Pontifícia Universidade Católica
do Rio de Janeiro

**Leonardo Quatrin Campagnolo**

# Interactive Directional Occlusion Shading and Black Oil Reservoir Visualization using Ray Casting



**Tese de Doutorado**

Thesis presented to the Programa de Pós–graduação em Informática of PUC-Rio in partial fulfillment of the requirements for the degree of Doutor em Informática.

Advisor: Prof. Waldemar Celes Filho

Rio de Janeiro
April 2021

Pontifícia Universidade Católica
do Rio de Janeiro

**Leonardo Quatrin Campagnolo**

# Interactive Directional Occlusion Shading and Black Oil Reservoir Visualization using Ray Casting

Thesis presented to the Programa de Pós–graduação em Informática of PUC-Rio in partial fulfillment of the requirements for the degree of Doutor em Informática. Approved by the Examination Committee.

**Prof. Waldemar Celes Filho**
Advisor
Departamento de Informática – PUC-Rio

**Prof. Marcelo Gattass**
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. Manuel Menezes de Oliveira Neto**
Universidade Federal do Rio Grande do Sul – UFRGS

Rio de Janeiro, April 9th, 2021

**Leonardo Quatrin Campagnolo**

Bachelor's in Computer Science (2012) at the Federal University of Santa Maria (UFSM). Masters' in Informatics (2015) at PUC-Rio with emphasis on Volumetric Visualization. Works at Tecgraf/PUC-Rio.



Bibliographic data

Campagnolo, Leonardo Quatrin

Interactive Directional Occlusion Shading and Black Oil Reservoir Visualization using Ray Casting / Leonardo Quatrin Campagnolo; advisor: Waldemar Celes Filho. – Rio de janeiro: PUC-Rio , Departamento de Informática, 2021.

v., 89 f: il. color. ; 30 cm

Tese (doutorado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática.

Inclui bibliografia

1. Informática – Teses. 2. Visualização volumétrica;. 3. Iluminação volumétrica;. 4. Oclusão de Ambiente Direcional;. 5. Sombras;. 6. Visualização de reservatórios.. I. Celes, Waldemar. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

## Acknowledgments

I would like to thank...

Sabrina Dal Ongaro Savegnago, my wife, who supported me through all hardships during these years. For making me smile at the most complicated days, sharing all the happiness that a life can bring. It is impossible to put in words how important you are and i'll never be able to express how lucky i feel for being part of your life.

My parents, Antonio and Lisete for life and support through all these years and always cheering for me. To my brothers Angela and Fernando for all the support and conversation moments.

Prof. Waldemar Celes for the help and opportunities during all these years. For accepting to be my advisor on my both master and doctor's degree. For all the advices and guidance that turned me better as a professional and researcher.

The visualization group at Tecgraf, which i'm current working on. Thank you for all the moments during these years.

All my friends, thank you for being around during all these years. Thank you Luiz Schirmer, Guilherme Schardong, Fabrício Cardoso, Abner Cardoso, Luiz Felipe Netto for all the fun moments playing games and talking about nonsense things.

Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) for partially financing this research under grant 153737/2014-0.

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001.

## Abstract

Campagnolo, Leonardo Quatrin; Celes, Waldemar (Advisor). **Interactive Directional Occlusion Shading and Black Oil Reservoir Visualization using Ray Casting** . Rio de Janeiro, 2021. 89p. Tese de doutorado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Volume rendering is a widely used technique to visualize 3D scalar data. To enhance visual shape and depth perception, distinct illumination techniques have been proposed, adding different types of lighting effects. In this thesis, we explore a new strategy to compute directional ambient occlusion and shadows for volume ray casting. Our algorithm computes occlusion of traced cones by evaluating Gaussian integrals at discrete samples along the cone axis. The computed occlusion is then used to add directional ambient occlusion effects and to generate shadows. Given the extinction coefficient data volume, we create one extra volume computing representative amplitudes of Gaussian functions. Mipmapping is then used to effectively evaluate Gaussian integrals with different sizes placed along the cone axis, adapting a circle packing approach. We demonstrate that the proposed method delivers a better balance between quality results and performance when compared to previous specialized procedures, with the advantage of combining directional ambient occlusion and shadow generation under the same framework. We also explore three volume rendering algorithms for black oil reservoir models, represented by irregular hexahedral meshes with geometry distortions and discontinuities. These algorithms were implemented under a compact representation that stores the model in the GPU. We compare performance and image quality delivered by each strategy by running a set of experiments with different models. We then investigate the gain in perception when applying our technique to compute directional ambient occlusion effects. The algorithms were entirely implemented on graphics card to produce interactive visualizations.



## Keywords

Volume rendering; Volumetric illumination; Diretional Ambient Occlusion; Shadows; Black oil reservoir Visualization.

## Resumo

Campagnolo, Leonardo Quatrin; Celes, Waldemar. **Oclusão direcional e visualização volumétrica de reservatórios utilizando traçado de raios**. Rio de Janeiro, 2021. 89p. Tese de Doutorado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

A visualização volumétrica é uma técnica amplamente utilizada para visualizar dados escalares tridimensionais. Para melhorar a percepção de profundidade e forma, diversas técnicas de iluminação foram propostas, adicionando diferentes tipos de efeitos. Neste trabalho, foi explorada uma nova estratégia para calcular oclusão de ambiente direcional e sombras para *volume ray casting*. Ela consiste em avaliar a oclusão de um traçado de cone atra'ves de integrais Gaussianas posicionadas de maneira discreta ao longo do eixo do cone. O valor resultado é utilizado para adicionar oclusão de ambiente direcional e sombras. A partir dos coeficientes de extinção dados pela função de transferência, um volume extra é gerado computando amplitudes representativas de distribuições Gaussianas. O Mipmapping também é utilizado para avaliar de maneira efetiva integrais Gaussianas em diferentes tamanhos posicionadas ao longo do eixo principal do cone, adaptando uma estratégia de *circle packing in a circle*. Nos resultados, é demonstrado que o método proposto obteve um melhor balanço entre performance e qualidade, comparado com trabalhos propostos anteriormente, com a vantagem de combinar oclusão de ambiente direcional e sombras utilizando o mesmo *framework*. Em seguida, exploramos três estratégias de visualização volumétrica para reservatórios de petróleo, representados por malhas irregulares contendo distorções geométricas e descontinuidades. Estes algoritmos foram implementados a partir de uma representação compacta que guarda o modelo em GPU. Testes comparativos de performance e qualidade foram feitos utilizando diferentes modelos de reservatório. Por fim, investigamos o ganho de percepção ao adicionar a nossa proposta de oclusão de ambiente direcional. Os algoritmos foram todos implementados utilizando programação de shaders para capacitar a geração de visualizações interativas.

## Palavras-chave

Visualização volumétrica; Iluminação volumétrica; Oclusão de Ambiente Direcional; Sombras; Visualização de reservatórios.

# Table of contents


1 Introduction **14**
1.1 Interactive Directional Ambient Occlusion and Shadows 15
1.2 Experimental Study on Black Oil Reservoir Volume Visualization 17
1.3 Contributions and Overview 19

2 Related Work **20**
2.1 Volumetric Illumination 20
2.2 Unstructured Volume Rendering 22

3 Directional ambient occlusion and shadows for volume ray casting **24**
3.1 Directional Illumination Model 24
3.1.1 Ray Tracing 25
3.2 Proposed Transparency Computation 26
3.2.1 Gaussian distribution of extinction coefficients 26
3.2.2 Cone sampling 28
3.2.3 Use of the mipmap pyramid 30
3.2.4 Cone splitting 31
3.3 Directional Shading 32
3.3.1 Directional ambient occlusion 33
3.3.2 Shadow generation 33
3.3.3 Pre-computed transparency 34
3.4 Parameter Control 35
3.5 Results and Discussion 37
3.5.1 Directional ambient occlusion comparison 38
3.5.2 Shadow comparison 41
3.5.3 Pre-processing analysis 43
3.5.4 Combining directional ambient occlusion and shadows 44
3.5.5 Downscaling and Upscaling Filtering 45

4 An experimental study on volumetric visualization of black oil reservoir models **51**
4.1 Ray Casting Algorithms for Black Oil Reservoirs 51
4.1.1 Structured Resampled Grid 53
4.1.2 Linear Interpolation with Barycentric Coordinates 54
4.1.3 Point Location with Topological Search 56
4.2 Adapting Directional Ambient Occlusion Effects 57
4.3 Results and Discussion 58
4.3.1 Memory Comparison 59
4.3.2 Quality and Performance Comparison 60
4.3.3 Combining with Directional Ambient Occlusion 61
4.3.4 Pre-processing 62

5 Conclusions and future work **67**
5.1 Future Work 68

Bibliography **69**

A Implementation Details of Transparency Computation **76**

B Parameter Control **81**
B.1 Initial standard deviation: $\sigma_0$ 81
B.2 Initial step to avoid self-occlusion: $h_0$ 82
B.3 Cone apertures: $\theta_o$ and $\theta_s$ 83
B.4 Number of splittings: $C_{split}$ 84

C Additional Ground Truth Comparisons **86**

## List of figures

Figure 1.1 Volume rendering of VisMale dataset using different associated transfer functions. 15
Figure 1.2 Different approaches to computing ambient occlusion (AO). The first is applied to surface models; the last two suits better to volume rendering. 16
Figure 1.3 Cone placements for both directional ambient occlusion and shadow computations. The lighting effects are computed at each sample along the viewing rays. For directional ambient occlusion, we use cones with greater aperture angle, $\theta_o$, to capture the environment. For shadow computation, we use cones with smaller aperture, $\theta_s$, to generate hard shadows. 17
Figure 1.4 A cone-shaped flow of water revealed by the volume visualization. The image was rendered by adding directional ambient occlusion effects. 18

Figure 3.1 Our proposed approach for transparency computation. We first consider a volumetric integral instead of a bunch of rays (a), which is computed by adding consecutive samples (b). Each sample represents a Gaussian distribution (c), which is truncated by the cone domain (d) and evaluated along the $w$ direction (e). Finally, consecutive samples are placed along $w$ to approximate the whole cone transparency $T_\Psi(L)$ via numerical integration (f). 27
Figure 3.2 At each sample, the transparency on the $uv$ plane perpendicular to the cone axis is captured by a Gaussian integral. This integral is truncated by the cone surface: we limit the integral interval from $-r$ to $r$ in both $u$ and $v$ directions, where $r$ represents the circular section radius. As an example, limiting a 1D Gaussian integral from $-\sigma$ to $\sigma$ results in 68.27% of the full value. 29
Figure 3.3 Placement of successive extinction coefficient distributions along the cone axis, from left to right. The integral from the first to the last sample can be approximated by the trapezoid rule, considering the Gaussian amplitudes. 30
Figure 3.4 Example of cone sampling using $r_c = 2\sigma$. Each time the current section radius becomes larger than $2\sigma$, we increase the fetched pyramid level, doubling $\sigma$. The initial gap, $h_0$, is necessary to avoid self-occlusion. 31
Figure 3.5 The "circle packing in a circle" arrangement applied in our technique. For each split step, we must correctly update the current transparency of the new set of cones. 32
Figure 3.6 Types of shadows implemented using our cone tracing. When defining the position of a light source, we also store its orthogonal axes, composed by a forward, up and, right vectors, orienting the cones appropriately. 33

Figure 3.7 Types of shadows implemented using our cone tracing on Bonsai dataset. 34
Figure 3.8 Illustration of both evaluation models implemented with our proposed technique. For object space, an additional texture is generated and the transparency is calculated for each voxel before the rendering stage, being interpolated at intermediate positions. 34
Figure 3.9 Image quality comparison for the Synthetic model with cone angle aperture of 10°. 39
Figure 3.10 Image quality comparison for the VisMale dataset with cone angle aperture of 20°. 40
Figure 3.11 Image quality comparison for the VisMale(T) dataset with cone angle aperture of 25°. 41
Figure 3.12 Directional ambient occlusion results of our method for VisMale, CT-Knee, and Foot datasets, either in image space (left) and object (middle and right) space, with viewport set to $768^2$. 43
Figure 3.13 Shadow quality comparison of SyntheticBars dataset using a cone angle aperture of 0.5°. 47
Figure 3.14 Shadow quality comparison for Engine dataset using a cone angle aperture of 2.0°. 48
Figure 3.15 Hazelnut and Flower datasets rendered with directional ambient occlusion ($\theta_o = 25^o$), with different extinction coefficient volume resolutions. 49
Figure 3.16 Achieved results combining directional ambient occlusion and shadow generation: on the left, emission and absorption illumination model with directional occlusion; on the right, shadows with Blinn-Phong shading are added. 49
Figure 3.17 Results from VisMale, Bonsai, Hazelnut and Flower datasets, using a viewport of $768^2$ and considering the parameters from Section 3.5.1 for VisMale and Table 3.6 for the rest. For Downscaling, we render the image using a viewport of $1536^2$ and then scale back to $768^2$. For Upscaling, we render the image using a viewport of $384^2$, and then extend back to $768^2$. We use the Cardinal Cubic O-MOMS kernel implementation provided by [48], for both Downscaling and Upscaling operations. In this experiment, light is evaluated in object space only for Hazelnut dataset. 50

Figure 4.1 Example of reservoir model with geometry discontinuities. 52
Figure 4.2 Franceschin et al.'s proposal [21] starts the point location procedure by identifying the corresponding voxel ID from a regular grid; then, it applies a Newton-Raphson procedure to check if each cell intercepted by the voxels contains the point, returning its parametric coordinate, if the case. The low resolution regular grid drawn above is for illustrative purposes; in practice, we set $r = 1.75$, where each voxel ends up being smaller than the average cell size. 52

Figure 4.3 We create an additional 2-channel 3D texture to store the property value and $\alpha_p$, which is multiplied by the current opacity along each ray in the rendering stage. This image is illustrative; the grid resolution is $r'$ higher than the reservoir topological dimension. 54
Figure 4.4 When using a resampled regular grid, we must use a proper resolution; otherwise, the reservoir model is not adequately represented. Still, geological faults might be misrepresented even when using higher resolutions. 55
Figure 4.5 The performance may abruptly decrease when using resampled regular grids with higher resolution, generating more texture memory cache hit misses. However, we still must keep a reasonable $r'$ value to generate a good approximation of the reservoir. 56
Figure 4.6 When employing a topological search, the parametric distance is evaluated in each direction, proceeding to the lower distance direction. We only pass through active elements, so we must take the element adjacency list. 57
Figure 4.7 We first build a resampled grid based on the AABB enclosing only active elements. We then estimate $\sigma_0$ by averaging the cell sizes. The resampled regular grid is used as input data to generate the volume of extinction coefficients. 58
Figure 4.8 Ambient occlusion computed in object space: (a) Precomputed values stored in a refined regular grid; (b) Precomputed values stored at model vertices. 59
Figure 4.9 Rotation positions for R2, R3 and R5 datasets at 0°, 90° and 180° for $X$, $Y$, and $Z$. 61
Figure 4.10 Performance and quality comparison for R2, R3, and R5 reservoir using a viewport of $1000^2$. The PLTS was used as ground truth for PSNR and SSIM computations. For all plots, higher values mean better results. 62
Figure 4.11 LAB color difference [56] considering the implemented strategies for R2, R3, and R5. The first two models present geometry discontinuities; the last is a large reservoir model. Note how SRG strategy suffered for delivering good approximations when the viable regular grid resolution is not sufficient to capture the scalar field variation. 63
Figure 4.12 Achieved results with only Emission and Absorption illumination model, and then combining directional ambient occlusion. Note how ambient occlusion helps reveal isosurfaces and regions placed in similar screen positions, but at different depths. 65

Figure 4.13 R3 and R5 datasets rendered with directional ambient occlusion using different extinction coefficient volume resolutions. We are still able to generate interesting illumination effects even using lower resolutions, which considerably decreases the memory requirements and pre-processing time, but might degrade the illumination quality. For a fair comparison, these images were rendered without using pre-illumination. 66

Figure A.1 Example of consecutive samples evaluated along a traced cone. 77

Figure B.1 Visual occlusion effect for Backpack dataset varying the standard deviation of the initial smoothing Gaussian kernel $\sigma_0$, using $\theta_o = 15°$. 82
Figure B.2 Visual shadow effect for SyntheticBars dataset varying the standard deviation of the initial smoothing Gaussian kernel $\sigma_0$, using $\theta_s = 0.5°$. 82
Figure B.3 Bonsai dataset varying the current initial step $h_0$, using $\sigma_0 = 1.0$. 83
Figure B.4 Comparison betwen Monte Carlo integration and our method for VisMale dataset using different cone aperture angles $\theta_o$. 84
Figure B.5 Visual shadow effects for Bonsai dataset using different cone aperture angles $\theta_s$. 84
Figure B.6 Comparison between our method and Monte Carlo for Backpack and VisMale datasets varying the maximum number of cones per section $C_{split}$. Backpack dataset was rendered using $\theta_o = 15°$. For VisMale, we set $\theta_o = 30°$. 85
Figure B.7 Comparison between our method and Monte Carlo for the VisMale dataset with a large region of semitransparent voxels, varying the maximum number of cones per section $C_{split}$. 85

Figure C.1 Backpack dataset comparison with Monte Carlo. 86
Figure C.2 Bonsai dataset comparison with Monte Carlo. 87
Figure C.3 CT-Knee dataset comparison with Monte Carlo. 88
Figure C.4 Foot dataset comparison with Monte Carlo. 88
Figure C.5 Hazelnut dataset comparison with Monte Carlo. 89
Figure C.6 Flower dataset comparison with Monte Carlo. 89

# List of tables

Table 3.1 Datasets used in our experiments. 38
Table 3.2 Performance comparison between ours and Schott et al.'s methods, using a viewport of $768^2$. The model VisMale(T) refers to the VisMale with semi-transparency region. The second column indicates the cone length, defined by the volume diagonal $D$, according to Section 3.4. 42
Table 3.3 Performance for occlusion comparison using two viewports with sizes $768^2$ and $1000^2$, considering our illumination method being evaluated in image and object spaces. 42
Table 3.4 Performance comparison for shadow generation using a viewport of $768^2$. The second column indicates the cone length based on the volume diagonal $D$, according to Section 3.4. 44
Table 3.5 Preprocessing times to generate the Extinction Coefficient Volume. 44
Table 3.6 Performance results with directional occlusion and shadows using a viewport of $768^2$; when pre-illumination is employed (object space light evaluation), the corresponding volume resolution is annotated. 45

Table 4.1 Reservoir models used in our experiments and base memory requirements from Franceschin et al. [21], using $r = 1.75$. 59
Table 4.2 Additional memory requirements to resample each reservoir, using different resolutions $r'$. 60
Table 4.3 Table with the used parameters to visualize the reservoirs. We also consider the memory required to store the additional light volume if pre-illumination is enabled (using half-precision floats). 64
Table 4.4 Performance results with the defined parameters in Table 4.3 using a viewport of $1000^2$, first computing Emission and Absorption only and then adding directional ambient occlusion. We can see how pre-illumination greatly increases performance for directional ambient occlusion computation. 64
Table 4.5 Pre-processing times to generate the Extinction Coefficient Volume for different reservoir models, considering $2 + \frac{1}{7}$ of memory consumption based on the volume resolution, using half-precision floats. 64

Table C.1 Performance results with directional occlusion compared with Monte Carlo using viewport of $768^2$; when pre-illumination is employed, the corresponding volume resolution is annotated. 87

# 1
# Introduction

Volume rendering is a valuable tool to visualize three-dimensional scalar data, from medical images to numerical simulation results. The technique is based on taking samples from a 3D scalar field to generate a representation of the volumetric data, without generating intermediate geometry. With the increasing power of hardware graphics, it became possible to visualize and explore volumetric datasets at interactive frame rates.

A volumetric data set is usually defined by a set of samples representing the values of some property at 3D locations $(x, y, z)$. When represented by a regular grid, samples are defined at evenly spaced intervals along the three orthogonal axes, which defines a structured dataset. Volumetric data can also be irregular, formed by cells of arbitrary shapes such as tetrahedra, hexahedra, or prisms [28]. For intermediate positions, the property value can be approximated via interpolation.

Accoring to [39], volume rendering algorithms can be grouped into four categories: ray casting, resampling or shear-warp, texture slicing (also defined as slice-based), and splatting. Isosurface extraction methods are also an alternative to produce hard surfaces and render them as a polygonal mesh. As discussed by Hadwiger et al. [17], ray casting is an attractive volume rendering technique because the rays are handled independently from each other, allowing several optimization strategies, such as natural parallel computing, early ray termination, adaptive sampling, and space leaping.

Considering each ray from a camera, the final color of the corresponding image pixel (assuming one ray per pixel) is defined by solving the volume rendering integral, which computes the amount of light emmited and absorbed by consecutive samples. It was presented by Max [37] and updated by Max and Chen [38]:

$$I(x, \hat{\omega}) = \int_0^D T(s)\tau(x(s))c(x(s))ds + T(D)B \tag{1-1}$$

where $D$ corresponds to ray's length, $c(x(s))$ is the emitted light, $\tau(x(s))$ is the extinction coefficient and $B$ is the background color. $T(s)$ corresponds to

the transparency, given by:

$$T(s) = e^{-\int_0^s \tau(x(u))du} \tag{1-2}$$

Both emitted light $c(x(s))$ and extinction coefficient $\tau(x(s))$ are given by a transfer function, which maps each scalar value to correspondent color and opacity values. It is possible to produce different visualizations by using appropriate transfer functions (see Figure 1.1).

Figure 1.1: Volume rendering of VisMale dataset using different associated transfer functions.

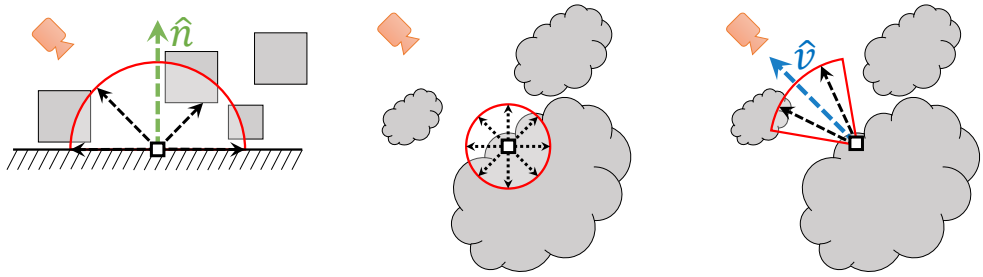


## 1.1 Interactive Directional Ambient Occlusion and Shadows

The standard emission and absorption illumination model, given by solving the volume rendering integral (Equation 1-1) [37], is unable to deliver a good depth and shape perception. A set of different lighting models have been proposed to generate better visualization of volumetric fields. The most common approach is the traditional gradient-based shading [33], which adds lighting effects using the classical Blinn-Phong illumination model [7], with the normals given by the gradients of the scalar field. Gradient-based shading is often applied because of its simplicity and negligible computation cost, but it only captures local lighting effects, making it more attractive when combined with other illumination models.

Ambient occlusion is a popular technique to simulate global illumination effects. It captures the amount of ambient light on each point by evaluating the nontransparent structures in its neighborhood, estimating the environment lighting. The computation of ambient occlusion appears in three different flavors in the literature. For surface models, ambient occlusion is usually computed by integrating the visibility function over the hemisphere positioned at the point of interest, aligned to the surface normal, as illustrated in Fig-

ure 1.2(a) [20, 13, 63, 50]. This approach is generally avoided for volume rendering. First, the gradient-based normal is not always well defined; second, one would have to consider different hemisphere orientations along a ray, introducing bias in the integration. For volume rendering, there are two commonly used strategies. The first strategy considers all the direction around a point to approximate the amount of light reaching it, as shown in Figure 1.2(b). It is usually performed by just evaluating the local occlusion around the point [24, 51, 15, 54], which is a fast but inaccurate approximation. A better strategy is the *directional ambient occlusion* computation, as illustrated in Figure 1.2(c). It integrates the transparency over a cone always pointed towards the viewer [55]. Our work focuses on this last strategy.




(a): Surface AO (b): Local AO (c): Directional AO

Figure 1.2: Different approaches to computing ambient occlusion (AO). The first is applied to surface models; the last two suits better to volume rendering.

The goal of ambient occlusion for volume rendering is to extend the basic absorption and emission illumination model, introducing scattering effects [37, 38]. It is too costly to consider multiple scattering effects, making it hard to achieve interactive frame rate. However, considering only single scattering effects crucially simplifies the illumination model and still significantly enriches the visualization. Nevertheless, the challenge of achieving interactive frame rate remains. Accurately computing the ambient occlusion would require evaluating a large set of secondary rays covering the cone domain. As several previous works [55, 54, 13, 20, 58], we have opted for replacing the tracing of a large number of rays by the tracing of a cone, achieving interactive frame rates. Different from others, the proposed cone tracing procedure evaluates the incoming light at samples along the ray using a mathematically plausible model based on Gaussian integrals. We are then able to efficiently obtain an approximation of the transparency integral within the cone.

