implementation. Forth. USA: MIT Press, 2023.

SCHERZER, D.; WIMMER, M.; PURGATHOFER, W. A survey of real-time hard shadow mapping methods. *In*: WILEY ONLINE LIBRARY. COMPUTER GRAPHICS FORUM. 30 n. 1., 2011. **Anais [...]** [*S.l.*], 2011. p. 169–186.

SEUX, C. **Classroom Scene – Blender Demo Files**. [*S.l.*]: , 2019. Licença: Creative Commons Zero (CC0). Acesso em: 23 jun. 2025. Disponível em: https://www.blender.org/download/demo-files/.

SHANNON, C. Communication in the presence of noise. **Proceedings of the IRE**, v. 37, n. 1, p. 10–21, 1949.

STAMMINGER, M.; DRETTAKIS, G. Perspective shadow maps. *In*: PROCEEDINGS OF THE 29TH ANNUAL CONFERENCE ON COMPUTER GRAPHICS AND INTERACTIVE TECHNIQUES. 2002. **Anais [...]** [*S.l.*: *s.n.*], 2002. p. 557–562.

WRÓNSKI, B. **Volumetric Fog and Lighting**. USA: A K Peters, 2015. (GPU Pro 6: Advanced Rendering Techniques).

ZHOU, K. *et al.* Real-time smoke rendering using compensated ray marching. *In*: **ACM SIGGRAPH 2008 papers**. [*S.l.*: *s.n.*] 2008. p. 1–12.