The generation of shadows is another illumination effect that enriches the tridimensional perception of models. The goal is to compute the amount of direct light reaching a given point. The same cone tracing approach is used. In

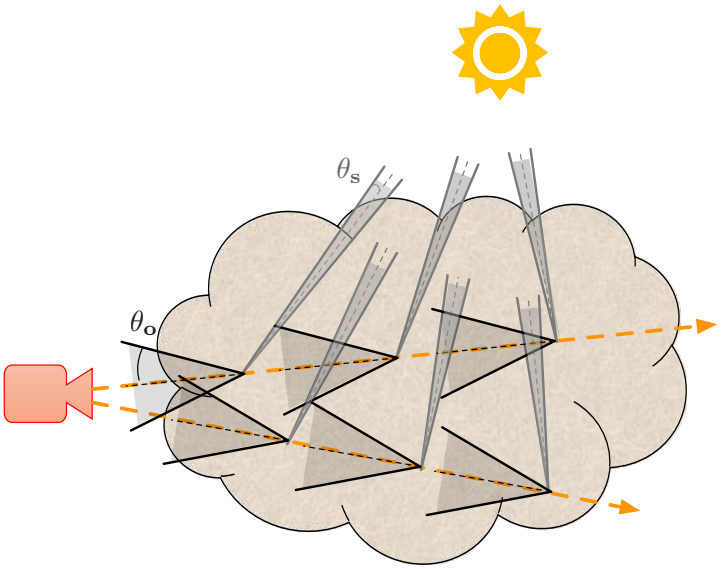



Figure 1.3: Cone placements for both directional ambient occlusion and shadow computations. The lighting effects are computed at each sample along the viewing rays. For directional ambient occlusion, we use cones with greater aperture angle, $\theta_o$, to capture the environment. For shadow computation, we use cones with smaller aperture, $\theta_s$, to generate hard shadows.

this case, we use cones with small apertures pointed towards the light sources, evaluating the amount of direct light reaching each voxel along the viewing ray. Figure 1.3 illustrates the placement of cones for both directional ambient occlusion and shadow computations at samples along viewing rays.

We ran a set of computational experiments to demonstrate the effectiveness of the proposed approach. We compare our proposal, regarding achieved image quality and performance, with previous interactive solutions, for both directional ambient occlusion and shadow generation. We also contrast the achieved quality results against a ground truth solution that uses Monte Carlo integration.

## 1.2 Experimental Study on Black Oil Reservoir Volume Visualization

Since the advent of GPU programming, much research has been conducted on efficient and accurate volume rendering techniques applied to unstructured meshes, considering both tetrahedral and hexahedral ones [60, 4, 44, 8, 41, 46]. The challenge resides on how to efficiently evaluate the underlying scalar field along the traced viewing rays. The task becomes easier for tetrahedral meshes because both geometry and scalar field vary linearly inside each cell (tetrahedron). One step is enough to evaluate the volume integration within a cell.

Things get more complicated for unstructured hexahedral meshes; both geometry and scalar field vary trilinearly within a cell. Regarding the geometry, even point location procedures require iterative numerical algorithms when dealing with cells delimited by non-planar faces. The scalar field sampled in the cell vertices presents a trilinear variation inside the cell; thus, isosurface patches present non-planar geometry. One single step may not be accurate to evaluate the volume integration within a cell. It is even hard to measure the loss in accuracy and the gain in performance when the problem is somehow linearized. How accurate is it to subdivide each hexahedron cell and render the volume as composed by tetrahedra? How accurate is it to approximate each non-planar hexahedron face by two triangles? How accurate is it to consider a linear variation of the scalar field along the ray within a cell?

These questions justify a careful investigation, which is the second goal of this thesis. We conduct an experimental study comparing different solutions to compute the volume integral for hexahedral meshes. We focus on visualizing the result of black oil reservoir simulations. Reservoir simulation is widely used in the oil industry to plan and predict field exploration. Traditional reservoir viewer solutions rely on map, section, and boundary views to reveal the volume of data. However, with the increasing computational power and the use of massive models (e.g., for modeling the giant Brazilian pre-salt fields), a general volume visualization solution is necessary. Figure 1.4 illustrates the effectiveness of volume rendering for reservoir model inspection; the cone-shaped flow of water is clearly revealed. The identification of such fluid flow pattern is meaningful in the study of reservoir dynamics [53].

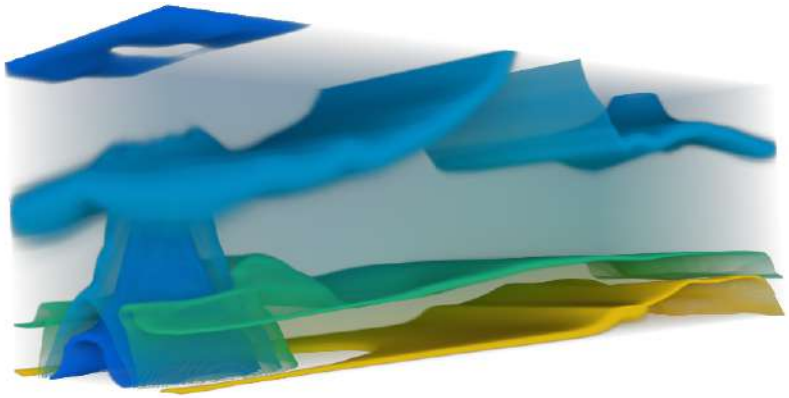


Figure 1.4: A cone-shaped flow of water revealed by the volume visualization. The image was rendered by adding directional ambient occlusion effects.

An essential aspect of a visualization system is the ability to provide useful data insights, helping users to better analyze different regions in the dataset. Also, illumination techniques can be helpful in data interpretation by

enhancing depth and shape comprehension. However, as a pre-requisite, interactive performance must be ensured to prevent harming the user experience.

This experimental study is focused on black oil reservoir models, which are hexahedral meshes laying on topological grids. This structured topological aspect of reservoir models is explored for both reducing the amount of memory needed to store the model in the GPU [21] and also to accelerate retrieving scalar field values along the rays. We experiment different approaches for performing the ray casting algorithm, comparing achieved image quality, memory consumption, and performance. We also adapt our directional illumination model [9] to enhance volume inspection.

## 1.3 Contributions and Overview

The main contributions of this thesis are:

- A new method to compute directional shading for structured datasets, adding directional ambient occlusion and shadows.
- An experimental study to generate volumetric visualizations of black oil reservoirs, also adapting our proposed directional illumination technique.

The first contribution of this thesis, showing our illumination method for structured datasets, was published in Volume 84 (November 2019) of Computers & Graphics Journal [9]. The second contribution, showing our experimental study with volumetric visualizations of black oil reservoirs, was published in Volume 93 (December 2020) of Computers & Graphics Journal [10]. The remainder of this thesis is organized as follows. In Chapter 2, we review some techniques related to volumetric illumination and non-structured volume rendering. In Chapter 3, we present our first contribution, adding directional ambient occlusion and shadows for structured datasets. In Chapter 4, we present our experimental study with volume rendering of black oil reservoirs, adapting our directional illumination model. In Chapter 5, we present conclusions and ideas for future work.

# 2
# Related Work

In this chapter, we review past works related to volumetric illumination and unstructured volume rendering.

## 2.1
## Volumetric Illumination

Jönsson et al. [26] discussed several illumination techniques for volume rendering, with only few exploring the integration with non-structured datasets. The Gradient-based shading [33] is the simplest approach to add lighting effects in volume rendering applications. It computes local effects by applying the Blinn-Phong illumination model, composed by ambient, diffuse and specular terms, using the gradient of the scalar field as the normal vector. Since gradient-based shading is cheap, it is commonly mixed with other methods to enhance the results. Correa et al. [12] also presented a study of gradient estimation techniques for unstructured volume rendering concerning their cost and performance to compute local illumination.

Schott et al. [55] proposed a directional occlusion shading model exploiting the slice-based composition. The amount of remaining light is stored in a second buffer that is updated after each integration step of the color buffer. Some related works [34, 16] have shown that directional occlusion shading achieves better results on improving perceptual capabilities of volumetric models when compared to other illumination methods. Šoltészová et al. [61] made an extension to directional occlusion shading enabling the light position to be around the viewer's hemisphere. Later, Magnus and Bruckner [35] also added refraction and caustics effects using a similar slice-based approach.

Some methods rely on a precomputed auxiliary structure, storing intermediate light information used to evaluate each sample. Ropinski et al. [51] constructed local histograms to estimate the amount of occlusion at each voxel. Hernell et al. [24] made a local approximation, casting multiple rays around the spherical neighborhood, to compute the current visibility at each voxel.

Some works employed Summed-Area Tables (SAT) to provide light computation for interactive visualizations. Being introduced as a 2D structure by Crow [14], it is capable of efficiently evaluating the sum of an arbitrary

rectangular region using a small number of fetches. Each entry of a SAT holds the sum of values between the sample location and the bottom left sample in the corresponding texture. SATs can be extended to evaluate sums of cuboid regions in 3D scalar fields. While the sum of any 2D rectangular region can be evaluated using only four texture samples, any 3D cuboid region can be evaluated using eight samples. SATs can suffer from precision issues, since it may need to store very large numbers in textures. Some modifications can be done to decrease the precision errors, like subtracting the mean value on each sample and adding it back after each evaluation, but dealing with values that represent sums of data is inevitable. A major disadvantage of using SATs is the inability to compute sums not aligned to the main axes.

Díaz et al. [15] first used 2D and 3D SATs to evaluate occlusion. One 2D SAT was used to add halos based on the depth information stored in screen space. A 3D SAT was used to evaluate opacity in the neighborhood of a voxel, as the accumulated opacity reaches a specific value. Later, Ament et al. [3] used SATs to approximate the extinction of a neighborhood to compute multiple scattering and to evaluate directional and point light sources for shadow generation [2].

Schlegel et al. [54] used SAT to evaluate local ambient occlusion and color bleeding, using cuboid shells, also soft directional shadows, and scattering, based on projected light cones. Their method produces interesting results, and, since it is not dependent on the viewer's position, all illumination data can be stored into one additional 3D texture, fetched in the fragment shader. The main limitation of their method is the need to evaluate the sections of each light cone aligned with the 3D axes of the volume.

Crassin et al. [13] developed a voxel cone tracing technique to compute occlusion and shadows for polygonal meshes. They step along the cone axis and perform lookups in a mipmapped pyramid of pre-integrated values, with the pyramid level corresponding to the cone radius. Favera and Celes [20] also applied a cone tracing approach to approximate ambient occlusion, sampling a sequence o tangent spheres. Boths works uses front-to-back composition to evaluate the remaining light from each cone by sequencial samples along the cone axis. Recently, Kraft et al. [30] presented an adaptive sampling approach to compute ambient occlusion for volume rendering based on voxel cone tracing. The method is cheap, since they subdivide the primary ray into a limited number of segments, and ambient occlusion is evaluated once per segment.

Using a similar approach from [13], Shih et al. [58] proposed a parallel GPU-distributed volume rendering for large datasets and also used voxel cone

tracing to evaluate indirect illumination, focusing on shadow generation. They opted for computing a clipmap texture, where higher levels double the covered area, named as a super voxel. Each super voxel stores the mean and standard deviation of data values in the covered region. With these values, they apply a Gaussian filter on the discretized opacity transfer function.

We assume Gaussian distribution differently. For each voxel, we assume a local Gaussian distribution of extinction coefficient values; then, we use Gaussian integrals to approximate the integral evaluations in the cone volume. As we shall demonstrate, the proposed approach is mathematically plausible, fast, and does not suffer from axis alignment.

Some techniques proposed the approximation of global illumination for volume rendering. Zhang and Ma [70] approximated this effect by numerically solving a convection-diffusion equation, later being extended by Shih et al. [57] for unstructured data. More recent works used caching-based approaches to provide interactive visualizations by storing and reusing light information for photon mapping [27], irradiance caching [29] and path tracing [36], using progressive updates when necessary. Ament and Dachsbacher [1] also proposed a way to compute anisotropic shading of surface-like structures, but requiring better investigation on its perceptual benefits. Recent works also investigate the use of denoising [25] for volumetric path tracing and 3D convolutional neural networks (CNNs) [18] to approximate ambient occlusion.



## 2.2 Unstructured Volume Rendering

According to Muigg et al. [45], the four main approaches to render unstructured grids are based on cell projection (via the projected tetrahedra algorithm) [59], ray-casting [66], resampling into a structured grid [68], and point-based approaches [71]. We focus our experimental study on both resampling and ray-casting. Both cell projection and point-based approaches require a visibility sorting computation, which might become a bottleneck for these methods [45]. Resampling might simplify the solution to a structured volume rendering, which can be advantageous since it simplifies the rendering step. However, the use of high resolutions tends to increase the memory required to store the dataset abruptly.

Resampling can be done using hierarchical or flat blocking representations. Leven et al. [32] proposed to resample the unstructured grid into an adaptive 3D texture octree. Beyer et al. [5] also proposed a mixed-resolution volume rendering with flat multi-resolution bricks, instead of a hierarchical approach.

Garrity [23] proposed the first technique for unstructured volume rendering based on ray-casting, traversing the whole mesh following the connectivity of neighboring elements (cells) and using a uniform grid to find each ray entry point. Weiler et al. [66] later proposed a GPU solution based on Garrity's work.

Bernardon et al. [4] proposed a new algorithm using depth-peeling, rendering only the boundary faces of the mesh multiple times to define each entry point. Weiler et al. [67] also proposed a similar solution using depth-peeling, handling non-convex meshes. Espinha and Celes [19] explored partial pre-integration to render tetrahedral unstructured meshes.

For rendering hexahedral meshes, one approach is to subdivide each cell into five or more tetrahedra, with the drawback of requiring additional memory to store the datasets. Carr et al. [11] considered different subdivision schemes, evaluating artifacts generated using isosurface rendering. Miranda and Celes [41] opted for rendering the hexahedral mesh directly, computing the volume integral in each cell using the Gauss-Legendre quadrature method. It uses less memory than tetrahedron subdivision schemes since it stores fewer elements per mesh.

Muigg et al. [45] used structured bricks to simplify the original unstructured grid representation based on the degree of interest measures, with structured bricks being resampled in less important regions. Muigg et al. [46] also proposed a face-based data structure called two-sided face sequence lists (TSFSL) to store polyhedral grids, and a GPU ray-casting approach was employed to render the proposed representation, using different strategies to evaluate each type of cell.

For topological grids, as reservoir models, more specialized data structures can take advantage of implicit neighboring rules. Recently, Franceschin et al. [21] proposed a specialized compact data structure to store reservoir models in the GPU memory. Combined with a regular grid as the acceleration technique, their solution provides an efficient point location algorithm for handling hexahedral elements with non-planar faces.

# 3
# Directional ambient occlusion and shadows for volume ray casting

This chapter presents our first contribution, adding directional ambient occlusion and shadows on structured datasets for volume ray casting. We first show the directional illumination model and our proposal to compute transparency using Gaussian integrals. Then, we describe our shading model and discuss the choice of parameters that control our method. We also made a comparative analysis of the achieved results using past proposed techniques.

## 3.1
## Directional Illumination Model



Directional occlusion shading for volume rendering requests the computation of the amount of light that reaches a given point through a representative cone. Considering the single-scattering model [37][38], the amount of radiance $I(x, \hat{\omega})$ integrated along a viewing ray, considering the inscattered light at each sample, is given by:

$$I(x, \hat{\omega}) = \int_0^D T(s)\tau(x(s))g(x(s), \hat{\omega})ds + T(D)B \tag{3-1}$$

where $D$ is the distance travelled by the ray until it reaches an opaque background, $\tau(x(s))$ is the extinction coefficient, $B$ is the background color, $\hat{\omega}$ is the direction of the viewing ray, and $x(s)$ is the current position between the eye and the background. The transparency $T(s)$ is given by:

$$T(s) = e^{-\int_0^s \tau(x(u))du} \tag{3-2}$$

Also, the scattering term $g(x, \hat{\omega})$ considers incoming light covering the whole sphere $\Omega$:

$$g(x, \hat{\omega}) = \int_\Omega r(\hat{\omega}, \hat{\omega}')I(x, \hat{\omega}')d\hat{\omega}' \tag{3-3}$$

where $\hat{\omega}'$ represents all incoming directions, $r(\hat{\omega}, \hat{\omega}')$ is the phase function, a 1D function of the angle $\theta$ between two directions $\hat{\omega}$ and $\hat{\omega}'$, defining the angular distribution of scattered radiance, also considered as a normalization factor. The isotropic phase function is a trivial case considering equal scattering in all directions, defined by $r(\hat{\omega}, \hat{\omega}') = \frac{1}{4\pi}$. Equation 3-3 is difficult to solve for interactive visualizations if we consider multiple scattering scenarios, since it requests recursive evaluations of $I(x, \hat{\omega})$. However, considering the single-scattering model, which assumes that the probability of light scattering more than once is very low [38], only the attenuation of a light source (or the background) with color $L_0$, at a distance $L$ needs to be computed, simplifying the solution of $I(x, \hat{\omega}')$ to:

$$I(x, \hat{\omega}') = L_0 e^{-\int_0^L \tau(x - s\hat{\omega}')ds} \tag{3-4}$$

Instead of covering the whole spherical domain $\Omega$, directional shading uses a cone-shaped phase function. Then, $g(x, \hat{\omega})$ accounts for the radiance in a cone domain $\Psi$:

$$g(x, \hat{\omega}) = \int_{\Psi} r(\hat{\omega}, \hat{\omega}') I(x, \hat{\omega}') d\hat{\omega}' \tag{3-5}$$

Equation 3-5 provides a directional illumination effect to volume rendering visualization.

### 3.1.1 Ray Tracing

An accurate solution for Equation 3-5 uses Monte Carlo integration based on casting a large set of rays starting from the cone apex in all directions of the cone domain. Using the trapezoid rule as the numerical integration method, we have:

$$g(x, \hat{\omega}) \approx \frac{\sum_{i=1}^{n} (\hat{\omega} \cdot \hat{\omega}_i) V_\tau(x, \hat{\omega}_i)}{\sum_{i=1}^{n} (\hat{\omega} \cdot \hat{\omega}_i)} \tag{3-6}$$

with $V_\tau$ being defined as:

$$V_\tau(x, \hat{\omega}') = \prod_{j=1}^{m-1} e^{\left(-\frac{\tau(x_j) + \tau(x_{j+1})}{2} h\right)} \tag{3-7}$$

having $x_j = x + (h_0 + jh)\hat{\omega}'$, where $h$ is the integration step and $h_0$ is an initial gap used to avoid self-occlusion. Since we must evaluate a high number of rays for each sample to generate an accurate result, it is costly to apply Equation 3-6 for interactive visualization.

## 3.2 Proposed Transparency Computation

Instead of computing Equation 3-5 by a large set of rays, we propose to approximate transparency over the whole cone $\Psi$ with a single light ray:

$$g(x, \hat{\omega}) \approx L_0 T_\Psi(L) \tag{3-8}$$

with $T_\Psi$ being computed by evaluating the transparency in the cone domain:

$$T_\Psi(L) = e^{-\iiint_\Psi \tau(u,v,w) du dv dw} \tag{3-9}$$



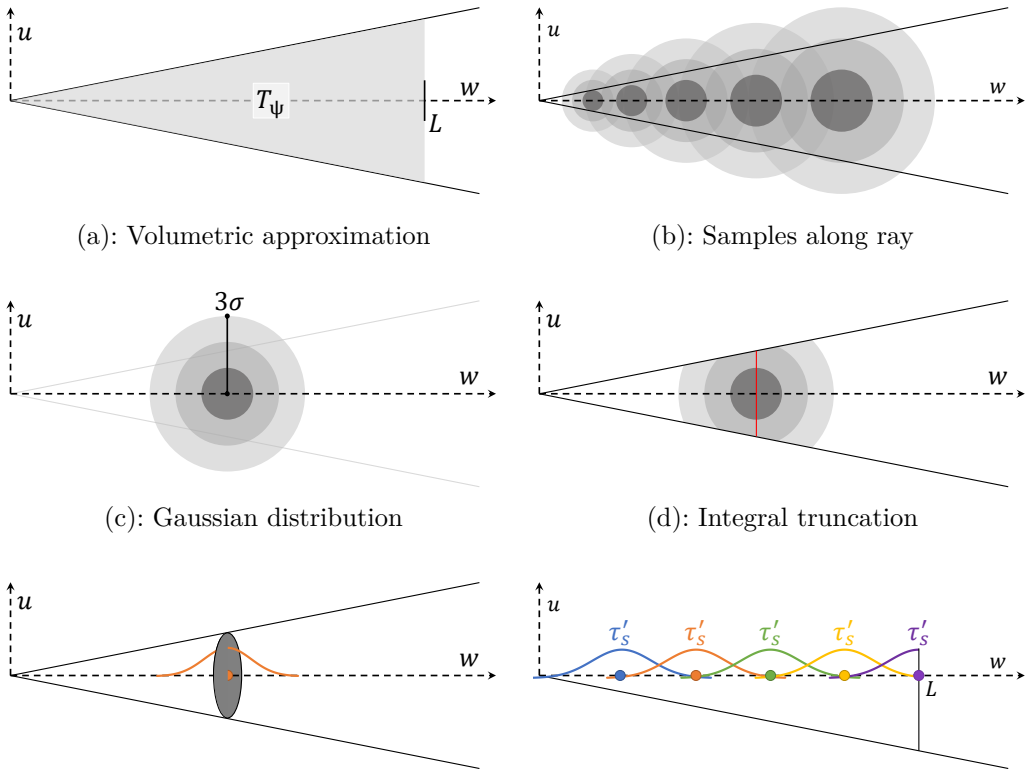
where $(u, v, w)$ is the coordinate system of samples inside $\Psi$, with $w$ in the direction of the cone axis.

The cone transparency is evaluated by discrete samples positioned along the ray axis. We then assume that the variation of the extinction coefficients around each sample follows a Gaussian distribution. The volume transparency in the vicinity of each sample is then computed by a Gaussian integral. In the $uv$ plane, perpendicular to the cone axis, the integrals are limited by the cone surface; in the $w$ direction, the integrals are combined in a continuous way to compute the overall transparency inside the cone. These steps are illustrated by Figure 3.1.

In the following subsections, we present in detail the mathematical model that supports this strategy, describing: how to consider a representative Gaussian distribution of extinction coefficients, how to position samples along the ray, how to efficiently evaluate transparency integrals, and how to truncate and combine integrals.

### 3.2.1 Gaussian distribution of extinction coefficients

Given a transfer function, we compute the associated volume of extinction coefficients. We want to assume a Gaussian distribution of values around each voxel in isolation, with an imposed standard deviation ($\sigma$) and the amplitude given by the voxel value. As each voxel in isolation must represent




(a): Volumetric approximation

(b): Samples along ray

(c): Gaussian distribution

(d): Integral truncation

(e): 1D representative value along $w$

(f): Consecutive 1D samples

Figure 3.1: Our proposed approach for transparency computation. We first consider a volumetric integral instead of a bunch of rays (a), which is computed by adding consecutive samples (b). Each sample represents a Gaussian distribution (c), which is truncated by the cone domain (d) and evaluated along the $w$ direction (e). Finally, consecutive samples are placed along $w$ to approximate the whole cone transparency $T_\Psi(L)$ via numerical integration (f).

its vicinity region, the associated value needs to be a representative value of its neighborhood. Based on computational experiments, we have opted to compute the local average of surrounding values using Gaussian weights, considering the chosen $\sigma$, what is equivalent to applying a Gaussian filter in the volume. We do that by convolving the volume with a Gaussian kernel considering the standard deviation of $\sigma_0 = 1.0$, creating a new volume of values. We then perform successive convolutions using Gaussian kernel to build the mipmap pyramid levels. Each level doubles the $\sigma$ value of the associated distribution. We end up with a mipmap texture of representative extinction coefficients.

The final average value stored at each voxel, $\tau_s$, is considered as the amplitude of the associated Gaussian:

$$f(u,v,w) = \tau_s e^{-\frac{u^2+v^2+w^2}{2\sigma^2}} \tag{3-10}$$

The texture is fetched at each sample to estimate the Gaussian integral around the point. The level of the mipmap pyramid is chosen by the circular section radius of the cone, as we shall discuss in the next subsections.

It is worth mentioning that, as extinction coefficients span to the infinity, we do all these computations on opacity values. In the end, we convert opacities back to extinction coefficients. Extinction coefficients $\tau \in [0, \infty]$ can be converted to opacity values $\alpha \in [0, 1]$ according to the expression: $\alpha = 1 - e^{-\tau l}$. Here, $l$ represents the length traveled by the light in the volume. It is then easy to convert the extinction coefficient to opacity and vice-versa [69].

### 3.2.2 Cone sampling

The cone is sampled along its axis; at each sample, the extinction coefficient variation follows a Gaussian distribution. The radius of the cone circular section at a sample is expressed by:

$$r = d \tan(\theta) \tag{3-11}$$

where $d$ represents the distance to the cone apex and $\theta$, the cone aperture, as illustrated in Figure 3.2.

Our first goal is to reduce the volumetric Gaussian distribution to a representative unidimensional one, along the $w$ direction. We compute the Gaussian integral on the circular cone section at the sample, perpendicular to the cone axis (the $uv$ plane), truncating it by the cone surface.

Recall that the integral of a 1D Gaussian function spanning to the infinity is given by:

$$\int_{-\infty}^{\infty} \tau e^{-\frac{t^2}{2\sigma^2}} dt = \tau \sigma \sqrt{2\pi} \tag{3-12}$$

with $\tau$ being the extinction coefficient of the medium. The integral can be easily truncated for limited intervals. For symmetric interval, the truncation can be expressed by a computed percentage of the whole domain, $p$:

$$\int_{-s}^{s} \tau e^{-\frac{t^2}{2\sigma^2}} dt = \tau p \sigma \sqrt{2\pi} \tag{3-13}$$

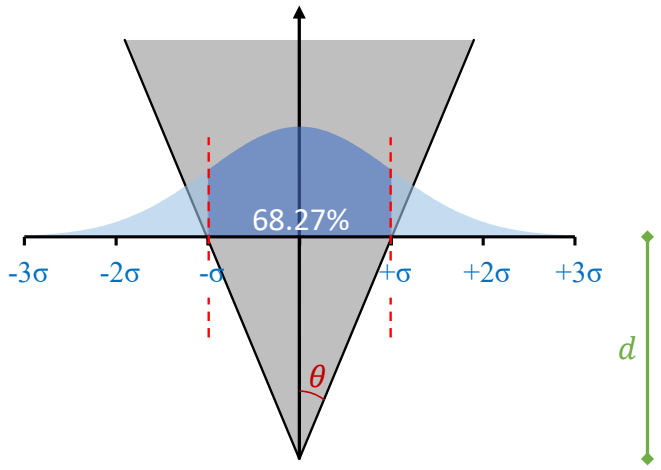



Figure 3.2: At each sample, the transparency on the $uv$ plane perpendicular to the cone axis is captured by a Gaussian integral. This integral is truncated by the cone surface: we limit the integral interval from $-r$ to $r$ in both $u$ and $v$ directions, where $r$ represents the circular section radius. As an example, limiting a 1D Gaussian integral from $-\sigma$ to $\sigma$ results in 68.27% of the full value.

We use these properties to first compute the integral on the $uv$ plane and truncate it to span from $-r$ to $r$, as illustrated in Figure 3.2. The truncated integral value is then divided by the circular section area, ending up with a 1D Gaussian distribution along the cone axis:

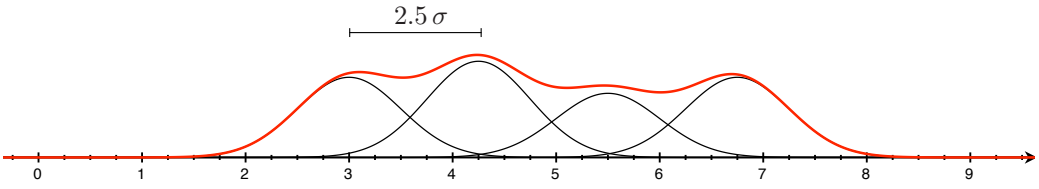
$$\iiint_{\Psi} \tau_s \, dudvdw \approx \int_0^L \frac{\left(p_r \sigma \sqrt{2\pi}\right)^2}{A_c} \tau_s \, dw$$

where $p_r$ is the percentage due to limiting the integral from $-r$ to $r$ in both $u$ and $v$ directions, $A_c = \pi r^2$ is the circular section area, $L$ is the ray length and $\tau_s$ is the averaged extinction coefficient value of the associated distribution.

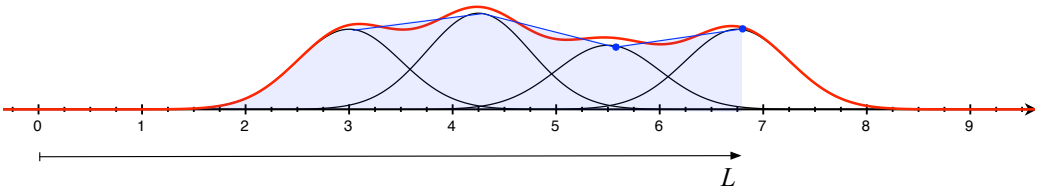
Now, to approximate the radiance in the cone domain $\Psi$, we evaluate the total amount of non-occluded ambient light (transparency), which can then be expressed by:

$$T_{\Psi} \approx e^{-\int_0^L \frac{\left(p_r \sigma \sqrt{2\pi}\right)^2}{A_c} \tau_s \, dw} \tag{3-14}$$

To compute this integral, we consider all the samples along the cone axis. At each sample, we have now a 1D Gaussian distribution with amplitude given by:




(a): Summing consecutive Gaussian functions distance apart $2.5\sigma$




(b): Aproximating the sum by a linear function

Figure 3.3: Placement of successive extinction coefficient distributions along the cone axis, from left to right. The integral from the first to the last sample can be approximated by the trapezoid rule, considering the Gaussian amplitudes.



$$\tau_s^{'} = \frac{\left(p_r \sigma \sqrt{2\pi}\right)^2}{A_c} \tau_s \tag{3-15}$$

As illustrated in Figure (a), we set sample distance as $h = 2.5\sigma$, in order to cover the whole axis domain. Note that summing the Gaussians results in a curve which integral can be approximated by the polygonal area painted in blue in Figure (b), from left to right:

$$\iiint_{\Psi} \tau(u,v,w) du dv dw \approx 0.5\, \sigma \sqrt{2\pi}\, \tau_{s(0)}^{'} + \sum_{0}^{n-1} 2.5\sigma \frac{\tau_{s(i)}^{'} + \tau_{s(i+1)}^{'}}{2} \tag{3-16}$$

where $n$ represents the number of samples to reach distance $L$ and $\tau_{s(i)}^{'}$ is the Gaussian amplitude at sample $i$. This equation approximates the integral by considering 50% of the first Gaussian integral plus the trapezoid rule considering the Gaussian amplitudes.

### 3.2.3 Use of the mipmap pyramid

In the pre-processing phase, we assume a Gaussian distribution with $\sigma = 1$ for the base level of the mipmap pyramid. For each upper level, the $\sigma$ value is doubled in relation to the previous level. The upper levels of the pyramid are needed to cover large cone sections. The Gaussian distribution

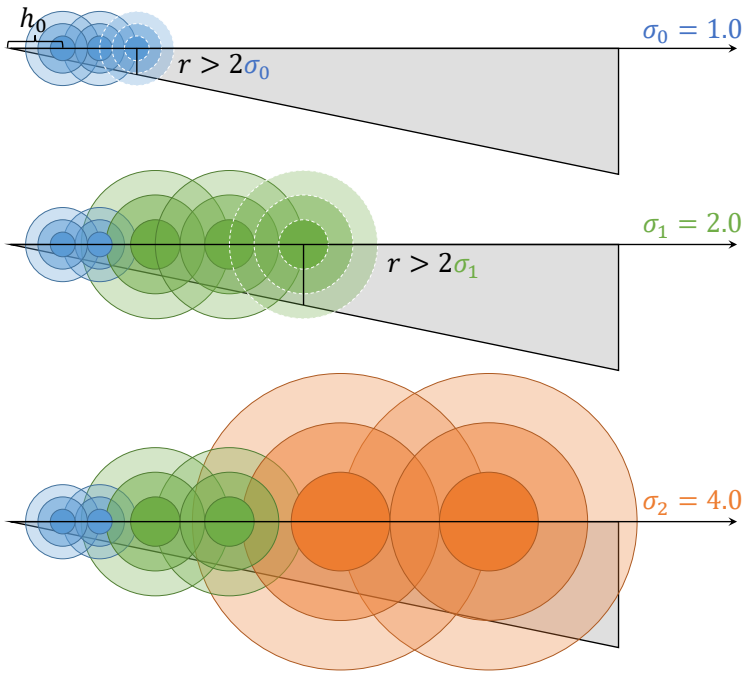



Figure 3.4: Example of cone sampling using $r_c = 2\sigma$. Each time the current section radius becomes larger than $2\sigma$, we increase the fetched pyramid level, doubling $\sigma$. The initial gap, $h_0$, is necessary to avoid self-occlusion.

with $\sigma = 1$ does not cover the entire circular section for large $r$ values. The proposed strategy always seeks to use a distribution that spans a region larger than the cone section at the considered sample, using the upper levels.

Starting with $\sigma = 1$, the initial samples are distanced $h = 2.5$ from each other. The level 0 of the mipmapping pyramid is fetched while the circular cone section has radius below a given limit, $r_c$. If the circular section radius exceeds $r_c$, we start fetching level 1, doubling $\sigma$, and consequently doubling $h$ and $r_c$. We keep increasing the fetched pyramid level until the desired cone length, $L$, is reached. An initial gap, $h_0$, is employed to avoid self-occlusion. Figure 3.4 illustrates our sampling process.

### 3.2.4 Cone splitting

As long as the cone circular section radius is small, the Gaussian integral provides a reasonable approximation of the extinction integral. However, as upper levels of the mipmap pyramid are accessed, the approximation degrades, mainly because a large Gaussian is not capable of capturing occluder's details. To reduce this loss in accuracy, we propose to split the cone as the cone section radius enlarges.

Our main goal is to preserve the texture access at level 0, as far as

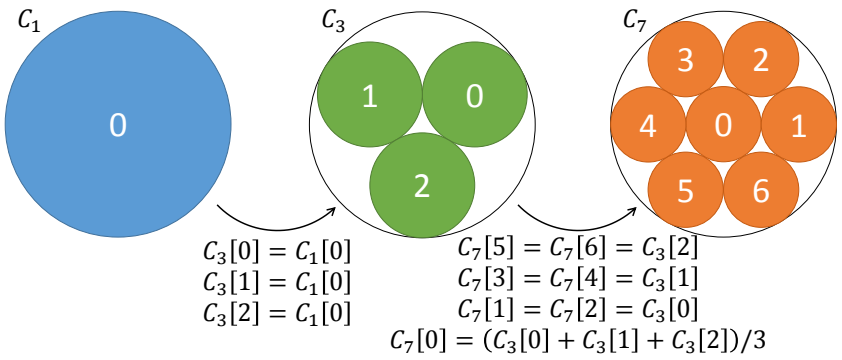



Figure 3.5: The "circle packing in a circle" arrangement applied in our technique. For each split step, we must correctly update the current transparency of the new set of cones.

possible. We start with one Gaussian for each cone section. When the section radius achieves its limit, $r_c$, we split the initial cone in three, with smaller apertures. When the section radius again achieves its limit, we split the three cones in seven. During splitting, the cone positioning and apertures follow the "circle packing in a circle" arrangement [22], as illustrated in Figure 3.5. For the one-to-three split, all new three cones inherit from the previous cone the computed transparency. For the three-to-seven split, the new cones inherit from the previous cones differently: the center cone receives as its transparency the average value from all three previous cones, and the border cones inherit the transparency from the corresponding closest previous cone, as depicted in Figure 3.5. With this strategy, we only access the base level of the mipmap pyramid, until the limit radius, $r_c$, is reached after the last splitting step. After each split, all cones are updated independently. In the end, the final transparency is given by the cosine-weighted average of all cones.

We have chosen to use the only additional arrangements of 3 and 7 circles because of simplicity, circle coverage, and axis-symmetry, trying to balance performance and quality. To employ more than seven sub-cones would penalize the performance without bringing significant quality improvements in practice. In Appendix A, we added some implementation details of our proposed transparency computation.

## 3.3 Directional Shading

Now that we have an efficient way to compute transparency in a cone region, we can apply illumination effects for ray casting.

### 3.3.1 Directional ambient occlusion

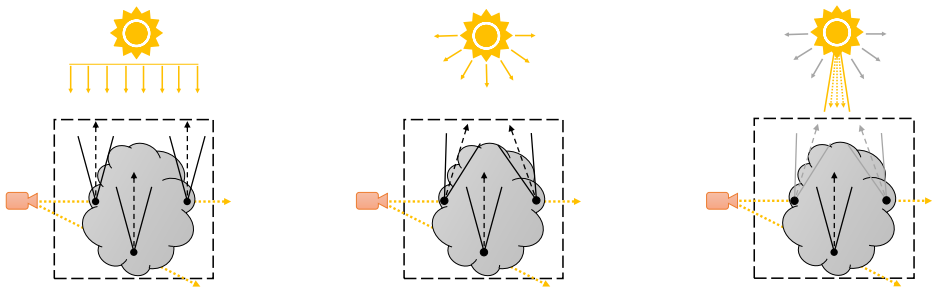
For directional ambient occlusion, the cones are pointed towards the viewer. Ray casting computes lighting by numerically evaluating the volume integral equation, along the primary rays. At each iteration, a cone has to be traced towards the viewer, and the ambient occlusion is computed for each integration step. The transparency computation is only performed for non-transparent samples along the ray. As a result, the use of transfer functions that result in transparent voxels tends to perform better, if compared to volumes with large semi-transparent regions.

### 3.3.2 Shadow generation

Inspired by Schlegel et al. [54], we employ our same transparency computation for shadow generation. Here, the cones are oriented to the light sources. Again, transparency computation is only performed for non-transparent samples.

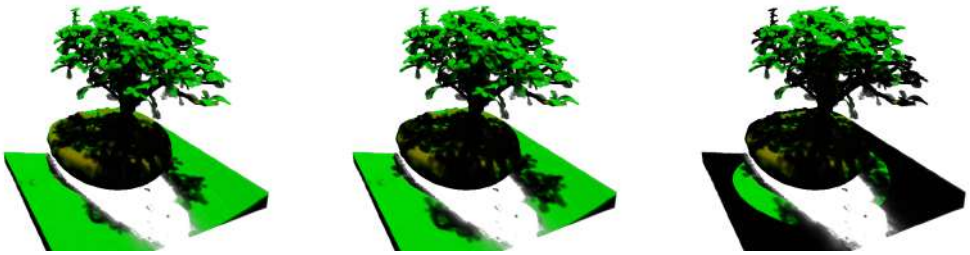
Both hard and soft shadows can be generated, varying the cone apertures. In general, we prefer to use very narrow cones to capture hard shadows. The use of narrow apertures requires more samples to compute transparency within a cone, since the radius of circular sections are small, and the samples are kept close to each other. On the other hand, the splitting strategy does not introduce significant gain in quality and can be discarded.

The cone orientation varies according to the type of light, directional, point, or spot, as illustrated in Figure 3.6 and exemplified by Figure 3.7.



(a): Directional (b): Point (c): Spot

Figure 3.6: Types of shadows implemented using our cone tracing. When defining the position of a light source, we also store its orthogonal axes, composed by a forward, up and, right vectors, orienting the cones appropriately.

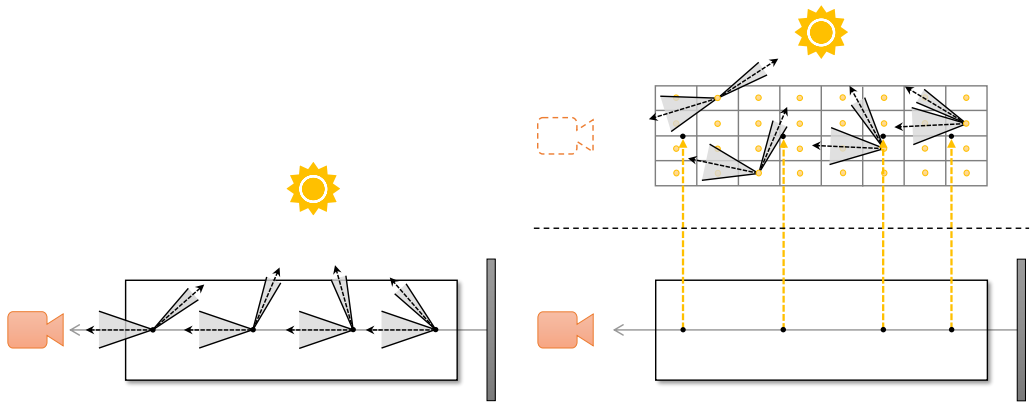


(a): Directional (b): Point (c): Spot

Figure 3.7: Types of shadows implemented using our cone tracing on Bonsai dataset.

### 3.3.3 Pre-computed transparency

Our illumination model can be easily evaluated in either image space [58, 55, 15] or object space [58, 54, 2], as illustrated by Figure 3.8. In image space, directional ambient occlusion and shadows are computed for each sample along the viewing ray during the rendering stage, as described. In object space, the transparency is pre-computed. Given the viewer position (for directional ambient occlusion) or the light source position (for shadows), one additional texture is created to store the transparency calculations for each voxel. It is possible to even employ a resolution smaller than the input dataset. In the rendering stage, this texture is fetched for each sample along the viewing ray.



(a): Image space (b): Object space

Figure 3.8: Illustration of both evaluation models implemented with our proposed technique. For object space, an additional texture is generated and the transparency is calculated for each voxel before the rendering stage, being interpolated at intermediate positions.

Note that this pre-computation has to be done at each frame for directional ambient occlusion, as the viewer moves. Even so, the pre-computation

significantly improve performance for large semi-transparent regions, when the screen resolution is larger than the volume resolution or when the integration step is smaller than one voxel unit. In these cases, the pre-computed transparency volume avoids evaluating the same cone multiple times. The loss in image quality is only noticeable if the texture resolution is too small. On the other hand, for volumes with several transparent or opaque voxels (this latter due to early ray termination), the pre-computation can be costlier than calculating the transparency when needed during rendering.

## 3.4 Parameter Control

The proposed method suggests a set of parameters that affects the achieved result. Our goal is to propose default values to minimize user intervention while preserving the applicability of the algorithm for different transfer functions and datasets. In Appendix B, additional results were generated changing the parameters.

First, there is a set of parameters that is intrinsic to the method:

- $\sigma_0$: initial standard deviation of the assumed Gaussian distribution of extinction coefficients; we have set $\sigma_0 = 1$. Setting larger numbers to $\sigma_0$ would make the associated Gaussian distributions representative of larger regions, decreasing accuracy when sampling small circular sections (the integral truncation would be severe). Smaller values would require the use of more samples to cover the cone axis domain, decreasing performance.

- $h_0$: initial gap to avoid self-occlusion. This parameter is probably the most difficult to set a good general default value. The appropriate value varies according to the scene and the boundaries of the scene revealed by the transfer function in use. Based on computational experiments, we have chosen to use $h_0 = 3\sigma_0$. Smaller values would increase self-occlusion, darkening the image; greater values would miss high-frequency occlusion near the sample.

- $h$: sample separation along cone axis; as explained, we have fixed $h = 2.5\sigma$, with $\sigma$ being the current Gaussian standard deviation, given by $\sigma = 2^{level}$. As illustrated in Figures 3.3, this value makes it plausible to approximate the integration of the sum of consecutive Gaussian distribution by a polygonal area.

- $r_c$: maximum circular cone section before splitting or increasing the pyramid level; we have fixed $r_c = 2\sigma$. Smaller values would make upper

levels of the pyramid being accessed earlier, decreasing quality; greater values would turn the Gaussian distribution less representative of the region inside the cones, also decreasing quality.

Second, there is a set of extrinsic parameters that allow the user to adjust the desired result, balancing performance and image quality.

- $\theta$: cone aperture, controlling the size of the vicinity region considered for occlusion. The proposed method works better for relatively small apertures ($\theta_o < 30°$). Large apertures would require each sample to represent a large region, decreasing accuracy. For shadow generation, as mentioned, we employ very small apertures ($\theta_s < 3°$), the smaller the aperture, the harder is the shadow.

- $L$: ray length; for ambient occlusion computation, we have successfully set $L = 0.50D_v$, with $D_v$ being the volume diagonal length; for shadows computation, we have set $L = 0.75D_v$. Smaller values may ignore important distant obstacles; greater values may introduce unnecessary computations.

- $C_{split}$: maximum number of cones per section; valid values are 1, 3, and 7. This parameter is critical for sampling large cone circular sections. If we set $C_{split} = 1$, we only use one Gaussian per cone section, increasing performance but decreasing image quality, since upper levels of the mipmap pyramid are used very early. Setting $C_{split} = 7$ provides the best result; however, for transfer functions that result in large semi-transparent media, setting $C_{split} = 3$ improves performance with low impact in image quality. For directional ambient occlusion computation, we have tried $C_{split}$ varying from 1 to 7; for shadow generation, we have set $C_{split} = 1$ or 3.

- $h_p$: integration step for the primary ray tracing. The primary step controls the numerical accuracy of the ray tracing. Smaller values bring numerical precision, while larger values improve performance. We have set $h_p = 0.5$ voxel unit in our tests.

Finally, there is an additional intrinsic parameter, a global attenuation factor $\alpha < 1$. The use of such factor is a common practice in illumination techniques [58, 54]. The goal is to adjust the overall lightness of the scene. It appears as an additional parameter in the Equation 3-15.

$$\tau_s^{'} = \alpha \frac{\left(p_r \sigma \sqrt{2\pi}\right)^2}{A_c} \tau_s = \alpha \, \kappa \, \tau_s \tag{3-17}$$

It is left to the user to choose an adequate value for each case interactively. This parameter is needed because we concentrate the occlusion in the cone domain along its axis. As a consequence, the method tends to block more light than it should. Different obstacles in the cone domain do not necessarily overlap along a ray; however, by combining the influence of all obstacles along the cone axis, all obstacles inevitably accumulate contributions to the transparency reduction.

## 3.5 Results and Discussion

The proposed approach was incorporated in a GPU-based ray casting volume renderer for structured datasets. Our implementation employs early ray termination (whenever the opacity reaches 99%) to increase performance. In all experiments, we set the parameters as discussed in Chapter 3.4, unless otherwise noted. For efficiency, the ray tracing was implemented using GLSL compute shader [52].

For quality comparison, we implemented an algorithm that uses Monte Carlo integration. For each sample along the primary ray, we trace a large set of secondary rays covering the cone region in order to evaluate the transparency accurately. For this implementation, we have set $h_0 = 1.0$ as the initial gap to avoid self-occlusion, $h_p = 0.5$ as the integration step for the primary rays, and $h_s = 0.5$ for the secondary rays. This implementation is far from achieving interactive rates; each frame takes seconds to be rendered in our naive implementation. The goal here is to produce images adopted as a ground truth. We also use the LAB color difference [56] to measure the error between each implemented strategy.

We have run a set of computational experiments with different regular datasets for testing the proposed method. We generated two synthetic datasets for comparison. The first dataset, named *Synthetic*, is composed of boxes with a transfer function that makes them opaque, used to validate directional ambient occlusion computation. The second synthetic dataset, named *SyntheticBars*, is composed of bars far from the ground, used to test shadow generation. The other datasets are known models commonly used in previous works [1]. Table 3.1 shows the list of datasets used in the experiments, with the

[1]Datasets from The Volume Library, available at http://schorsch.efi.fh-nuernberg.de/data/volume/ (accessed on Apr 13, 2019) and from Dep. of

Table 3.1: Datasets used in our experiments.

| Dataset | Volume Size | Voxel Size |
|---|---|---|
| Synthetic | $128 \times 128 \times 128$ | [1.00, 1.00, 1.00] |
| SyntheticBars | $512 \times 256 \times 256$ | [1.00, 1.00, 1.00] |
| VisMale | $128 \times 256 \times 256$ | [1.58, 0.99, 1.01] |
| Engine | $256 \times 256 \times 256$ | [1.00, 1.00, 1.00] |
| Bonsai | $256 \times 256 \times 256$ | [1.00, 1.00, 1.00] |
| Backpack | $512 \times 512 \times 373$ | [0.98, 0.98, 1.25] |
| CT-Knee | $379 \times 229 \times 305$ | [1.00, 1.00, 1.00] |
| Foot | $256 \times 256 \times 256$ | [1.00, 1.00, 1.00] |
| Hazelnut | $512 \times 512 \times 512$ | [1.00, 1.00, 1.00] |
| Flower | $1024 \times 1024 \times 1024$ | [1.00, 1.00, 1.00] |

corresponding discretization and voxel size. We employed piecewise linear 1D transfer functions to maps scalar values.

In the following sections, we present the results and make a comparative analysis with previous works. We also discuss the limitations of the proposed strategy. We start by comparing quality and performance results on directional ambient occlusion, followed by comparison on shadow generation. We then illustrate the effectiveness of our method by combining directional ambient occlusion and shadows. Finally, we discuss preprocessing performance and memory requirements. The light evaluation is performed in image space, unless otherwise mentioned. All the tests were performed on a i7-8700 3.20 GHz computer with a NVIDIA GeForce RTX 2080 Ti graphics card with 11 Gb of memory.

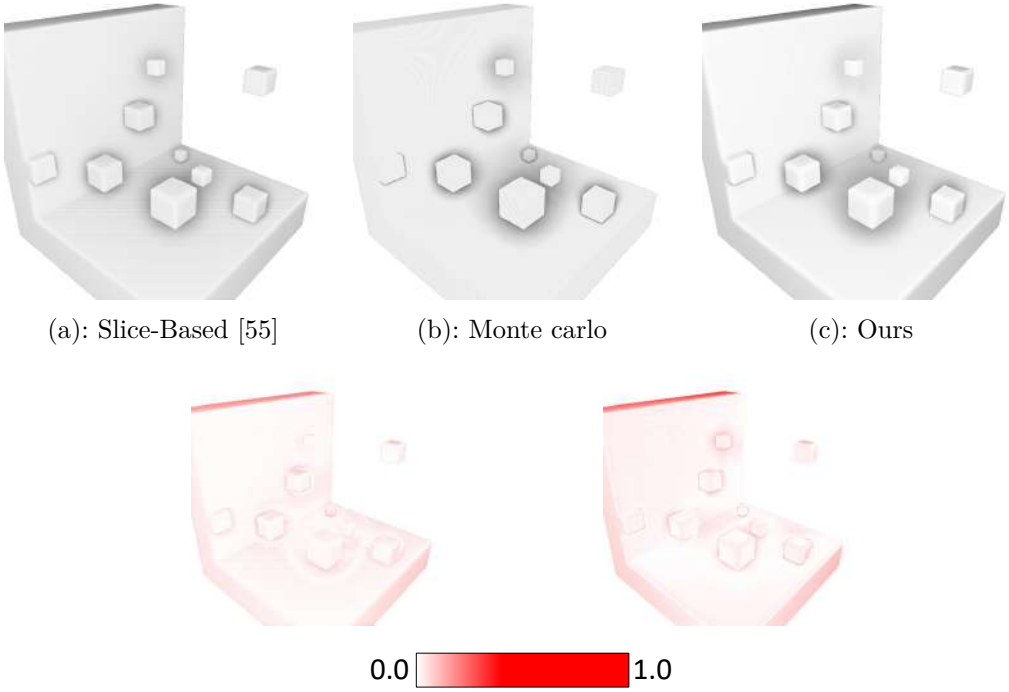


### 3.5.1 Directional ambient occlusion comparison

For directional ambient occlusion comparison, we implemented the slice-based approach proposed by Schott et al. [55]. We did our best to get an efficient implementation. In the tests, we employ $h = 0.5$ as the separation between consecutive slices. Their method has been cited as a reference of quality directional ambient occlusion for volume datasets. The method was implemented in a fragment shader since it requires rasterization of slices.

The slice-based approach has a different concept concerning cone aperture. We made a manual adjustment to its cone aperture angle to generate the results as similar as possible to the Monte Carlo integration for each dataset. For our method, we always used the same cone aperture as the Monte Carlo in-

Informatics - Vis. and Multimedia Lab, University of Zurich, available at https://www.ifi.uzh.ch/en/vmml/research/datasets.html (accessed on Jun 7, 2019)




(a): Slice-Based [55] (b): Monte carlo (c): Ours

(d): LAB Color Difference

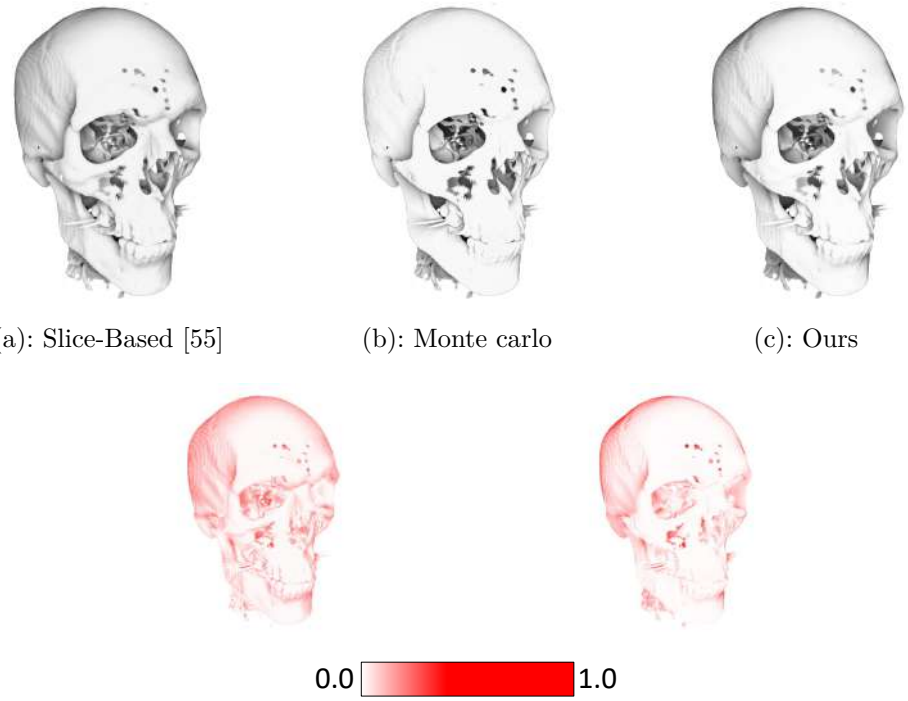
Figure 3.9: Image quality comparison for the Synthetic model with cone angle aperture of 10°.

tegration. For both methods, we interactively adjusted the global attenuation factor $\alpha$ to better approximate the Monte Carlo result.

We first ran a test with the Synthetic dataset. As shown in Figure 3.9, both Schott et al.'s method and ours were able to deliver good results. We note that our method presented a better approximation on shadowed regions. In all these tests, we used $C_{split} = 7$.

Note, however, that the achieved overall illumination differs from the Monte Carlo integration. There are two reasons for such a difference. First, there is the choice of the initial gap, $h_0$, to avoid self-occlusion. As we employ a piecewise linear 1D transfer function, some amount of self-occlusion is inevitable. It is hard to find correspondence of $h_0$ between the Monte Carlo and our methods; we have fixed it to $h_0 = 3\sigma_0$ considering the achieved quality for all tested models. The second reason is related to oblique opaque regions. This effect is a limitation of any cone tracing strategy for directional ambient occlusion. As the cone is pointed towards the viewer, part of it enters the oblique opaque region. The combination of successive samples partially occluded artificially decreases the transparency. This effect is also the reason for the edges being perceived as lighter; along the edges, the cones are completed free of obstacles, preserving the real transparency.

The darkness of oblique opaque regions can also be noted in the VisMale




(a): Slice-Based [55] (b): Monte carlo (c): Ours

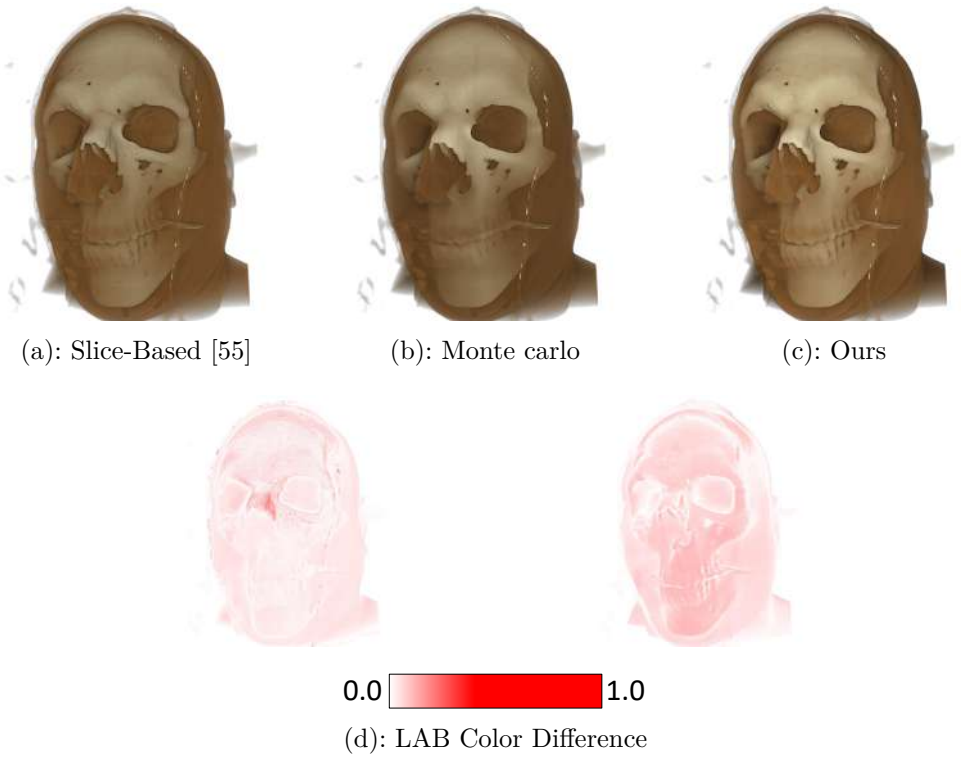
(d): LAB Color Difference

Figure 3.10: Image quality comparison for the VisMale dataset with cone angle aperture of 20°.

dataset, as illustrated in Figure 3.10. In this case, as can be noted, our method presents a better result for capturing the occlusion of internal parts of the skull.

In a third experiment, we again used the VisMale dataset but now with a different transfer function. This transfer function results in a large region of semi-transparent voxels. Figure 3.11 illustrates the achieved results. Note that the semi-transparent region appears lighter in our method. Handling semi-transparent regions, with constant opacity values, is a major limitation of the proposed strategy. We assume a Gaussian distribution of opacity values; in regions where the opacity is uniform, the Gaussian integral results in values smaller than it should. Nevertheless, the overall occlusion of the scene is coherently captured and still close to the result from using Monte Carlo integration.

Table 3.2 shows the achieved performance for both Schott et al.'s method and ours, considering different datasets. As can be noted, in general, our method performed better. The main reason is the use of early termination in our ray tracing. Krüger and Westermann [31] proposed a way to implement early ray termination for slice-based volume rendering. However, their strategy cannot be used in Schott et al.'s proposal, because the occlusion layer needs to be continuously updated, requiring the rendering of all slices.




(a): Slice-Based [55] (b): Monte carlo (c): Ours

(d): LAB Color Difference

Figure 3.11: Image quality comparison for the VisMale(T) dataset with cone angle aperture of 25°.

Note that Schott's method beat ours for three models, VisMale(T), CT-Knee and Foot; these models were rendered with a large region of semi-transparent voxels, in which cases early termination does not help. Nevertheless, for such cases, as stated in Section 3.3.3, we can employ object space illumination, computing a transparency volume before rendering. Table 3.3 shows the gain in performance for the models, using different dimensions of the precomputed volume, for different screen resolutions. Figure 3.12 illustrates that the loss in image quality is subtle. In Appendix C, we added more results comparing Monte Carlo with our illumination technique.

### 3.5.2 Shadow comparison

For shadow comparison, we implemented two previous works: the method proposed by Schlegel et al. [54], which uses summed area table (SAT) for directional shadow generation, and a single-GPU version of the Voxel Cone Tracing method from Shih et al. [58]. For all tested datasets, we used point light sources, and the light computation was performed for each sample along the viewing rays.

We first test the algorithm for the SyntheticBars dataset. In this test, as can be noted in Figure 3.13, our method achieved better image quality

Table 3.2: Performance comparison between ours and Schott et al.'s methods, using a viewport of $768^2$. The model VisMale(T) refers to the VisMale with semi-transparency region. The second column indicates the cone length, defined by the volume diagonal $D$, according to Section 3.4.

| | | Frame Rate (*fps*) | |
|---|---|---|---|
| Dataset | $0.5D$ | Slice-Based [55] | Ours |
| Synthetic | 110.85 | 116 | 273 |
| VisMale | 207.58 | 67 | 233 |
| VisMale(T) | 207.58 | 73 | 49 |
| Engine | 221.70 | 52 | 90 |
| Bonsai | 221.70 | 58 | 110 |
| Backpack | 423.50 | 27 | 83 |
| CT-Knee | 268.84 | 60 | 22 |
| Foot | 221.70 | 59 | 50 |

Table 3.3: Performance for occlusion comparison using two viewports with sizes $768^2$ and $1000^2$, considering our illumination method being evaluated in image and object spaces.

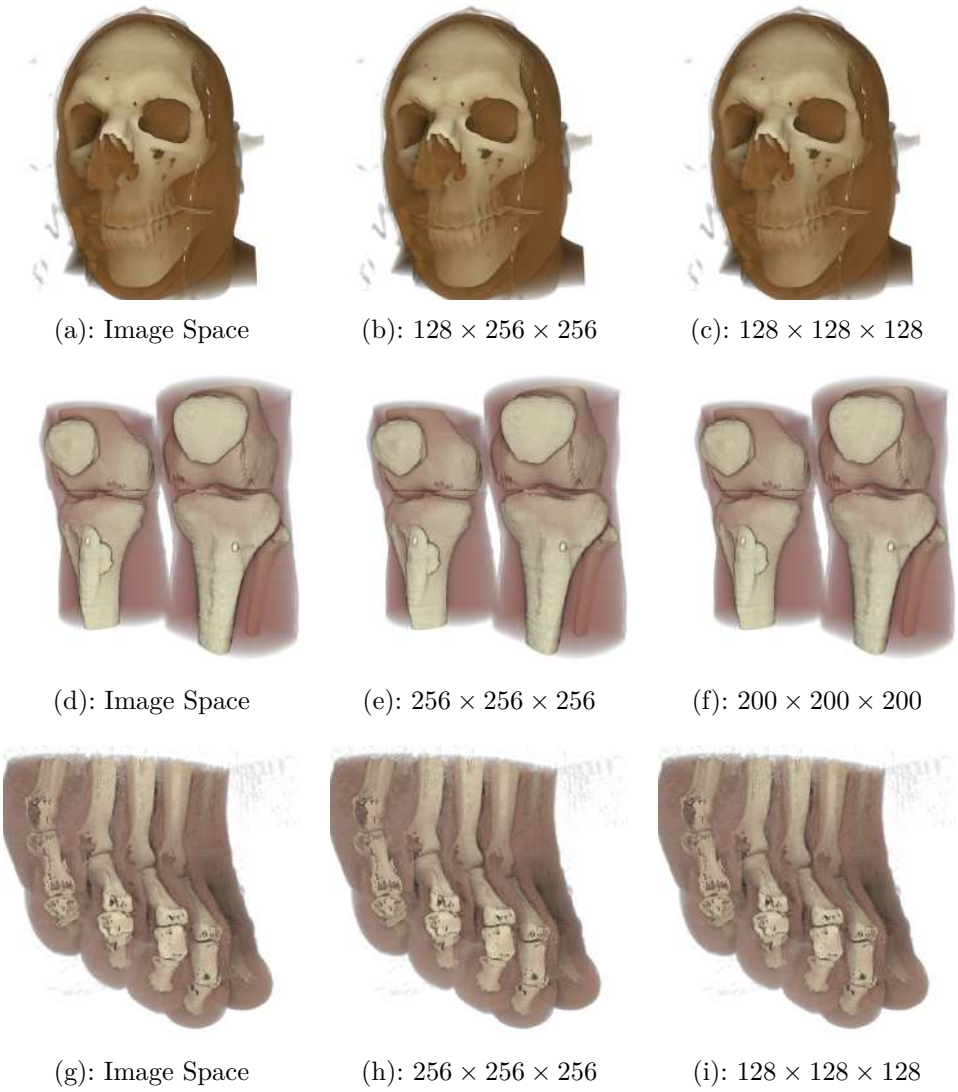
| | | | Frame rate (fps) | |
|---|---|---|---|---|
| Dataset | Method | Dimension | $768^2$ | $1000^2$ |
| VisMale(T) | Slice-Based[55] | – | 73 | 45 |
| | Ours image | – | 49 | 30 |
| | Ours object | $128 \times 256 \times 256$ | 159 | 140 |
| | Ours object | $128 \times 128 \times 128$ | 376 | 290 |
| CT-Knee | Slice-Based[55] | – | 60 | 39 |
| | Ours image | – | 22 | 13 |
| | Ours object | $256 \times 256 \times 256$ | 51 | 49 |
| | Ours object | $200 \times 200 \times 200$ | 96 | 88 |
| Foot | Slice-Based[55] | – | 59 | 38 |
| | Ours image | – | 50 | 33 |
| | Ours object | $256 \times 256 \times 256$ | 98 | 88 |
| | Ours object | $128 \times 128 \times 128$ | 313 | 231 |



than the previous proposals. Shadows cast by distant bars get blurred in the Schlegel et al.'s method and are enlarged in the Shih et al.'s method. The shadows achieved by our proposed method are quite close to the ones obtained with the Monte Carlo integration. However, for achieving such good result, we needed to set $C_{split} = 3$. Without splitting, shadows cast by distant bars also became blurred.

For the Engine dataset, as illustrated in Figure 3.14, our method again delivered a result closer to the one achieved by using the Monte Carlo integration. Again, previous proposals were not able to generate shadows casted by distant objects accurately. Note the main shadow casted on the floor: Schlegel et al.'s method enlarged this shadow, and Shih et al.'s method was not able to capture it faithfully.

Table 3.4 shows a performance comparison. Shih et al.'s method [58]



(a): Image Space (b): $128 \times 256 \times 256$ (c): $128 \times 128 \times 128$

(d): Image Space (e): $256 \times 256 \times 256$ (f): $200 \times 200 \times 200$

(g): Image Space (h): $256 \times 256 \times 256$ (i): $128 \times 128 \times 128$

Figure 3.12: Directional ambient occlusion results of our method for VisMale, CT-Knee, and Foot datasets, either in image space (left) and object (middle and right) space, with viewport set to $768^2$.



presented better performance, while ours presented competitive performance to Schlegel et al.'s method [54]. Again, the performance of our method can be improved with precomputed transparency. In particular, for a static scene, the precomputed transparency volume has to be built only once for shadow generation.

### 3.5.3 Pre-processing analysis

The representative extinction coefficient volume has to be built each time the transfer function changes, as discussed in Section 3.2.1. In this section, we present memory and time requirements to build the volume. The Gaussian filters were applied using a $7 \times 7 \times 7$ kernel per voxel, in the range $[-3\sigma, 3\sigma]$ on

Table 3.4: Performance comparison for shadow generation using a viewport of $768^2$. The second column indicates the cone length based on the volume diagonal $D$, according to Section 3.4.

| | | Frame Rate (*fps*) | | |
|---|---|---|---|---|
| Dataset | $0.75D$ | SAT [54] | VCT [58] | Ours |
| SyntheticBars | 470.30 | 89 | 178 | 83 |
| Engine | 332.55 | 74 | 97 | 63 |

Table 3.5: Preprocessing times to generate the Extinction Coefficient Volume.

| Volume Size | MipMap Levels | Average Time (*ms*) |
|---|---|---|
| $128 \times 128 \times 128$ | 7 | 57.85 |
| $256 \times 256 \times 256$ | 8 | 220.97 |
| $512 \times 512 \times 512$ | 9 | 995.12 |
| $379 \times 229 \times 305$ | 8 | 1729.99 |
| $512 \times 512 \times 373$ | 9 | 3249.57 |
| $1024 \times 1024 \times 1024$ | 10 | 6737.62 |

each direction (each sample is distant $\sigma$ from each other). In its original form, with the extinction volume having the same resolution as the original dataset, the method requires an amount of additional memory equal to $1 + 1/7$ of the original volume.

Table 3.5 shows the preprocessing times for different model resolutions. Note that a volume dimension up to $256^3$ can be preprocessed in real time, allowing interactive transfer function changes. For higher resolution, the interaction would suffer. In order to mitigate the preprocessing time and memory requirements, it is possible to use a smaller resolution for the extinction coefficient volume. The construction is similar; for each voxel the Gaussian filter is applied in the same range, considering the original dataset scales.

Figure 3.15 illustrates the results for Hazelnut and Flower datasets, reducing the resolution of the coefficient volume from 512 to 256 to 128 and from 1024 to 512 to 256, respectively. Note that the proposed method still deliver quality images. However, for smaller resolutions, the lighting effects would be blurrier with fewer occlusion details, possibly with some artifacts.

### 3.5.4 Combining directional ambient occlusion and shadows

As far as we know, this work is the first to propose a strategy that can be used both for directional ambient occlusion and shadow computations with good quality. Previous methods for one effect cannot be extended to handle the other with the same quality. Schott et al.'s method [55], a slice-based algorithm, would be impractical for shadow generation. Schlegel et al.'s method[54], based on SAT, would face problems on computing directional

Table 3.6: Performance results with directional occlusion and shadows using a viewport of $768^2$; when pre-illumination is employed (object space light evaluation), the corresponding volume resolution is annotated.

| Dataset | Ext. Coef. Vol. Res. | Pre-illumination | Parameters | | | Frame Rate (*fps*) | | |
|---|---|---|---|---|---|---|---|---|
| | | | $\theta_o$ | $C_{split}$ | $\theta_s$ | EA | AO | AO+SH |
| VisMale | $128 \times 256 \times 256$ | $128 \times 128 \times 128$ | 25 | 7 | 2.0 | 783 | 376 | 263 |
| Bonsai | $256 \times 256 \times 256$ | – | 20 | 3 | 0.5 | 458 | 155 | 64 |
| Backpack | $512 \times 512 \times 373$ | – | 15 | 7 | 1.0 | 235 | 83 | 56 |
| CT-Knee | $256 \times 256 \times 256$ | $200 \times 200 \times 200$ | 10 | 3 | 2.0 | 630 | 183 | 103 |
| Foot | $256 \times 256 \times 256$ | $256 \times 256 \times 256$ | 30 | 3 | 1.0 | 511 | 163 | 58 |
| Hazelnut | $512 \times 512 \times 512$ | $256 \times 256 \times 256$ | 25 | 7 | – | 219 | 61 | – |
| Flower | $512 \times 512 \times 512$ | – | 25 | 7 | – | 48 | 28 | – |

ambient occlusion because of alignment issues; the effect would vary as the viewer moves. Shih et al.'s method [58] would also suffer for ambient occlusion, where we need to use cones with larger apertures; large super voxels tend to miss obstacle details.

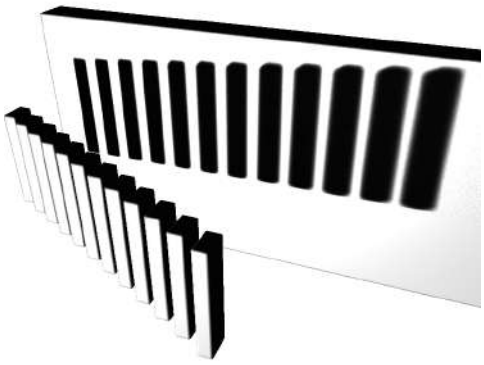
We ran additional computational experiments to demonstrate the effectiveness of our proposal for combining the effects. To test the proposal for different cases, we ran the tests with different cone apertures, for both the ambient occlusion cone ($\theta_o$) and the shadow cone ($\theta_s$), and different maximum number of cones per section ($C_{split}$). Table 3.6 shows the used parameters together with the performance achieved for three configurations: emission-absorption illumination model (EA), emission-absorption with directional ambient occlusion (AO), and emission-absorption with directional ambient occlusion and shadows due to a point light source, using Blinn-Phong illumination model (AO+SH). Note, in particular, that the CT-Knee and Foot models had their performances improved if compared to Table 3.3. The performance for the CT-Knee model with a precomputed transparency of $200^3$ improved from 96 to 183, while the performance for the Foot model with a precomputed transparency of $256^3$ improved from 98 to 163. The reason for such gains is that we reduced the $C_{split}$ parameter from 7 to 3. Figure 3.16 illustrates the obtained images. Besides the ambient occlusion effect, note how the shadows are correctly captured by our proposed approach.

### 3.5.5 Downscaling and Upscaling Filtering

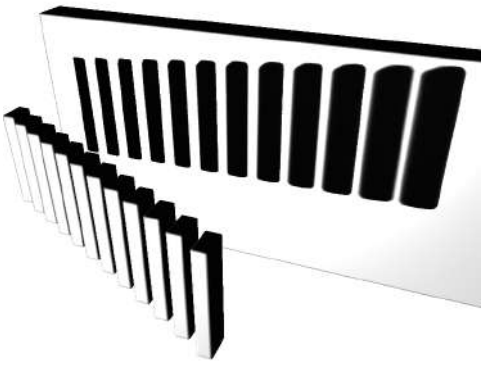
We also investigate the use of 2D interpolation filters for downscaling and upscaling [48] on the results of volumetric visualizations. Our goal is to see if we can produce quality images for volume exploration drawing in a lower resolution. We also tested the gain in quality if a higher resolution was employed. The filter was implemented on GPU, with each thread evaluating an entire column and row of the resulting image; better, optimized solutions

can be found in [47, 49].

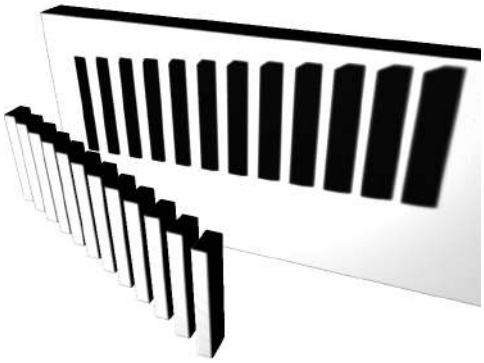
Figure 3.17 shows the achieved results using directional ambient occlusion. As can be noted, in general, upscaling delivers better performance without compromising image interpretation; for the VisMale model, downscaling captured smoother volume boundaries, while upscaling still provided a good representation of the volume, almost doubling the performance. For the Bonsai dataset, upscaling introduced noise in the final result (see the vessel) but still generates a good enough representation for volume exploration, especially because the performance was significantly improved. Here, downscaling eliminates the noise. For the Hazelnut model, upscaling did not deliver a significant gain in performance since we used pre-illumination, which is not affected by the image resolution. A similar result was achieved for the Flower model.



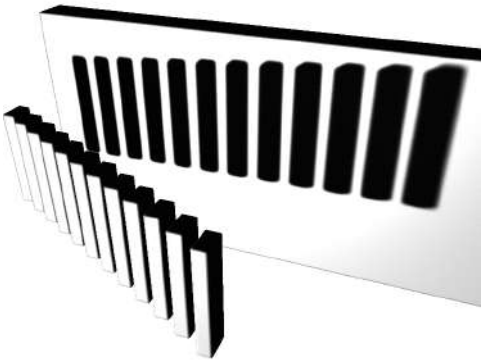
(a): SAT [54]



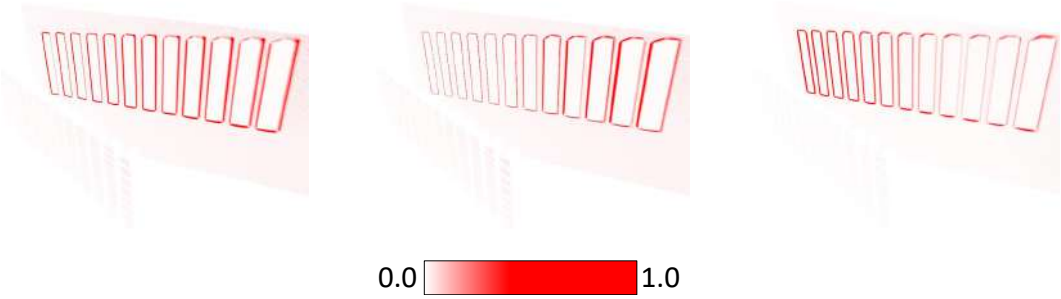
(b): VCT [58]



(c): Monte Carlo

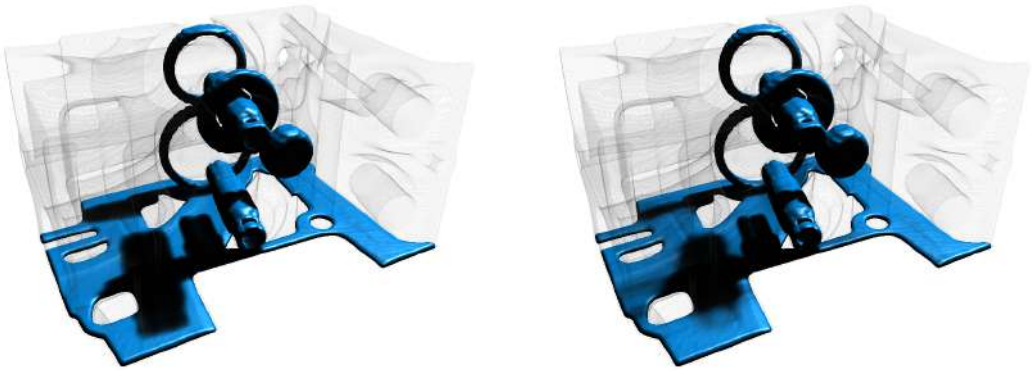


(d): Ours




(e): LAB Color Difference

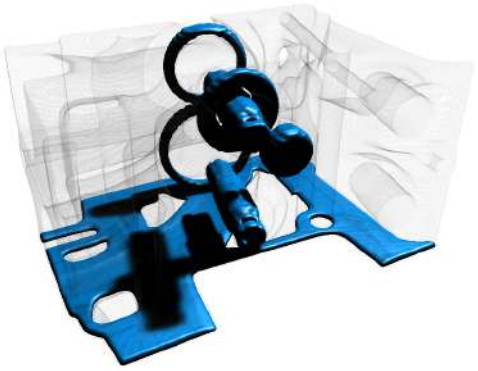
Figure 3.13: Shadow quality comparison of SyntheticBars dataset using a cone angle aperture of 0.5°.



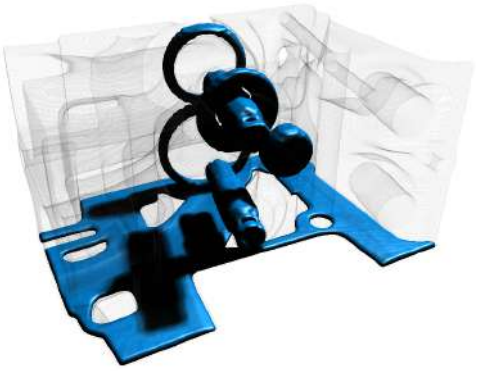
(a): SAT [54]



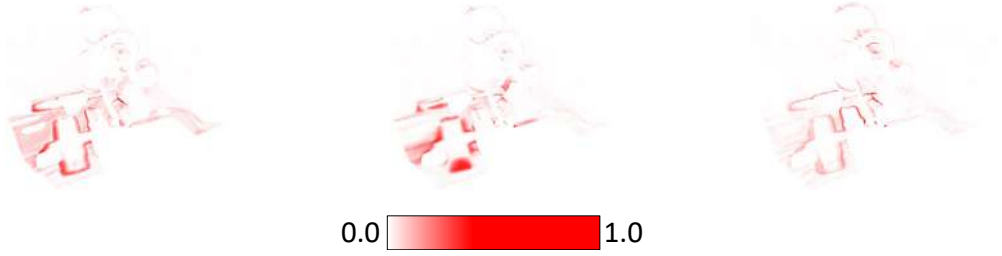
(b): VCT [58]



(c): Monte Carlo



(d): Ours



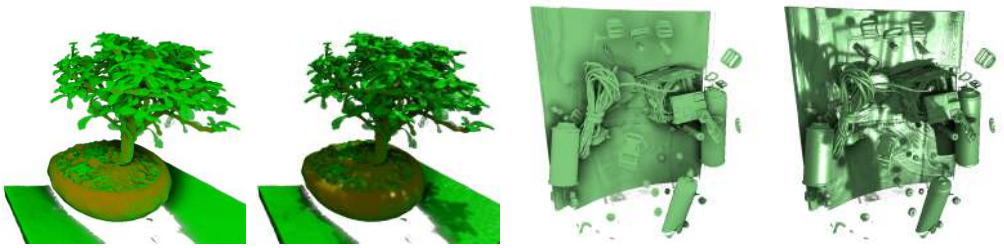

(e): LAB Color Difference

Figure 3.14: Shadow quality comparison for Engine dataset using a cone angle aperture of 2.0°.

(a): $512^3$ (b): $256^3$ (c): $128^3$

(d): $1024^3$ (e): $512^3$ (f): $256^3$

Figure 3.15: Hazelnut and Flower datasets rendered with directional ambient occlusion ($\theta_o = 25^o$), with different extinction coefficient volume resolutions.



(a): Bonsai (b): Backpack

(c): CT-Knee (d): Foot

Figure 3.16: Achieved results combining directional ambient occlusion and shadow generation: on the left, emission and absorption illumination model with directional occlusion; on the right, shadows with Blinn-Phong shading are added.

(a): Original (233 fps) (b): Downscaling (80 fps) (c): Upscaling (427 fps)

(d): Original (155 fps) (e): Downscaling (52 fps) (f): Upscaling (358 fps)

(g): Original (61 fps) (h): Downscaling (37 fps) (i): Upscaling (72 fps)

(j): Original (28 fps) (k): Downscaling (10 fps) (l): Upscaling (37 fps)

Figure 3.17: Results from VisMale, Bonsai, Hazelnut and Flower datasets, using a viewport of $768^2$ and considering the parameters from Section 3.5.1 for VisMale and Table 3.6 for the rest. For Downscaling, we render the image using a viewport of $1536^2$ and then scale back to $768^2$. For Upscaling, we render the image using a viewport of $384^2$, and then extend back to $768^2$. We use the Cardinal Cubic O-MOMS kernel implementation provided by [48], for both Downscaling and Upscaling operations. In this experiment, light is evaluated in object space only for Hazelnut dataset.

# 4
# An experimental study on volumetric visualization of black oil reservoir models

In this chapter, we present our second contribution, showing an experimental study on volumetric visualizations of black oil reservoirs. We present three strategies based on ray casting and made a comparative result of quality and performance, also extending our directional ambient occlusion technique to improve the perceptual results.

## 4.1
## Ray Casting Algorithms for Black Oil Reservoirs

A black oil reservoir model is represented by a mesh of hexahedral cells lying on a topological grid. Each cell is identified by its topological coordinates $[i, j, k]$. The geometry is irregular, and the mesh may present discontinuities, which model geological faults and non-permeable regions (represented by *inactive* cells). In the absence of local discontinuities, the cell $[i, j, k]$ shares faces with the cells $[i\pm1, j, k]$, $[i, j\pm1, k]$, and $[i, j, k\pm1]$. Figure 4.1 illustrates a reservoir model[1] with geometry discontinuities. Generally, a reservoir simulator assigns the resulting properties (scalar values) to cells. These properties are then transferred to the vertices by averaging neighboring cell values, resulting in a smooth scalar field.

It is assumed that, for each hexahedral cell, any property $\alpha$ within the cell is computed from the values at the eight vertices $\alpha_i$. This is done by a trilinear interpolation:

$$\alpha = \sum_{i=1}^{8} N_i \alpha_i$$

where $N_i = f_i(s, t, r)$ are the *shape (or interpolation) functions*, with $(s, t, r) \in [-1, 1]^3$ being the parametric coordinates in the cell. This interpolation also holds for the physical coordinate $(x, y, z)$ of any point within the cell.

[1]Model from "UNISIM-II: Benchmark Case Proposal Based on a Carbonate Reservoir". Available at: https://www.unisim.cepetro.unicamp.br/ benchmarks/br/unisim-ii/overview (30 July, 2020).

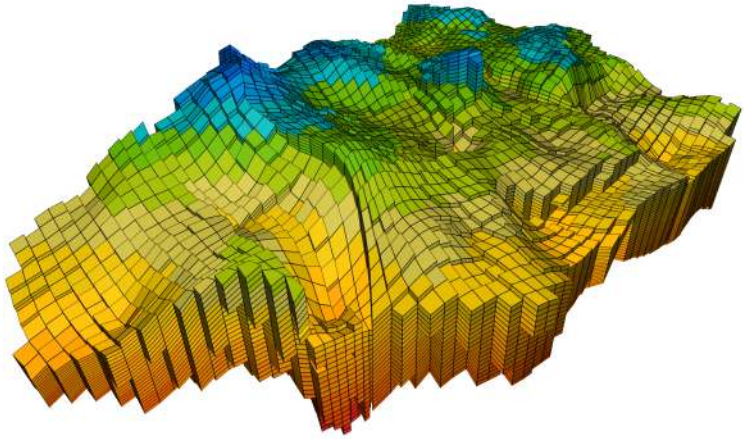


Figure 4.1: Example of reservoir model with geometry discontinuities.

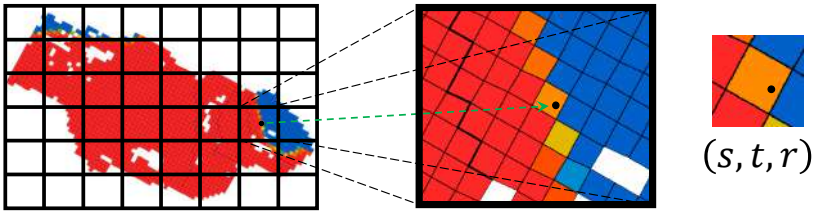



Figure 4.2: Franceschin et al.'s proposal [21] starts the point location procedure by identifying the corresponding voxel ID from a regular grid; then, it applies a Newton-Raphson procedure to check if each cell intercepted by the voxels contains the point, returning its parametric coordinate, if the case. The low resolution regular grid drawn above is for illustrative purposes; in practice, we set $r = 1.75$, where each voxel ends up being smaller than the average cell size.

Franceschin et al. [21] explored the topological layout and proposed a compact GPU-based representation for reservoir models. They also proposed an accurate and efficient point location algorithm: given a point coordinate $(x, y, z)$, the algorithm returns the cell, if any, containing the point and the corresponding parametric coordinate $(s, t, r) \in [-1, 1]^3$. Due to the high level of geometry distortion and the presence of bad-formed hexahedral cells, the point location algorithm uses an iterative Newton-Raphson procedure, limiting its use for many queries in realtime. As illustrated in Figure 4.2, their proposal uses an additional regular grid to accelerate the query. The resolution of this regular grid is provided by a single scalar factor $r$ that correlates the topological to the regular grid resolutions. If the reservoir topological model has $n_i \times n_j \times n_k$ cells, the regular grid will have $r\,n_i \times r\,n_j \times r\,n_k$ voxels. Therefore, the larger the $r$ factor, the faster are the queries, but the larger is the memory footprint.

Based on such a support for point location, it is straightforward to implement a volume visualization algorithm for reservoir models. The volume integral can be simply computed by a Riemann sum, evaluating the scalar

field at samples along each traced ray. As the point location procedure returns the parametric coordinate of the sample inside a cell, it is easy to retrieve the scalar field from the cell vertices using the hexahedral shape functions.

In summary, the data structure proposed by Franceschin et al. [21] works as a regular 3D texture. One can increase accuracy by using more and more samples along the rays. However, it does not perform at interactive rates.

To gain performance, we explore three different strategies. The first one consists of resampling the reservoir scalar field in a regular grid, proceeding with a regular volume visualization tool for structured data. The other two consider the model as an unstructured hexahedral mesh.

### 4.1.1 Structured Resampled Grid

The first strategy consists of creating a regular volume, resampling the scalar field of the reservoir model. At each voxel, we store the associated property value and an additional $\alpha_p$ value that indicates if the voxel is in the reservoir (value 1) or not (value 0). The $\alpha_p$ value is needed to correctly handle the reservoir boundaries, as illustrated by Figure 4.3. In the rendering stage, $\alpha_p$ is multiplied by the current opacity along each ray. The resampled regular grid resolution is given by $r' n_i \times r' n_j \times r' n_k$, where $r'$ is a grid refinement factor, while $n_i \times n_j \times n_k$ is the reservoir topological grid dimension. Note that the reservoir cells are, in general, not aligned to the regular grid. Enlarging the regular grid resolution tends to improve accuracy.

To improve performance, we define the ray initial and final points drawing the external front and back, respectively, faces of the reservoir model. For computing the integration, the rays are uniformly sampled inside the model.

The regular volume has to be rebuilt each time the scalar field is changed. In reservoir simulation analysis, it is common to play an animation of the scalar field along simulation time. So, this preprocessing cost may affect the appropriateness of this strategy in practice. However, it performs very well once the regular volume is built. We use the GPU model representation [21] to compute the regular volume more efficiently. In theory, once we have the volume, we could discard the model representation, but we maintain it for supporting updates of the scalar field. Therefore, this strategy also represents a significant additional memory cost.

The significant advantage of this strategy is its simplicity and the ability to use any existing algorithm for regular volume visualization. However, the achieved result lacks accuracy. Even employing high grid resolution, discontinuities due to geological faults and cell misalignment may not be accurately

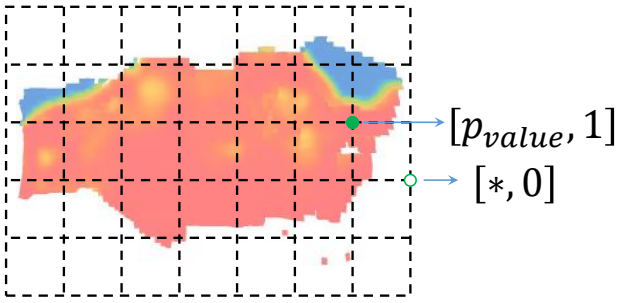



Figure 4.3: We create an additional 2-channel 3D texture to store the property value and $\alpha_p$, which is multiplied by the current opacity along each ray in the rendering stage. This image is illustrative; the grid resolution is $r'$ higher than the reservoir topological dimension.

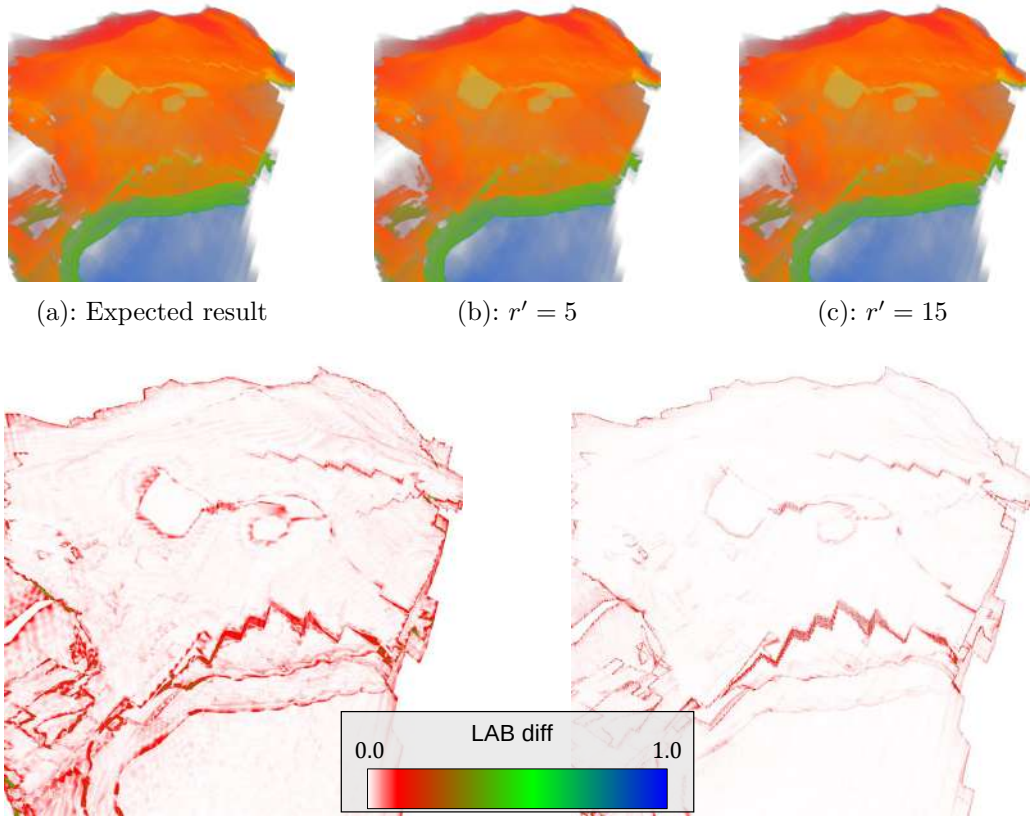
represented due to trilinear interpolation of scalar values, as illustrated by Figure 4.4. Additionally, the use of high-resolution grids may be prohibitive because of the required memory space and preprocessing time [28]. Previous works [62, 64, 42] also have shown that large 3D textures may abruptly decrease the frame rate, because samples covering different image slices may be far from each other in the memory space.

The significant advantage of this strategy is its simplicity and the ability to use any existing algorithm for regular volume visualization. However, the achieved result lacks accuracy. Even employing high grid resolution, discontinuities due to geological faults and cell misalignment may not be accurately represented due to trilinear interpolation of scalar values, as illustrated by Figure 4.4. Additionally, the use of high-resolution grids may be prohibitive because of the required memory space and preprocessing time [28]. Previous works [62, 64, 42] also have shown that large 3D textures may abruptly decrease the frame rate, because texels from consecutive image slices may be far from each other in the memory space. Figure 4.5 shows an example of how performance degrades when choosing higher refinement factors.

### 4.1.2 Linear Interpolation with Barycentric Coordinates

In the second strategy, we employ several simplifications to improve performance while considering the hexahedral mesh. As in [41], each hexahedral face is subdivided into two triangles. Once each ray enters the first cell, we compute the exit point using a ray-triangle intersection test [43].

Within each cell, given the entry and exit points, with respect to the face triangles, we use the triangle barycentric coordinates to approximate the cell




(a): Expected result (b): $r' = 5$ (c): $r' = 15$

(d): LAB color difference computed using $r' = 5$ and $r' = 15$

Figure 4.4: When using a resampled regular grid, we must use a proper resolution; otherwise, the reservoir model is not adequately represented. Still, geological faults might be misrepresented even when using higher resolutions.

parametric coordinates: $(s_0, t_0, r_0)$ and $(s_1, t_1, r_1)$. We then assume that inside the cell the parametric coordinate varies linearly along the ray:

$$\begin{bmatrix} s \\ t \\ r \end{bmatrix} = \begin{bmatrix} s_0 \\ t_0 \\ r_0 \end{bmatrix} + \frac{d - d_0}{d_1 - d_0} \left( \begin{bmatrix} s_1 \\ t_1 \\ r_1 \end{bmatrix} - \begin{bmatrix} s_0 \\ t_0 \\ r_0 \end{bmatrix} \right)$$

where $d$, $d_0$, and $d_1$ represent the distance from the camera to the current, entry, and exit points, respectively.

In this method, we employ the well known depth-peeling technique to handle holes and gaps [67, 4, 19, 41]. It starts by rendering the external front faces of the volume and storing the required data to enter the reservoir for each pixel on the screen. Then, the ray traverses the model until it exits the volume. When it contains holes or gaps, the ray might reenter the volume in the next peel. Color and opacity from each peel are accumulated to generate the final image, and the procedure ends when it reaches the last external face.

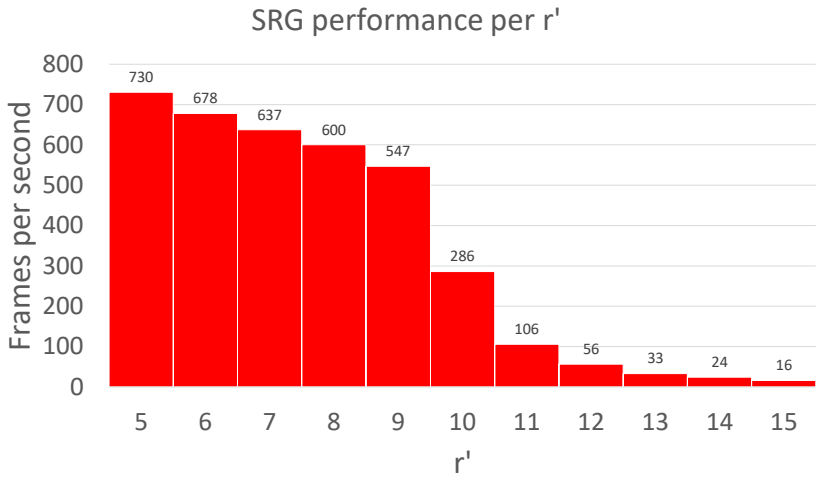



Figure 4.5: The performance may abruptly decrease when using resampled regular grids with higher resolution, generating more texture memory cache hit misses. However, we still must keep a reasonable $r'$ value to generate a good approximation of the reservoir.

We could carefully choose the diagonal that minimizes the geometry error at each hexahedral face. However, preliminary tests showed that choosing different diagonals for different faces introduces an impact on performance. We have opted to employ a fixed quadrilateral subdivision.

This second strategy presents a relatively good performance without the need of pre-computation. However, we have noticed that occasionally the ray-triangle intersection test fails, due to numerical precision, causing pixel artifacts. To reduce such numerical problems, as the cells are not necessarily convex, we always adopt the farthest exit point, thus avoiding the need to handle ray reentering the cells.

### 4.1.3 Point Location with Topological Search

As mentioned before, the point location procedure, from Franceschin et al. [21], has a high computational cost when considering volume rendering applications, since we usually have a high number of samples per ray. Therefore, to gain performance without introducing errors, we investigated using a topological search to optimize subsequent point location queries. Instead of reentering the point location algorithm for each sample, we explore spatial coherence between subsequent samples. We only call the point location algorithm when the topological search fails. The chances are big that the subsequent sample remains in the same cell or nearby. We then directly apply a Newton-Raphson procedure to locate the sample in the previous cell, using the previous parametric coordinate as the initial guess: one or two iterations usually suffice to

locate the point.

In the case the sample remains in the same cell, the vertex scalar values can be reused without extra fetches. If the sample is outside the current cell (the returned parametric coordinate is outside the range $[-1, 1]^3$), we perform a topological search. We combine the cell adjacent list and the returned parametric coordinate to visit the closest, in parametric space, active neighbor cell. This process is repeated until the cell containing the sample is found, or the search reaches the model boundary, as illustrated in Figure 4.6. If an internal boundary is reached, as in geological faults, we rely on the original point location algorithm to advance.

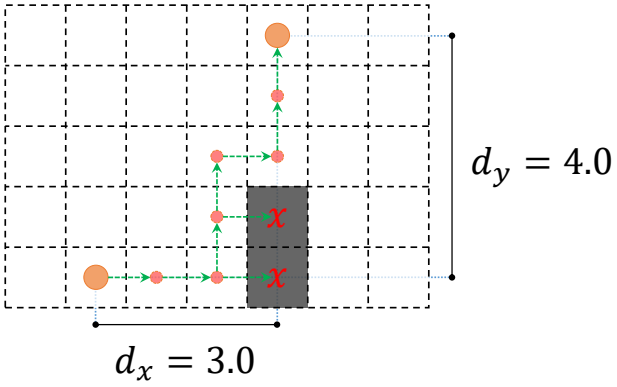



Figure 4.6: When employing a topological search, the parametric distance is evaluated in each direction, proceeding to the lower distance direction. We only pass through active elements, so we must take the element adjacency list.

As in the first described strategy, we render the reservoir's external faces to determine the initial and final points for traversing the rays. The use of the Newton-Raphson method ensures that we accurately retrieve the scalar field value. However, the use of such a faster search may not be enough for achieving interactive rates for large reservoir models.

## 4.2 Adapting Directional Ambient Occlusion Effects

The simple Phong illumination model is not enough to reveal the complex structures of a reservoir model. To enhance the perception, we adopted our proposal to compute ambient occlusion on reservoir model visualization. After a few experiments, we decided to compute ambient occlusion based on a resampled regular grid. Figure 4.7 shows the steps we added in the preprocessing stage to build the mipmap texture with representative extinction coefficient values $\tau_s$. A regular grid, defined by an axis-aligned bounding box enclosing all active reservoir cells, stores the extinction coefficient computed with the point location algorithm, based on a provided transfer function.

The standard $\sigma_0$ value used to define the representative amplitudes is set by averaging the structured voxel size in each direction. The rest of the algorithm and respective parameters work as defined in Chapter 3.4.

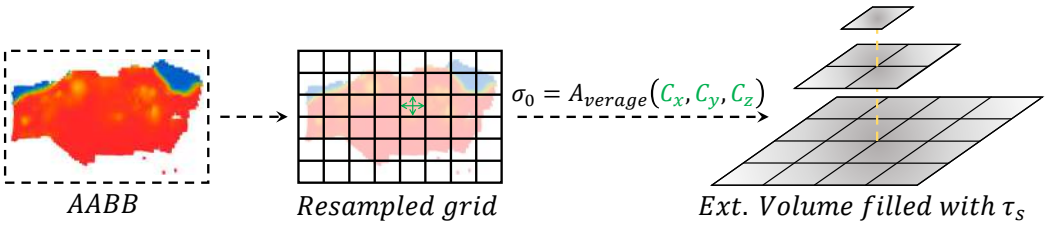


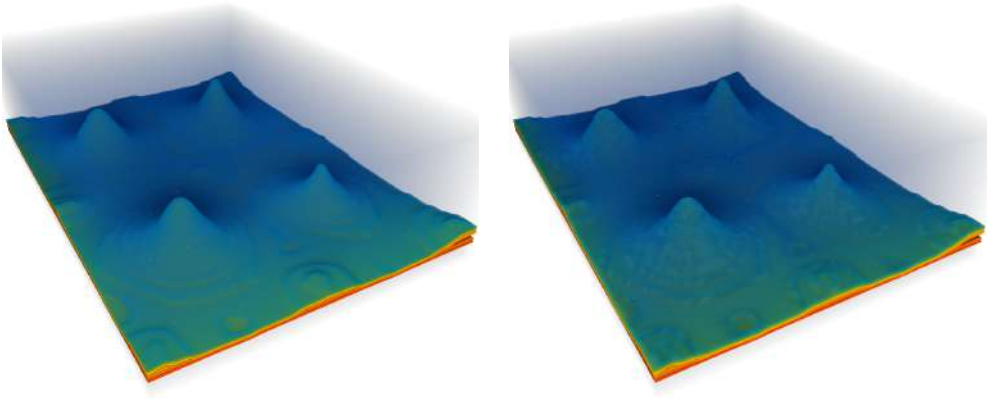

Figure 4.7: We first build a resampled grid based on the AABB enclosing only active elements. We then estimate $\sigma_0$ by averaging the cell sizes. The resampled regular grid is used as input data to generate the volume of extinction coefficients.

Previous works explore the evaluation of incident light at each sample in both image space [9, 58, 15, 55] and object space [9, 57, 70, 58, 54, 2]. In image space, the incident light is computed for each sample along the viewing ray. In object space, it is precomputed in an additional light volume and only accessed in the rendering stage. In reservoir rendering, a cell is usually traversed by a set of rays. Therefore, to improve performance, we have opted to use object space in our implementation.

Shih et al. [57] proposed to store pre-illumination values at the vertices for unstructured meshes. They argue that an unstructured grid can be highly adaptive; thus, with the use of a resampled regular grid, regions with smaller cells may not be adequately sampled. While this is true, we have noticed that the opposite does not hold. Regions with large cells cannot be sampled only at vertices. Although the scalar field within a cell varies (tri-)linearly, the incident light does not – it depends on the arrangement of other diverse cells in front of each vertex. Figure 4.8 illustrates how incident light stored in vertices may result in a wrong illumination. The figure illustrates a simple reservoir model; therefore, the number of reservoir cells is relatively small, and each cell is projected on a large number of pixels on the screen. Adopting a (tri-)linear interpolation of the incident light introduces artifacts.

## 4.3 Results and Discussion

We have run a set of computational experiments to compare the different visualization strategies applied to a set of reservoir models. The reservoir models used in the tests are listed in Table 4.1. The models are identified by R1, R2, R3, R4, and R5. The reservoir models R2 and R3 present a set of geological faults, introducing complexity and discontinuities to the



(a): Refined regular grid (b): Model vertices

Figure 4.8: Ambient occlusion computed in object space: (a) Precomputed values stored in a refined regular grid; (b) Precomputed values stored at model vertices.

Table 4.1: Reservoir models used in our experiments and base memory requirements from Franceschin et al. [21], using $r = 1.75$.

| Res. model | $ni \times nj \times nk$ | N. Active Elements | $h_p$ | Mem. Req.(MB) |
|---|---|---|---|---|
| R1 | $60 \times 20 \times 10$ | $12,000$ | 6.25 | 1.1 |
| R2 | $76 \times 43 \times 23$ | $28,952$ | 4.96 | 2.3 |
| R3 | $46 \times 69 \times 30$ | $42,353$ | 3.99 | 4.4 |
| R4 | $102 \times 72 \times 40$ | $286,840$ | 9.38 | 17.7 |
| R5 | $346 \times 352 \times 100$ | $6,245,219$ | 3.20 | 211.5 |

hexahedral mesh. Besides the topological resolution and the number of active cells, Table 4.1 shows the integration step along the primary ray $h_p$ [2] and the memory requirement to store each dataset, according to Franceschin et al. [21], considering single-precision floats. The integration steps were computed by averaging the model cell sizes.

### 4.3.1 Memory Comparison

In all rendering strategies, the model's external faces have also to be stored to render the reservoir hull. Besides that, only the regular resampled grid requires extra memory to compute the volume integral. This extra memory is necessary to store the 2-channel 3D texture, which may represent a significant drawback since the other strategies operate directly on the GPU representation of the model.

[2] to achieve good quality, $h_p$ is calculated by taking the minimum averaged cell size from $x$ $y$ and $z$ direction divided by 8, i.e. $h_p = min(Avg_{CellSizeX}, Avg_{CellSizeY}, Avg_{CellSizeZ})/8$.

For each reservoir model, we have chosen the $r'$ factor value trying to balance the achieved rendering quality and the required memory space. Table 4.2 shows the amount of memory required to store the regular resampled grid for each reservoir model, with the corresponding $r'$ factor. Note that for the model R5, about 6 Gb of memory is needed, even for a relatively small factor value ($r' = 4$).

Table 4.2: Additional memory requirements to resample each reservoir, using different resolutions $r'$.

| Reservoir model | $r'$ | Memory (MB) |
|---|---|---|
| R1 | 17 | 450 |
| R2 | 15 | 1,935 |
| R3 | 11 | 967 |
| R4 | 6 | 484 |
| R5 | 4 | 5,947 |

### 4.3.2 Quality and Performance Comparison

All the computational tests were performed on an i7-8700 3.20 GHz computer with an NVIDIA GeForce RTX 2080 Ti graphics card with 11 Gb of memory. For all rendering strategies, we employed early ray termination (whenever the opacity reaches 99%) to increase performance, preventing the evaluation of samples that do not significantly contribute to the final image. We employ piecewise linear 1D transfer functions manually adjusted for each dataset, based on the automatic boundary detection procedure proposed by Mesquita and Celes [40]. We ensured that all strategies evaluated the integral using the same set of samples per ray.

For quality comparison, we computed the peak signal-to-noise ratio (PSNR) and the structural similarity (SSIM) metrics [65], using the *point location with topological search* strategy as a quality reference. The performance and quality measurements were taken incrementally rotating the model 180° around the around $X$, $Y$, and $Z$ axes, similar to [64]. Figure 4.9 shows some of the evaluated positions for R2, R3 and R5. In the figures and tables that follow, the rendering strategies are identified by the following acronyms: SRG (Structured Resampled Grid), LIBC (Linear Interpolation with Barycentric Coordinates), and PLTS (Point Location with Topological Search).

Figure 4.10 shows the results achieved for the three most representative models, R2, R3, and R5. We tracked the frame rate and the quality metrics for each rendering strategy. The *structured resampled grid* strategy tends to perform better; however, when a high-resolution regular volume is necessary,

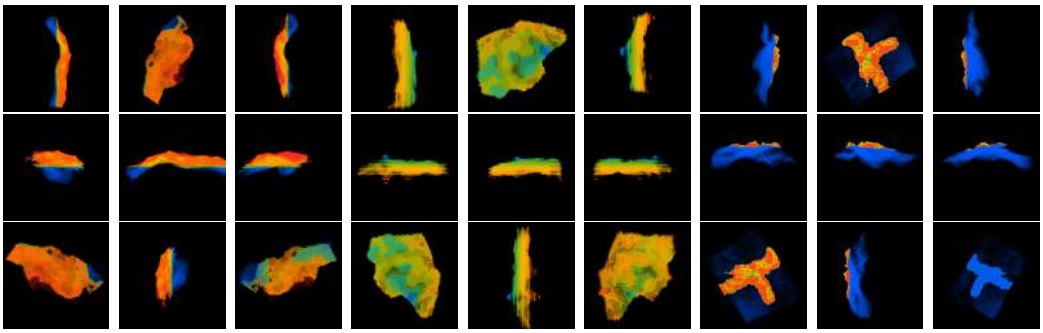


Figure 4.9: Rotation positions for R2, R3 and R5 datasets at 0°, 90° and 180° for $X$, $Y$, and $Z$.

the performance degrades, due to texture hit cache misses. The *linear interpolation with barycentric coordinates* strategy delivered a better balance between performance and quality. The strategy that uses accurate point location could hardly deliver an interactive frame rate.

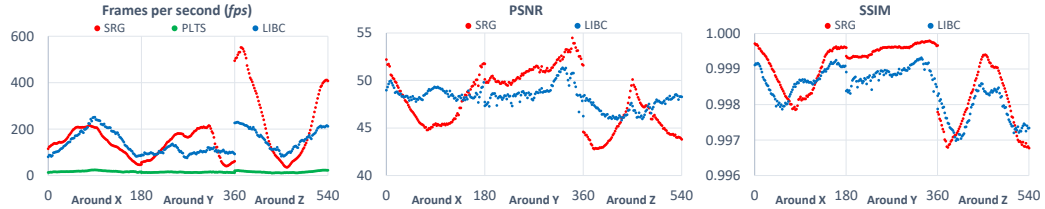
For a visual quality comparison, we compute the LAB color difference [56] between each strategy, again using the most accurate as a reference. Figure 4.11 shows the achieved results, using the same color scale from Figure 4.4. We can note that the LIBC strategy presented some differences due to the linear approximation of the parametric coordinates. The SRG strategy presented worse quality results, especially for revealing discontinuities or when the regular volume resolution was not enough to capture all scalar field variation.



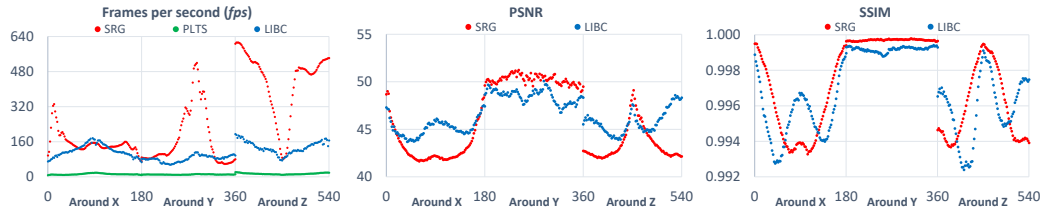
### 4.3.3 Combining with Directional Ambient Occlusion

We ran additional experiments adding directional ambient occlusion to the reservoir visualization. We opted to use the LIBC rendering strategy to compute these results because it achieved a good quality while keeping interactive performance for the tested models. We use different cone aperture angles ($\theta_o$) and maximum numbers of cones per section ($C_{split}$). Table 4.3 shows the used parameters, and Table 4.4 the achieved performance. It also shows the $\sigma_0$ value used, resulting in the resolution of our extinction coefficient volumes. The table also shows the resolution used to precompute the incident light in a pre-illumination step.

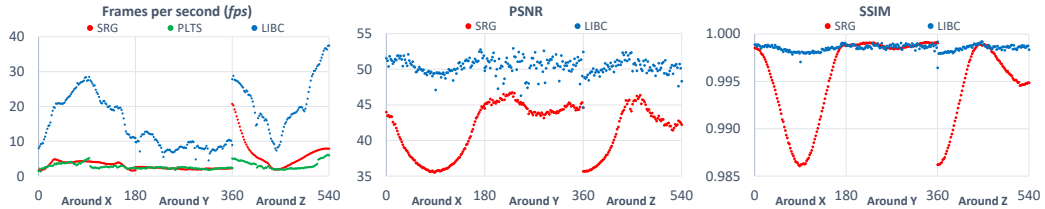
Figure 4.12 shows the achieved results computing only emission and absorption, and then adding directional ambient occlusion. We can see the isosurfaces are more prominent after adding illumination effects, offering better volume perception.




(a): R2




(b): R3




(c): R5

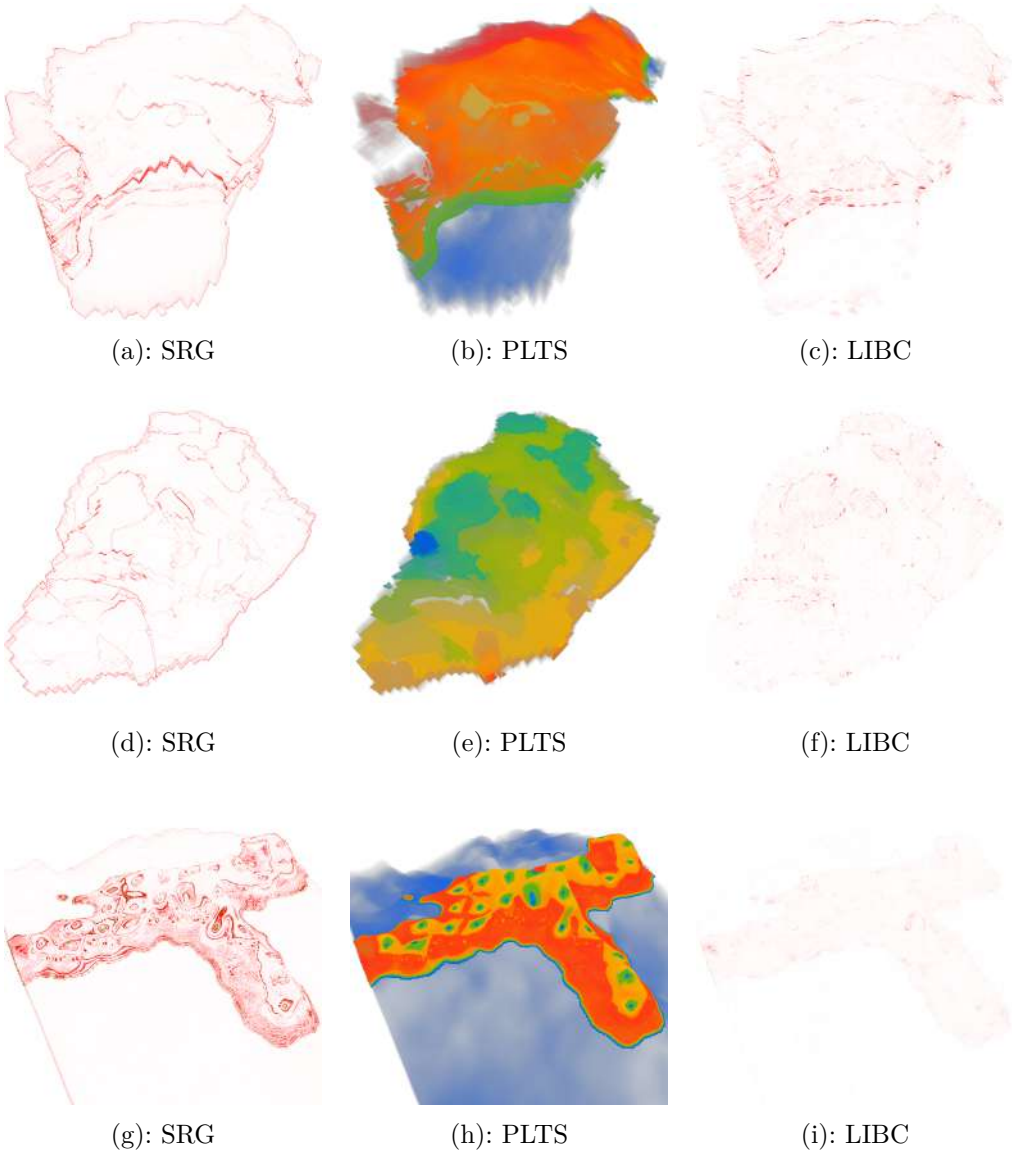
Figure 4.10: Performance and quality comparison for R2, R3, and R5 reservoir using a viewport of $1000^2$. The PLTS was used as ground truth for PSNR and SSIM computations. For all plots, higher values mean better results.

### 4.3.4 Pre-processing

The illumination technique requires an amount of additional memory equals to $2 + \frac{1}{7}$ of the extinction coefficient volume resolution, taking into account its mipmap levels and the additional volume of opacities. The opacity volume can be discarded after computing the base level of the texture mipmap. Besides that, we also must keep the pre-illumination data accessed for each sample along the viewing ray in the rendering stage. To reduce memory consumption, we use half-precision float for all these volumes, without compromising image quality.

Table 4.5 shows pre-processing times to compute the volume of representative extinction coefficients for different resolutions and reservoir models. The volume is built using the accurate point location algorithm. For higher resolutions, an interactive frame rate while modifying the transfer function is compromised.

In Figure 4.13, we made an additional experiment modifying the extinction coefficient volume resolution according to Table 4.5, for the model R3 and R5. As we decrease the extinction coefficient volume resolution, the lighting



(a): SRG (b): PLTS (c): LIBC

(d): SRG (e): PLTS (f): LIBC

(g): SRG (h): PLTS (i): LIBC

Figure 4.11: LAB color difference [56] considering the implemented strategies for R2, R3, and R5. The first two models present geometry discontinuities; the last is a large reservoir model. Note how SRG strategy suffered for delivering good approximations when the viable regular grid resolution is not sufficient to capture the scalar field variation.

effects tend to blurry; however, even with smaller resolutions, the achieved images are of good quality, requiring less memory space and pre-processing time.

Table 4.3: Table with the used parameters to visualize the reservoirs. We also consider the memory required to store the additional light volume if pre-illumination is enabled (using half-precision floats).

| | | | Pre-illumination | | Parameters | |
|---|---|---|---|---|---|---|
| Dataset | $\sigma_0$ | Ext. Coef. Vol. Res. | Vol. Size | Memory (MB) | $\theta_o$ | $C_{split}$ |
| R1 | 13.02 | $128 \times 128 \times 128$ | $64 \times 64 \times 64$ | 0.5 | 20 | 3 |
| R2 | 20.77 | $256 \times 256 \times 256$ | $128 \times 128 \times 128$ | 4.0 | 15 | 3 |
| R3 | 17.40 | $256 \times 256 \times 256$ | $128 \times 128 \times 128$ | 4.0 | 15 | 3 |
| R4 | 23.70 | $256 \times 256 \times 768$ | $128 \times 128 \times 256$ | 8.0 | 25 | 7 |
| R5 | 35.39 | $768 \times 768 \times 768$ | $400 \times 400 \times 400$ | 122.1 | 20 | 7 |

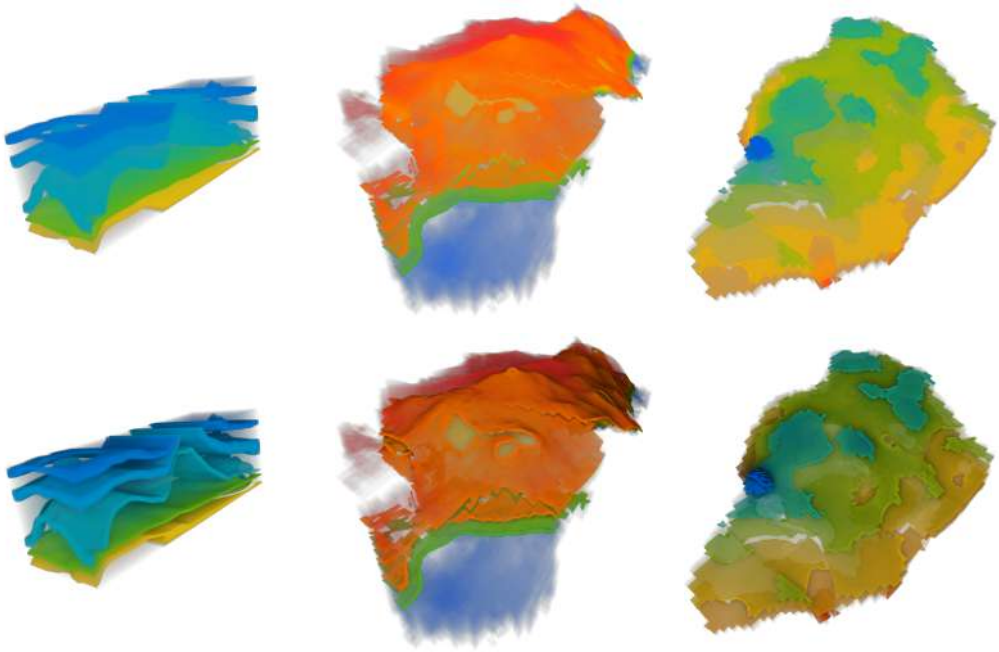
Table 4.4: Performance results with the defined parameters in Table 4.3 using a viewport of $1000^2$, first computing Emission and Absorption only and then adding directional ambient occlusion. We can see how pre-illumination greatly increases performance for directional ambient occlusion computation.

| | Frame Rate (*fps*) | | |
|---|---|---|---|
| Dataset | EA | DAO | DAO w/ Pre-illumination |
| R1 | 426 | 65 | 334 |
| R2 | 91 | 10 | 70 |
| R3 | 105 | 12 | 80 |
| R4 | 85 | 3 | 54 |
| R5 | 17 | $< 1$ | 7 |

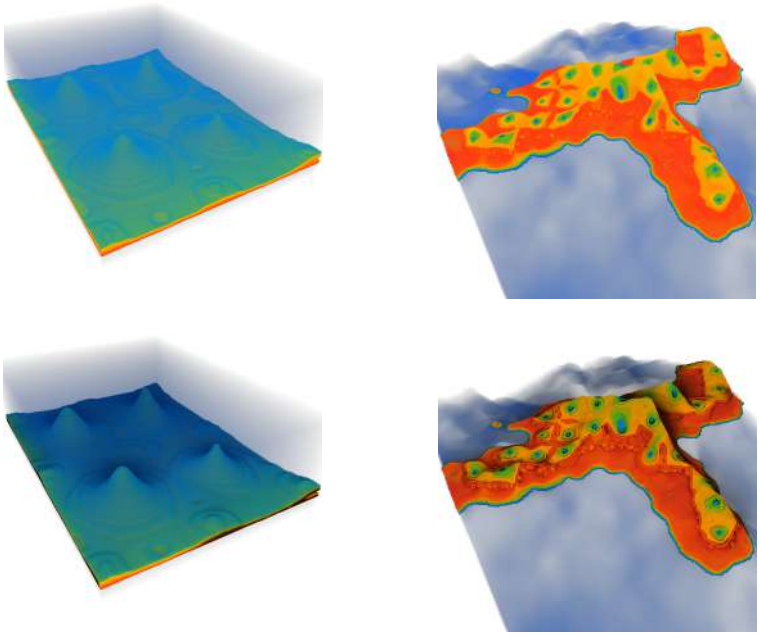


Table 4.5: Pre-processing times to generate the Extinction Coefficient Volume for different reservoir models, considering $2 + \frac{1}{7}$ of memory consumption based on the volume resolution, using half-precision floats.

| Reservoir Resolution | $\sigma_0$ | Ext. Coef. Vol. Resolution | Time (*ms*) | Memory (MB) |
|---|---|---|---|---|
| $46 \times 69 \times 30$ | 69.62 | $64^3$ | 12 | 1.1 |
| | 34.82 | $128^3$ | 38 | 8.6 |
| | 17.40 | $256^3$ | 190 | 68.6 |
| | 8.71 | $512^3$ | 1164 | 548.6 |
| $346 \times 352 \times 100$ | 70.77 | $384^3$ | 2097 | 231.5 |
| | 53.08 | $512^3$ | 2704 | 548.6 |
| | 35.39 | $768^3$ | 15802 | 1,860.5 |
| | 26.54 | $1024^3$ | 17157 | 4,388.6 |

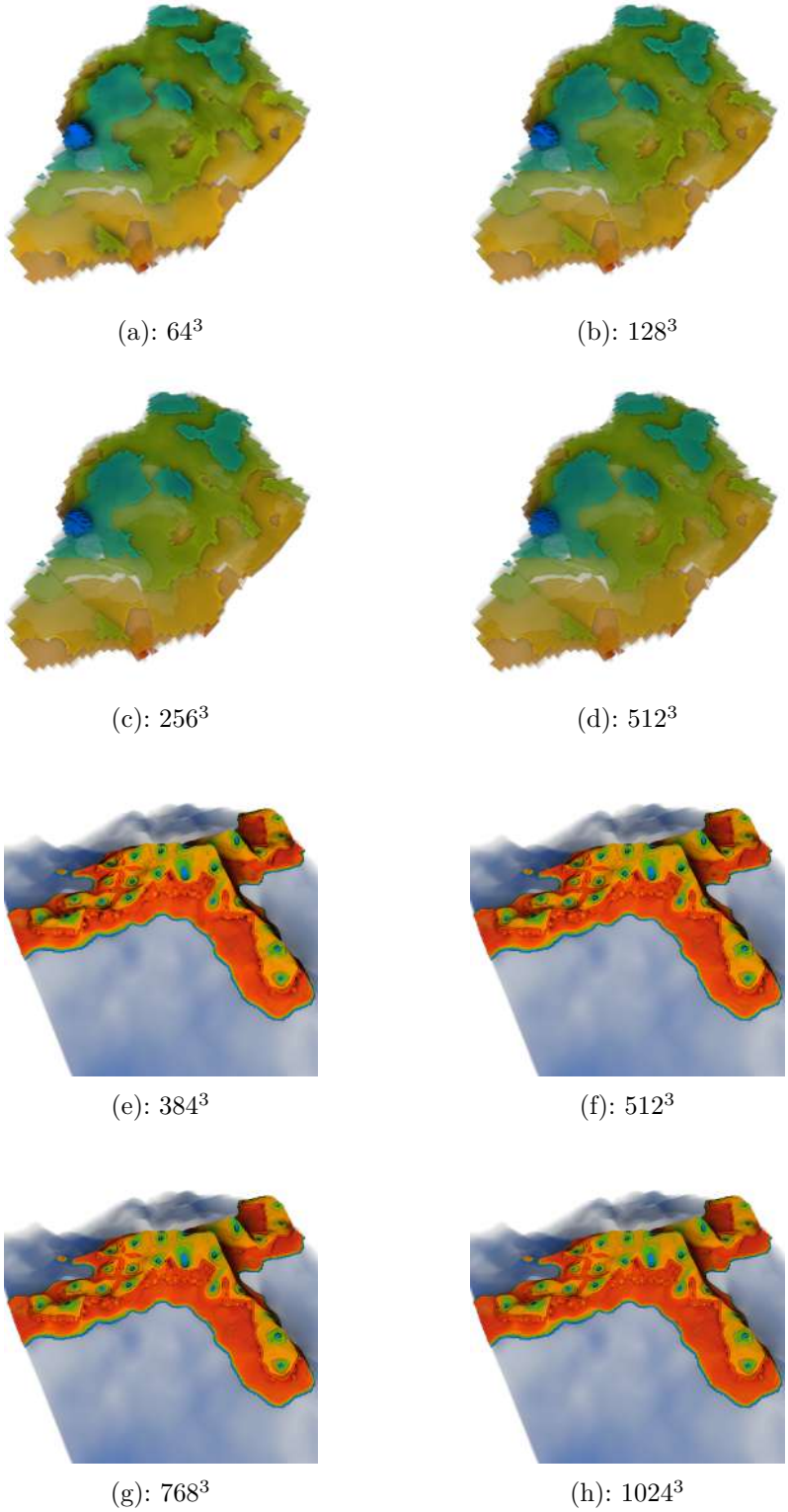


(a): R1 (b): R2 (c): R3



(d): R4 (e): R5

Figure 4.12: Achieved results with only Emission and Absorption illumination model, and then combining directional ambient occlusion. Note how ambient occlusion helps reveal isosurfaces and regions placed in similar screen positions, but at different depths.



(a): $64^3$ (b): $128^3$

(c): $256^3$ (d): $512^3$

(e): $384^3$ (f): $512^3$

(g): $768^3$ (h): $1024^3$

Figure 4.13: R3 and R5 datasets rendered with directional ambient occlusion using different extinction coefficient volume resolutions. We are still able to generate interesting illumination effects even using lower resolutions, which considerably decreases the memory requirements and pre-processing time, but might degrade the illumination quality. For a fair comparison, these images were rendered without using pre-illumination.

# 5
# Conclusions and future work

In this thesis, we proposed a new efficient method to compute transparency within a cone using Gaussian integrals for structured datasets. As a result, we were able to implement an interactive ray casting algorithm with two illumination effects: directional ambient occlusion and shadows. The proposed method delivers competitive image quality and performance. We also have discussed the main limitation of the method: the darkness of oblique opaque region and the lightness of semi-transparency regions with constant opacity values.

The proposed illumination method allows variation to achieve better performance, such as to precompute a transparency volume or to reduce the dimension of the extinction coefficient volumes. To the best of our knowledge, this is the first proposal that supports directional ambient occlusion and shadow computations under the same framework, while delivering a better balance between image quality and performance, when compared to previous specialized solutions.

We also made an experimental study on three different strategies to visualize black oil reservoir models. Such models are represented by irregular hexahedral meshes laying on a topological grid, with geometry discontinuities. All the rendering strategies were all implemented on GPU, using the representation proposed in [21]. The goal was to present different strategies to evaluate samples along the ray, comparing delivered performance and image quality.

The first strategy resampled the reservoir model in a regular grid; the second linearized the geometry of the cells and the variation of the scalar field along the ray within each cell; the third used an accurate point location algorithm to locate each sample in the reservoir model, being accelerated by a topological search. A set of computational experiments have demonstrated how the performance varies with the first strategy, depending on the size of the resampled regular grid. The experiments also revealed that the linearization strategy presents a balance between performance and quality. The more accurate strategy suffered for delivering competitive performance. Based on these experiments, we argue that the linearization strategy is a better choice for implementing interactive applications, a statement not so evident before

the conducted experiments.

We also show how directional ambient occlusion enhanced depth and shape perception on black oil reservoirs by extending our proposed illumination technique, resampling the reservoir to an opacity volume. In this case, pre-illumination was applied to achieve interactive frame rates.

## 5.1 Future Work

For future work, we plan to investigate the proposed illumination technique for large volumes, implying the use of out-of-core techniques with distributed algorithms. Volume ray casting seems to be the best approach to this kind of problem since rays are handled independently. Shih et al. [58] already showed a proposal for employing multiple GPU's. In our case, we must investigate how our preprocessing stage and evaluation could be distributed.

For black oil reservoir models, we plan to extend our study to multi-resolution representations to reach more accurate results in realtime. It would also be interesting to do additional experiments for time-varying data and hybrid visualizations [45, 6], combining structured and unstructured volume rendering techniques.

# Bibliography


[1] AMENT, M.; DACHSBACHER, C.. **Anisotropic ambient volume shading**. IEEE Transactions on Visualization and Computer Graphics, 22(1):1015–1024, 2016.

[2] AMENT, M.; SADLO, F.; DACHSBACHER, C. ; WEISKOPF, D.. **Low-pass filtered volumetric shadows**. IEEE Transactions on Visualization and Computer Graphics, 20(12):2437–2446, Dec 2014.

[3] AMENT, M.; SADLO, F. ; WEISKOPF, D.. **Ambient volume scattering**. IEEE Transactions on Visualization and Computer Graphics, 19(12):2936–2945, Dec. 2013.

[4] BERNARDON, F. F.; PAGOT, C. A.; COMBA, J. L. D. ; SILVA, C. T.. **Gpu-based tiled ray casting using depth peeling**. Journal of Graphics Tools, 11(4):1–16, 2006.

[5] BEYER, J.; HADWIGER, M.; MÖLLER, T. ; FRITZ, L.. **Smooth mixed-resolution gpu volume rendering**. In: PROCEEDINGS OF THE FIFTH EUROGRAPHICS / IEEE VGTC CONFERENCE ON POINT-BASED GRAPHICS, SPBG'08, p. 163–170, Goslar, DEU, 2008. Eurographics Association.

[6] BINYAHIB, R.; PETERKA, T.; LARSEN, M.; MA, K. ; CHILDS, H.. **A scalable hybrid scheme for ray-casting of unstructured volume data**. IEEE Transactions on Visualization and Computer Graphics, 25(7):2349–2361, July 2019.

[7] BLINN, J. F.. **Models of light reflection for computer synthesized pictures**. In: PROCEEDINGS OF THE 4TH ANNUAL CONFERENCE ON COMPUTER GRAPHICS AND INTERACTIVE TECHNIQUES, SIGGRAPH '77, p. 192–198, New York, NY, USA, 1977. ACM.

[8] CALLAHAN, S. P.; BAVOIL, L.; PASCUCCI, V. ; SILVA, C. T.. **Progressive volume rendering of large unstructured grids**. IEEE Transactions on Visualization and Computer Graphics, 12(5):1307–1314, 2006.

[9] CAMPAGNOLO, L. Q.; CELES, W.. **Interactive directional ambient occlusion and shadow computations for volume ray casting**. Computers & Graphics, 84:66 – 76, 2019.

[10] CAMPAGNOLO, L. Q.; CELES, W.. **An experimental study on volumetric visualization of black oil reservoir models**. Computers & Graphics, 93:84 – 94, 2020.

[11] CARR, H.; MOLLER, T. ; SNOEYINK, J.. **Artifacts caused by simplicial subdivision**. IEEE Transactions on Visualization and Computer Graphics, 12(2):231–242, 2006.

[12] CORREA, C. D.; HERO, R. ; MA, K.. **A comparison of gradient estimation methods for volume rendering on unstructured meshes**. IEEE Transactions on Visualization and Computer Graphics, 17(3):305–319, March 2011.

[13] CRASSIN, C.; NEYRET, F.; SAINZ, M.; GREEN, S. ; EISEMANN, E.. **Interactive indirect illumination using voxel cone tracing**. Computer Graphics Forum (Proceedings of Pacific Graphics 2011), 30(7), sep 2011.

[14] CROW, F. C.. **Summed-area tables for texture mapping**. In: PROCEEDINGS OF THE 11TH ANNUAL CONFERENCE ON COMPUTER GRAPHICS AND INTERACTIVE TECHNIQUES, SIGGRAPH '84, p. 207–212, New York, NY, USA, 1984. ACM.

[15] DÍAZ, J.; VÁZQUEZ, P.-P.; NAVAZO, I. ; DUGUET, F.. **Real-time ambient occlusion and halos with summed area tables**. Comput. Graph., 34(4):337–350, Aug. 2010.

[16] DÍAZ, J.; ROPINSKI, T.; NAVAZO, I.; GOBBETTI, E. ; VÁZQUEZ, P.-P.. **An experimental study on the effects of shading in 3d perception of volumetric models**. Vis. Comput., 33(1):47–61, Jan. 2017.

[17] ENGEL, K.; HADWIGER, M.; KNISS, J.; REZK-SALAMA, C. ; WEISKOPF, D.. **Real-Time Volume Graphics**. Ak Peters Series. Taylor & Francis, 2006.

[18] ENGEL, D.; ROPINSKI, T.. **Deep volumetric ambient occlusion**. IEEE Transactions on Visualization and Computer Graphics, 27(2):1268–1278, 2021.

[19] ESPINHA, R.; CELES, W.. **High-quality hardware-based ray-casting volume rendering using partial pre-integration**. In: XVIII BRAZILIAN SYMPOSIUM ON COMPUTER GRAPHICS AND IMAGE PROCESSING (SIBGRAPI'05), p. 273–280, Oct 2005.

[20] FAVERA, E. C. D.; CELES, W.. **Ambient occlusion using cone tracing with scene voxelization**. In: 2012 25TH SIBGRAPI CONFERENCE ON GRAPHICS, PATTERNS AND IMAGES, p. 142–149, Aug 2012.

[21] FRANCESCHIN, B.; ABRAHAM, F.; NETTO, L. F. ; CELES, W.. **Gpu-based rendering of arbitrarily complex cutting surfaces for black oil reservoir models**. In: 32ND SIBGRAPI CONFERENCE ON GRAPHICS, PATTERNS AND IMAGES (SIBGRAPI), p. 131–138, 2019.

[22] FRIEDMAN, E.. **Circles in circles on erich's packing center**. `https://www2.stetson.edu/~efriedma/cirincir/`, 2005. Online; accessed April 2019.

[23] GARRITY, M. P.. **Raytracing irregular volume data**. SIGGRAPH Comput. Graph., 24(5):35–40, Nov. 1990.

[24] HERNELL, F.; LJUNG, P. ; YNNERMAN, A.. **Local ambient occlusion in direct volume rendering**. IEEE Transactions on Visualization and Computer Graphics, 16(4):548–559, July 2010.

[25] IGLESIAS-GUITIAN, J. A.; MANE, P. S. ; MOON, B.. **Real-time denoising of volumetric path tracing for direct volume rendering**. IEEE Transactions on Visualization and Computer Graphics, p. 1–1, 2020.

[26] JÖNSSON, D.; SUNDÉN, E.; YNNERMAN, A. ; ROPINSKI, T.. **A Survey of Volumetric Illumination Techniques for Interactive Volume Rendering**. Computer Graphics Forum, 2014.

[27] JÖNSSON, D.; YNNERMAN, A.. **Correlated photon mapping for interactive global illumination of time-varying volumetric data**. IEEE Transactions on Visualization and Computer Graphics, 23(1):901–910, 2017.

[28] KAUFMAN, A.; MUELLER, K.. **Overview of volume rendering**. In: Hansen, C. D.; Johnson, C. R., editors, VISUALIZATION HANDBOOK, p. 127 – 174. Butterworth-Heinemann, Burlington, 2005.

[29] KHLEBNIKOV, R.; VOGLREITER, P.; STEINBERGER, M.; KAINZ, B. ; SCHMALSTIEG, D.. **Parallel irradiance caching for interactive**

**monte-carlo direct volume rendering**. Computer Graphics Forum, 33, 06 2014.

[30] KRAFT, V.; LINK, F.; SCHENK, A. ; SCHUMANN, C.. **Adaptive illumination sampling for direct volume rendering**. In: CGI, 2020.

[31] KRUGER, J.; WESTERMANN, R.. **Acceleration techniques for gpu-based volume rendering**. In: PROCEEDINGS OF THE 14TH IEEE VISUALIZATION 2003 (VIS'03), VIS '03, p. 38–, Washington, DC, USA, 2003. IEEE Computer Society.

[32] LEVEN, J.; CORSO, J.; COHEN, J. ; KUMAR, S.. **Interactive visualization of unstructured grids using hierarchical 3d textures**. In: SYMPOSIUM ON VOLUME VISUALIZATION AND GRAPHICS, 2002. PROCEEDINGS. IEEE / ACM SIGGRAPH, p. 37–44, Oct 2002.

[33] LEVOY, M.. **Display of surfaces from volume data**. IEEE Computer Graphics and Applications, 8(3):29–37, May 1988.

[34] LINDEMANN, F.; ROPINSKI, T.. **About the influence of illumination models on image comprehension in direct volume rendering**. IEEE Transactions on Visualization and Computer Graphics, 17(12):1922–1931, Dec 2011.

[35] MAGNUS, J. G.; BRUCKNER, S.. **Interactive dynamic volume illumination with refraction and caustics**. IEEE Transactions on Visualization and Computer Graphics, 24(1):984–993, Jan 2018.

[36] MARTSCHINKE, J.; HARTNAGEL, S.; KEINERT, B.; ENGEL, K. ; STAMMINGER, M.. **Adaptive temporal sampling for volumetric path tracing of medical data**. Computer Graphics Forum, 38, 2019.

[37] MAX, N.. **Optical models for direct volume rendering**. IEEE Transactions on Visualization and Computer Graphics, 1(2):99–108, Jun 1995.

[38] MAX, N. L.; CHEN, M.. **Local and global illumination in the volume rendering integral**. In: SCIENTIFIC VISUALIZATION: ADVANCED CONCEPTS, p. 259–274, 2010.

[39] MCREYNOLDS, T.; BLYTHE, D.. **Chapter 20 - scientific visualization**. In: McREYNOLDS, T.; BLYTHE, D., editors, ADVANCED GRAPHICS PROGRAMMING USING OPENGL, The Morgan Kaufmann Series in Computer Graphics, p. 531–570. Morgan Kaufmann, San Francisco, 2005.

[40] MESQUITA, R.; CELES, W.. **Robust and effective method for automatic generation of one-dimensional transfer functions**. In: 2019 32ND SIBGRAPI CONFERENCE ON GRAPHICS, PATTERNS AND IMAGES (SIBGRAPI), p. 92–99, 2019.

[41] MIRANDA, F. M.; CELES, W.. **Volume rendering of unstructured hexahedral meshes**. The Visual Computer, 28(10):1005–1014, Oct 2012.

[42] MISAKI, Y.; INO, F. ; HAGIHARA, K.. **Cache-aware, in-place rotation method for texture-based volume rendering**. IEICE Transactions, 100-D:452–461, 2017.

[43] MÖLLER, T.; TRUMBORE, B.. **Fast, minimum storage ray-triangle intersection**. J. Graph. Tools, 2(1):21–28, Oct. 1997.

[44] MORELAND, K.; ANGEL, E.. **A fast high accuracy volume renderer for unstructured data**. In: 2004 IEEE SYMPOSIUM ON VOLUME VISUALIZATION AND GRAPHICS, p. 9–16, 2004.

[45] MUIGG, P.; HADWIGER, M.; DOLEISCH, H. ; HAUSER, H.. **Scalable hybrid unstructured and structured grid raycasting**. IEEE Transactions on Visualization and Computer Graphics, 13(6):1592–1599, Nov 2007.

[46] MUIGG, P.; HADWIGER, M.; DOLEISCH, H. ; GROLLER, E.. **Interactive volume visualization of general polyhedral grids**. IEEE Transactions on Visualization and Computer Graphics, 17(12):2115–2124, 2011.

[47] NEHAB, D.; MAXIMO, A.; LIMA, R. S. ; HOPPE, H.. **Gpu-efficient recursive filtering and summed-area tables**. ACM Trans. Graph., 30(6):1–12, Dec. 2011.

[48] NEHAB, D.; HOPPE, H.. **A fresh look at generalized sampling**. Foundations and Trends® in Computer Graphics and Vision, 8(1):1–84, 2014.

[49] NEHAB, D.; MAXIMO, A.. **Parallel recursive filtering of infinite input extensions**. ACM Trans. Graph., 35(6), Nov. 2016.

[50] PAPAIOANNOU, G.; MENEXI, M. L. ; PAPADOPOULOS, C.. **Real-time volume-based ambient occlusion**. IEEE Transactions on Visualization and Computer Graphics, 16(5):752–762, Sep. 2010.

[51] ROPINSKI, T.; MEYER-SPRADOW, J.; DIEPENBROCK, S.; MENSMANN, J. ; HINRICHS, K.. **Interactive Volume Rendering with Dynamic Ambient Occlusion and Color Bleeding**. Computer Graphics Forum, 2008.

[52] SANS, F.; CARMONA, R.. **A comparison between gpu-based volume ray casting implementations: Fragment shader, compute shader, opencl, and cuda**. CLEI Electronic Journal, 20(2), 2017.

[53] SATTER, A.; IQBAL, G. M. ; BUCHWALTER, J. L.. **Practical Enhanced Reservoir Engineering: Assisted with Simulation Software**. PennWell Books, 2008.

[54] SCHLEGEL, P.; MAKHINYA, M. ; PAJAROLA, R.. **Extinction-based shading and illumination in gpu volume ray-casting**. IEEE Transactions on Visualization and Computer Graphics, 17(12):1795–1802, Dec 2011.

[55] SCHOTT, M.; PEGORARO, V.; HANSEN, C. D.; BOULANGER, K. ; BOUATOUCH, K.. **A directional occlusion shading model for interactive direct volume rendering**. Comput. Graph. Forum, 28:855–862, 2009.

[56] SHARMA, G.; WU, W. ; DALAL, E. N.. **The ciede2000 color-difference formula: Implementation notes, supplementary test data, and mathematical observations**. Color Research & Application, 30(1):21–30, 2005.

[57] SHIH, M.; ZHANG, Y. ; MA, K.-L.. **Advanced lighting for unstructured-grid data visualization**. In: 2015 IEEE PACIFIC VISUALIZATION SYMPOSIUM (PACIFICVIS), p. 239–246, April 2015.

[58] SHIH, M.; RIZZI, S.; INSLEY, J.; URAM, T.; VISHWANATH, V.; HERELD, M.; PAPKA, M. E. ; MA, K. L.. **Parallel distributed, gpu-accelerated, advanced lighting calculations for large-scale volume visualization**. In: 2016 IEEE 6TH SYMPOSIUM ON LARGE DATA ANALYSIS AND VISUALIZATION (LDAV), p. 47–55, Oct 2016.

[59] SHIRLEY, P.; TUCHMAN, A.. **A polygonal approximation to direct scalar volume rendering**. SIGGRAPH Comput. Graph., 24(5):63–70, Nov. 1990.

[60] SILVA, C. T.; COMBA, J. L. D.; CALLAHAN, S. P. ; BERNARDON, F. F.. **A survey of gpu-based volume rendering of unstructured grids**. Revista de Informática Teórica e Aplicada, 12(2):9–29, 2005.

[61] ŠOLTÉSZOVÁ, V.; PATEL, D.; BRUCKNER, S. ; VIOLA, I.. **A multidirectional occlusion shading model for direct volume rendering**. Computer Graphics Forum, 29(3):883–891, 2010.

[62] SUGIMOTO, Y.; INO, F. ; HAGIHARA, K.. **Improving cache locality for gpu-based volume rendering**. Parallel Comput., 40:59–69, 2014.

[63] 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, Jan 2010.

[64] WANG, J.; YANG, F. ; CAO, Y.. **Cache-aware sampling strategies for texture-based ray casting on gpu**. In: 2014 IEEE 4TH SYMPOSIUM ON LARGE DATA ANALYSIS AND VISUALIZATION (LDAV), p. 19–26, Nov 2014.

[65] WANG, Z.; BOVIK, A. C.; SHEIKH, H. R. ; SIMONCELLI, E. P.. **Image quality assessment: from error visibility to structural similarity**. IEEE Transactions on Image Processing, 13(4):600–612, 2004.

[66] WEILER, M.; KRAUS, M.; MERZ, M. ; ERTL, T.. **Hardware-based ray casting for tetrahedral meshes**. In: IEEE VISUALIZATION, 2003. VIS 2003., p. 333–340, Oct 2003.

[67] WEILER, M.; MALLON, P. N.; KRAUS, M. ; ERTL, T.. **Texture-encoded tetrahedral strips**. In: 2004 IEEE SYMPOSIUM ON VOLUME VISUALIZATION AND GRAPHICS, p. 71–78, Oct 2004.

[68] WEILER, M.; WESTERMANN, R.; HANSEN, C.; ZIMMERMANN, K. ; ERTL, T.. **Level-of-detail volume rendering via 3d textures**. In: PROCEEDINGS OF THE 2000 IEEE SYMPOSIUM ON VOLUME VISUALIZATION, VVS '00, p. 7–13, New York, NY, USA, 2000. Association for Computing Machinery.

[69] WILHELMS, J.; VAN GELDER, A.. **A coherent projection approach for direct volume rendering**. In: PROCEEDINGS OF THE 18TH ANNUAL CONFERENCE ON COMPUTER GRAPHICS AND INTERACTIVE TECHNIQUES, SIGGRAPH '91, p. 275–284, New York, NY, USA, 1991. ACM.

[70] ZHANG, Y.; MA, K.-L.. **Fast global illumination for interactive volume visualization**. In: PROCEEDINGS OF THE ACM SIGGRAPH SYMPOSIUM ON INTERACTIVE 3D GRAPHICS AND GAMES, I3D '13, p. 55–62, New York, NY, USA, 2013. Association for Computing Machinery.

[71] ZHOU, Y.; GARLAND, M.. **Interactive point-based rendering of higher-order tetrahedral data**. IEEE Transactions on Visualization and Computer Graphics, 12:1229–1236, 2006.

# A
# Implementation Details of Transparency Computation

Our technique requires a preprocessing stage to compute the current volume of extinction coefficients and additional data to place samples along the transparency cone.

The volume of extinction coefficients is a 3D texture, storing the representative amplitudes of Gaussian distributions in half-precision floats, by evaluating a sequence of Gaussian filters. We only calculate the first level of our mipmapped 3D texture directly from the original volume. The upper levels are computed using the amplitude from previous level.

We create another RGBA 1D texture to evaluate the samples along each transparency cone. We create one texture for directional ambient occlusion and one for each light source for shadow generation. It contains for each sample $S_i$:

- $d_s$: the distance to next sample. As we mentioned in Chapter 3.4, it is $h = 2.5\sigma$, when both samples are placed with same $\sigma$. When $\sigma[i] \neq \sigma[i+1]$, the distance to next sample is calculated by $d_s[i] = 1.25\,\sigma[i] + 1.25\,\sigma[i+1]$.
- $mm_{level}$: the current mipmap level, calculated by $mm_{level}[i] = \log_2\left(\frac{\sigma[i]}{\sigma_0}\right)$.
- $A_\tau$: current representative value to limit the Gaussian distribution inside a cone domain (as defined in Chapter 3.2). It is calculated by $A_\tau[i] = \left(\frac{p_r[i]^2\,(\sigma[i]\sqrt{2\pi})^2}{\pi\, r[i]^2}\right)$.
- $h_I$: the current integral used in the numerical approximation defined by Equation 3-16 in Section 3.2. The first integral is half of a 1D Gaussian Integral, $h_I[0] = 0.5\,\sigma[i]\sqrt{2\pi}$, and the rest is integrated by the trapezoidal rule, $h_I[i] = \frac{1.25\sigma[i-1] + 1.25\sigma[i]}{2}$.

Figure A.1 shows an illustration of samples positioned along the transparency cone. After computing the 1D texture data, Equation 3-16 is approximated by:

$$\iiint_{\Psi} \tau(u,v,w)dudvdw \approx h_I[0]\,\tau'_{s[0]} + \sum_{i=1}^{n} h_I[i]\left(\tau'_{s[i-1]} + \tau'_{s[i]}\right) \qquad \text{(A-1)}$$

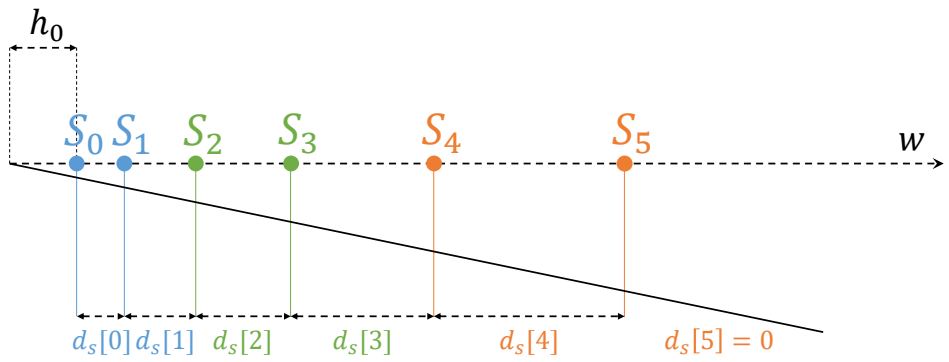



Figure A.1: Example of consecutive samples evaluated along a traced cone.

where $\tau'_{s[i]}$ is calculated using $d_s$ to define the distance of each sample along the cone ray, $mm_{level}$ to get the representative value of a Gaussian distribution in the volume of extinction coefficients, and $A_\tau$ to limit it inside the cone domain.

Our algorithm to render structured datasets is based on single pass volume ray casting. We first compute the rays from camera to intersect the Volume AABB. Then, we compute the ray entry and exit point to place samples along this interval. We have one additional 3D texture composed by normalized scalar values, representing the volume, and a 1D texture, representing the transfer function that maps scalar values to color.

To compute the transparency of each traced cone, we must know how to retrieve the representative extinction value of each Gaussian distribution $\tau_s$. In Algorhtm 1, we receive the position and mipmap level of the current sample and access the volume of extinction coefficients to retrieve the current representative value. If the sample is outside the volume, we attenuate the closest value inside the texture using a Gaussian filter evaluation.

**Algorithm 1:** Get Representative Gaussian ($\tau_s$).

**Data:** sample position $s_p$; mipmap level $mm_{level}$.
**Result:** Rpresentative value $\tau_s$.

$\tau_s = GetFromVolumeOfExtCoefficients(s_p, mm_{level})$
// Decrease extinction value based on the distance to border.
**if** $s_p$ *is outside the structured volume* **then**
    $\sigma = 2^{mm_{level}}$
    $dist = DistanceToVolumeBorder(s_p)$
    $\tau_s = \tau_s \, exp^{-dist/(2.0\,\sigma^2)}$
*return* $\tau_s$

Then, we use our precomputed data to compute the current transparency for each cone. The Algorithms 2, 3 and 4 shows how to evaluate 1, 3 and 7

**Algorithm 2:** Cone ray sample ($C_{split} = 1$)

**Data:** Sample position $s_p$; Orthogonal axis $u$, $v$, $w$ to add samples through the cone.
**Result:** Accumulated cone transparency.

$track\_distance = h_0$
$\tau_{ray}[0] = 0.0$
$Amp_\tau[0] = 0.0$
$step_i = 0$
**while** $step_i < Number\ of\ C_{split} = 1\ samples$ **do**
    $[d_s, mm_{level}, A_\tau, h_I] = GetSectionInfo(step_i)$
    $pos = s_p + w * track\_distance$
    // Call Algorithm 1
    $\tau_s = GetRepresentativeGaussian(pos, mm_{level})$
    $\tau_{ray}[0] = \tau_{ray}[0] + (Amp_\tau[0] + \tau_s A_\tau) h_I \alpha$
    $Amp_\tau[0] = \tau_s A_\tau$
    $track\_distance = track\_distance + d_s$
    $step_i = step_i + 1$
**if** $Number\ of\ C_{split} = 3\ samples > 0$ **then**
    // Call Algorithm 3
    $return\ ConeRay3Samples(s_p, track\_distance, u, v, w)$
$return\ exp^{-\tau_{ray}[0]}$

samples per section, starting by Algorithm 2. We also precompute the number of sections for each splitting mode.

**Algorithm 3:** Cone ray sample ($C_{split} = 3$)

**Data:** Sample position $s_p$; Current $track\_distance$; Orthogonal axis $u$, $v$, $w$ to add samples through the cone.
**Result:** Accumulated cone transparency.

// Convert $C_{split} = 1$ to $C_{split} = 3$
**for** $i = [0 - 2]$ *// 3 directions* **do**
    $\tau_{ray}[i] = \tau_{ray}[0]$
    $Amp_{\tau}[i] = Amp_{\tau}[0]$
    $w_v[i] = w\, coneray_{axis3}[i].z + u\, coneray_{axis3}[i].y + v\, coneray_{axis3}[i].x$
$s_c1 = Number\ of\ C_{split} = 1\ samples$
$step_i = 0$
**while** $step_i < Number\ of\ C_{split} = 3\ samples$ **do**
    $[d_s, mm_{level}, A_{\tau}, h_I] = GetSectionInfo(s_c1 + step_i)$
    **for** $i = [0 - 2]$ *// 3 samples* **do**
        $pos = s_p + w_v[i]\, track\_distance$
        // Call Algorithm 1
        $\tau_s = GetRepresentativeGaussianValue(pos, mm_{level})$
        $\tau_{ray}[i] = \tau_{ray}[i] + (Amp_{\tau}[i] + \tau_s A_{\tau}) h_I \alpha$
        $Amp_{\tau}[i] = \tau_s A_{\tau}$
    $track\_distance = track\_distance + d_s$
    $step_i = step_i + 1$
**if** $Number\ of\ C_{split} = 7\ samples > 0$ **then**
    // Call Algorithm 4
    $return\ ConeRay7Samples(s_p, track\_distance, u, v, w)$
$return\ \frac{\sum_{i=0}^{2} exp^{-\tau_{ray}[i]}}{3}$

**Algorithm 4:** Cone ray sample ($C_{split} = 7$)

**Data:** Sample position $s_p$; Current *track_distance*; Orthogonal axis $u$, $v$, $w$ to add samples through the cone.

**Result:** Accumulated cone transparency.

// Convert $C_{split} = 3$ to $C_{split} = 7$
$\tau_{ray}[6] = \tau_{ray}[5] = \tau_{ray}[2]$
$\tau_{ray}[4] = \tau_{ray}[3] = \tau_{ray}[1]$
$\tau_{avg} = \frac{\tau_{ray}[0]+\tau_{ray}[1]+\tau_{ray}[2]}{3}$
$\tau_{ray}[2] = \tau_{ray}[1] = \tau_{ray}[0]$
$\tau_{ray}[0] = \tau_{avg}$
$Amp_{\tau}[6] = Amp_{\tau}[5] = Amp_{\tau}[2]$
$Amp_{\tau}[4] = Amp_{\tau}[3] = Amp_{\tau}[1]$
$Amp_{avg} = \frac{Amp_{\tau}[0]+Amp_{\tau}[1]+Amp_{\tau}[2]}{3}$
$Amp_{\tau}[2] = Amp_{\tau}[1] = Amp_{\tau}[0]$
$Amp_{\tau}[0] = Amp_{avg}$
**for** $i = [0 - 6]$ // *7 directions* **do**
    $w_v[i] = w\, coneray_{axis7}[i].z + u\, coneray_{axis7}[i].y + v\, coneray_{axis7}[i].x$
$s_c1 = Number\ of\ C_{split} = 1\ samples$
$s_c3 = Number\ of\ C_{split} = 3\ samples$
$step_i = 0$
**while** $step_i < Number\ of\ C_{split} = 7\ samples$ **do**
    $[d_s, mm_{level}, A_{\tau}, h_I] = GetSectionInfo(s_c1 + s_c3 + step_i)$
    **for** $i = [0 - 6]$ // *7 samples* **do**
        $pos = s_p + w_v[i]\, track\_distance$
        // Call Algorithm 1
        $\tau_s = GetRepresentativeGaussianValue(pos, mm_{level})$
        $\tau_{ray}[i] = \tau_{ray}[i] + (Amp_{\tau}[i] + \tau_s A_{\tau}) h_I \alpha$
        $Amp_{\tau}[i] = \tau_s A_{\tau}$
    $track\_distance = track\_distance + d_s$
    $step_i = step_i + 1$
*return* $\frac{exp^{-\tau_{ray}[0]} + \left(\sum_{i=1}^{6} exp^{-\tau_{ray}[i]}\, cos_{adjray}(\theta/3)\right)}{1 + 6\, cos_{adjray}(\theta/3)}$

# B
# Parameter Control

We ran a set of additional computational experiments to better analyze how parameter tunings affect the visual results achieved by our method. In this supplementary document, we describe the main aspects of these experiments. In the subsequent sections, we present and discuss the results of the tests related to the following parameters:

- Initial standard deviation: $\sigma_0$
- Initial step to avoid self-occlusion: $h_0$
- Cone apertures: $\theta_o$ and $\theta_s$
- Number of splittings: $C_{split}$

The choices $h = 2.5\sigma$ (sample separation along cone axis) and $r_c = 2.0\sigma$ (maximum circular cone section before splitting or increasing the pyramid level) are befitting with the supporting mathematical model of our method. We use $h = 2.5$ for a good approximation of the integral along the cone axis and $r_c = 2.0\sigma$ for a good representation of the sample vicinity.

The choice of ray length ($L$) has a natural interpretation. Based on experimental tests, for ambient occlusion computation, we have set $L = 0.50D_v$, with $D_v$ being the volume diagonal length; for shadows computation, we have set $L = 0.75D_v$. Naturally, smaller values discard occlusion of distant obstacles, while greater values may introduce unnecessary computations.

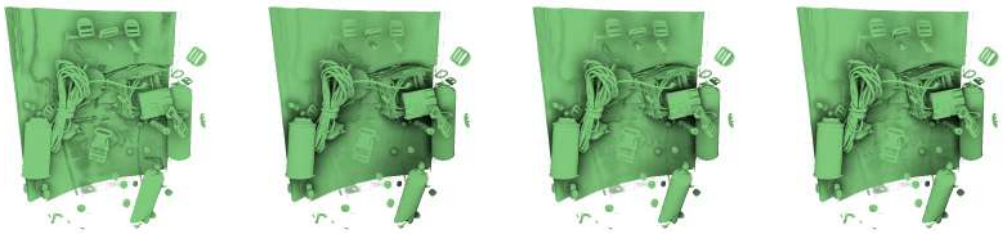
Also, the choice of the integration step for the primary rays ($h_p$) is intrinsic to solving the volume integral numerically. In our tests, we fixed the value $h_p = 0.5$ on structured datasets to achieve accurate results.

## B.1
## Initial standard deviation: $\sigma_0$

The parameter $\sigma_0$ represents the initial standard deviation of the assumed Gaussian distribution of extinction coefficients. As default, we have set $\sigma_0 = 1.0$. As stated in the paper, setting larger numbers to $\sigma_0$ would make the associated Gaussian distributions representative of larger regions, what would be a problem when sampling at small circular sections (the integral truncation

would be severe). Smaller values would require the use of more samples to cover the cone axis domain, decreasing performance.

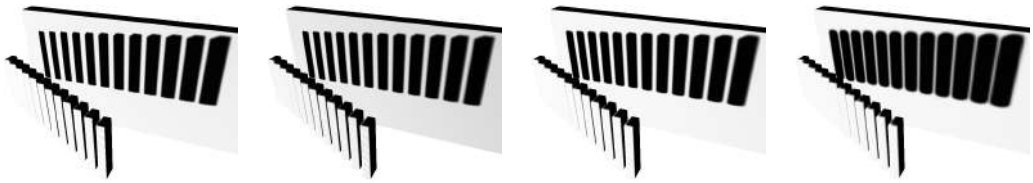
We ran two additional experiments to base our statement. In the first experiment, we try different $\sigma_0$ values for directional ambient occlusion, using the Backpack dataset. Figure B.1 illustrates the achieved results. In this case, the $\sigma_0$ variation affected the results only slightly. All three images captured the overall scene occlusion; the difference in darkness is due to the initial gap that, in this experiment, was set to $h_0 = 3.0\sigma_0$. The performance was also only slightly affected either.



(a): Monte Carlo (b): $\sigma_0 = 0.5(82fps)$ (c): $\sigma_0 = 1.0(84fps)$ (d): $\sigma_0 = 2.0(86fps)$

Figure B.1: Visual occlusion effect for Backpack dataset varying the standard deviation of the initial smoothing Gaussian kernel $\sigma_0$, using $\theta_o = 15°$.

However, let us consider the second experiment, for shadow generation, as illustrated in Figure B.2. As we need to employ cones with tiny apertures to generate accurate shadows, the effect of varying $\sigma_0$ is significant: for smaller values, we lost performance; for larger values, we lost quality.



(a): Monte Carlo (b): $\sigma_0 = 0.5(56fps)$ (c): $\sigma_0 = 1.0(83fps)$ (d): $\sigma_0 = 2.0(141fps)$

Figure B.2: Visual shadow effect for SyntheticBars dataset varying the standard deviation of the initial smoothing Gaussian kernel $\sigma_0$, using $\theta_s = 0.5°$.

## B.2
## Initial step to avoid self-occlusion: $h_0$

The choice of an adequate initial step to avoid selfocclusion is quite tricky. The adequate value depends on the scene and the transfer function in use. In volume rendering, abrupt variations of opacities are avoided. We ran all tests employing a piecewise linear 1D transfer functions to maps scalar

(a): $h_0 = 1.5$ (b): $h_0 = 3.0$ (c): $h_0 = 4.5$

Figure B.3: Bonsai dataset varying the current initial step $h_0$, using $\sigma_0 = 1.0$.

values. As a result, some amount of self-occlusion is inevitable. The tricky resides in choosing a value adequate for all models and all boundaries within a model. This parameter is required even for the method that uses Monte Carlo integration.

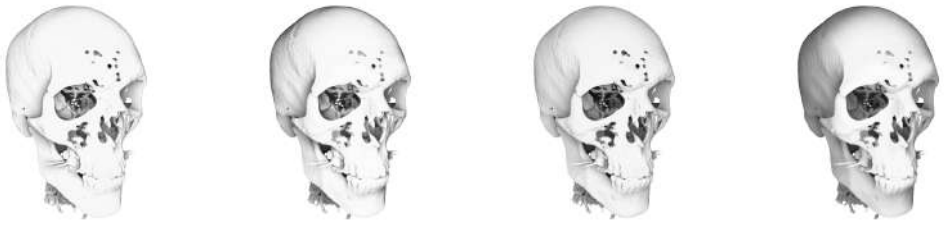
For our method, based on several computation experiments, we have set its default value to $h_0 = 3\sigma_0$. As stated in the paper, smaller values would increase self-occlusion, darkening the image; greater values would miss highfrequency occlusion near the sample. To support this argument, we ran a test with the Bonsai dataset. Figure B.3 illustrates the visual effect of varying the value of $h_0$.

## B.3
## Cone apertures: $\theta_o$ and $\theta_s$

The cone aperture controls the size of the vicinity region considered for occlusion. As stated in the paper, the proposed method works better for relatively small apertures ($\theta_o < 30°$). Large apertures would require each sample to represent a large region, decreasing accuracy. To support this argument, we ran a test with the VisMale dataset.

Figure B.4 illustrates directional ambient occlusion for using Monte Carlo integration and our method. Note that, for $\theta_o = 20°$, our method is capable to reveal occlusion details. When the aperture is increased to $\theta_o = 40°$, our method does not capture occlusion details in the inner region of the hull.

For shadow generation, as mentioned in the paper, we need to employ very small apertures ($\theta_s < 3°$) – the smaller the aperture, the harder is the shadow. We ran an experiment to demonstrate the visual effect on the generated shadows when varying the cone aperture. Figure B.5 illustrates different shadows generated using different $\theta_s$ values.



(a): MC $\theta_o = 20°$ (b): $\theta_o = 20°$ (c): MC $\theta_o = 40°$ (d): $\theta_o = 40°$

Figure B.4: Comparison betwen Monte Carlo integration and our method for VisMale dataset using different cone aperture angles $\theta_o$.

(a): $\theta_s = 0.5°$ (b): $\theta_s = 1.0°$ (c): $\theta_s = 1.5°$ (d): $\theta_s = 2.0°$

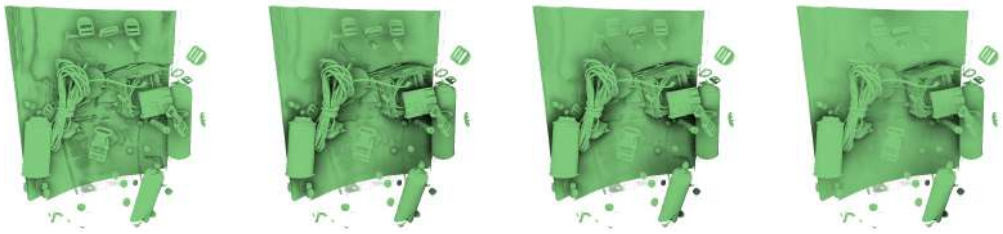
Figure B.5: Visual shadow effects for Bonsai dataset using different cone aperture angles $\theta_s$.



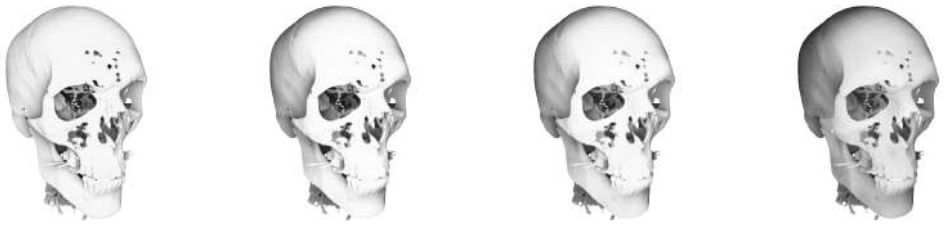
## B.4
## Number of splittings: $C_{split}$

The parameter $C_{split}$ controls the maximum number of cones per section; valid values are 1, 3, and 7. As stated in the paper, if we set $C_{split} = 1$, we only use one Gaussian per cone section, increasing performance but decreasing image quality, since upper levels of the mipmap pyramid are used very early. Setting $C_{split} = 7$ provides the best result; however, for transfer functions that result in large semi-transparent media, setting $C_{split} = 3$ improves performance with low impact in image quality. The value of $C_{split}$ is critical when sampling large cone circular sections. We ran two computational experiments to demonstrate the loss in quality as we reduce the value of Csplit. Figure B.6 illustrates the achieved results. Note that using $C_{split} = 1$ decreases image quality, when compared to the method that uses Monte Carlo integration; on the other hand, the smaller the value of $C_{split}$, the better is the achieved performance.

Lastly, we ran an experiment varying Csplit with the VisMale dataset considering a transfer function that results in a large semi-transparent region. In this case, decreasing $C_{split}$ from 7 to 3, for instance, increases performance while not compromising achieve image quality (Figure B.7).



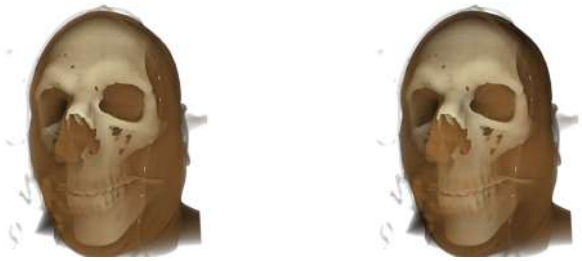
(a): Monte Carlo (b): $C_{split} = 7$ $(86fps)$ (c): $C_{split} = 3$ $(98fps)$ (d): $C_{split} = 1$ $(110fps)$



(e): Monte Carlo (f): $C_{split} = 7$ $(258fps)$ (g): $C_{split} = 3$ $(307fps)$ (h): $C_{split} = 1$ $(345fps)$

Figure B.6: Comparison between our method and Monte Carlo for Backpack and VisMale datasets varying the maximum number of cones per section $C_{split}$. Backpack dataset was rendered using $\theta_o = 15°$. For VisMale, we set $\theta_o = 30°$.

(a): Monte Carlo (b): $C_{split} = 7(49fps)$



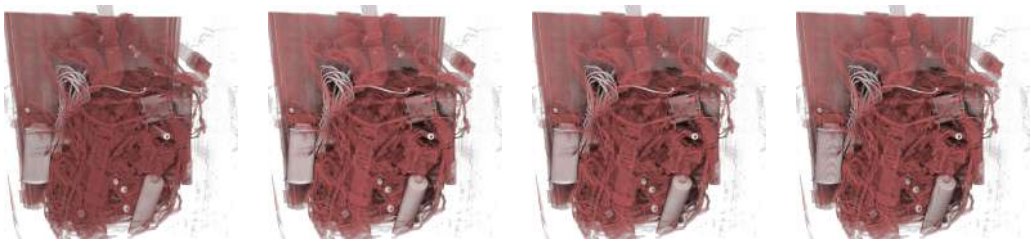
(c): $C_{split} = 3(93fps)$ (d): $C_{split} = 1(205fps)$

Figure B.7: Comparison between our method and Monte Carlo for the VisMale dataset with a large region of semitransparent voxels, varying the maximum number of cones per section $C_{split}$.

# C
# Additional Ground Truth Comparisons

We ran a set of additional comparative results between our technique and Monte Carlo for structured datasets. In this additional results, we only changed the current cone aperture angle $\theta_o$ and kept $C_{split} = 7$ for better quality.

The Table C.1 shows the results using different models in image space and object space for performance comparison. The visual results are reported in Figures C.1, C.2, C.3, C.4, C.5 and C.6. Using pre-illumination with lower resolutions increases the frame rate. However, it may generate some artifacts, specially for higher opacity variations. We tried to explore this behaviour at the second result generated with pre-illumination on each model.



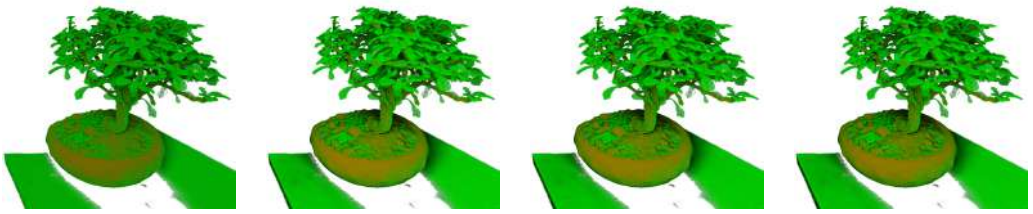
(a): Monte Carlo (b): Image (c): Object $256^3$ (d): Object $128^3$

(e): Monte Carlo (f): Image (g): Object $512^3$ (h): Object $256^3$

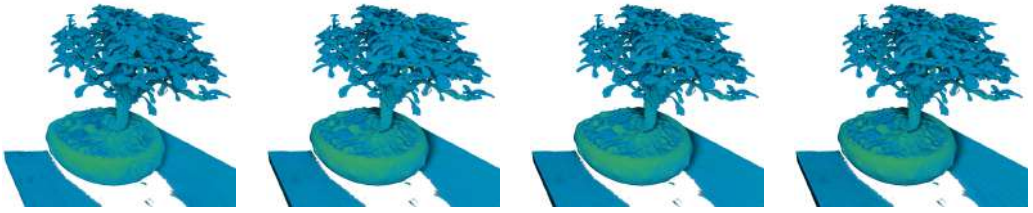
Figure C.1: Backpack dataset comparison with Monte Carlo.

Table C.1: Performance results with directional occlusion compared with Monte Carlo using viewport of $768^2$; when pre-illumination is employed, the corresponding volume resolution is annotated.

| Dataset | Ext. Coef. Vol. Res. | Pre-illumination | $\theta_o$ | Frame Rate (*fps*) EA | Frame Rate (*fps*) AO |
|---|---|---|---|---|---|
| Backpack | $512 \times 512 \times 373$ | – | 15 | 218 | 20 |
| | | $256 \times 256 \times 256$ | | | 50 |
| | | $128 \times 128 \times 128$ | | | 140 |
| | | – | 20 | 201 | 18 |
| | | $256 \times 256 \times 256$ | | | 55 |
| | | $128 \times 128 \times 128$ | | | 137 |
| Bonsai | $256 \times 256 \times 256$ | – | 20 | 438 | 113 |
| | | $256 \times 256 \times 256$ | | | 75 |
| | | $128 \times 128 \times 128$ | | | 267 |
| | | – | 10 | 456 | 123 |
| | | $256 \times 256 \times 256$ | | | 54 |
| | | $128 \times 128 \times 128$ | | | 232 |
| CT-Knee | $379 \times 229 \times 305$ | – | 10 | 730 | 42 |
| | | $200 \times 200 \times 200$ | | | 100 |
| | | $128 \times 128 \times 128$ | | | 264 |
| | | – | 15 | 629 | 135 |
| | | $200 \times 200 \times 200$ | | | 124 |
| | | $128 \times 128 \times 128$ | | | 296 |
| Foot | $256 \times 256 \times 256$ | – | 20 | 530 | 114 |
| | | $128 \times 128 \times 128$ | | | 294 |
| | | $64 \times 64 \times 64$ | | | 448 |
| | | – | 25 | 478 | 44 |
| | | $256 \times 256 \times 256$ | | | 83 |
| | | $128 \times 128 \times 128$ | | | 285 |
| Hazelnut | $512 \times 512 \times 512$ | – | 10 | 229 | 60 |
| | | $256 \times 256 \times 256$ | | | 39 |
| | | $128 \times 128 \times 128$ | | | 131 |
| | | – | 15 | 211 | 41 |
| | | $256 \times 256 \times 256$ | | | 48 |
| | | $128 \times 128 \times 128$ | | | 137 |
| Flower | $1024 \times 1024 \times 1024$ | – | 15 | 48 | 20 |
| | | $256 \times 256 \times 256$ | | | 16 |
| | | $128 \times 128 \times 128$ | | | 18 |
| | | – | 25 | 48 | 36 |
| | | $256 \times 256 \times 256$ | | | 19 |
| | | $128 \times 128 \times 128$ | | | 21 |



(a): Monte Carlo (b): Image (c): Object $256^3$ (d): Object $128^3$



(e): Monte Carlo (f): Image (g): Object $256^3$ (h): Object $128^3$

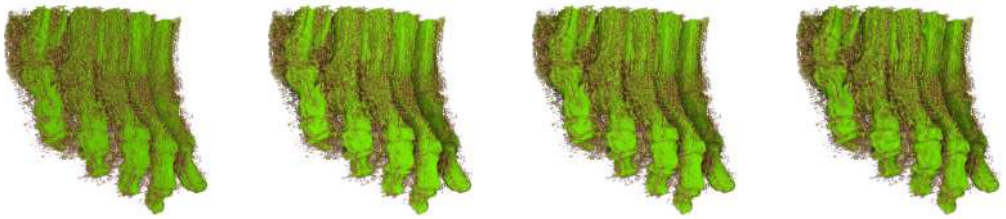
Figure C.2: Bonsai dataset comparison with Monte Carlo.

(a): Monte Carlo (b): Image (c): Object $200^3$ (d): Object $128^3$

(e): Monte Carlo (f): Image (g): Object $200^3$ (h): Object $128^3$

Figure C.3: CT-Knee dataset comparison with Monte Carlo.

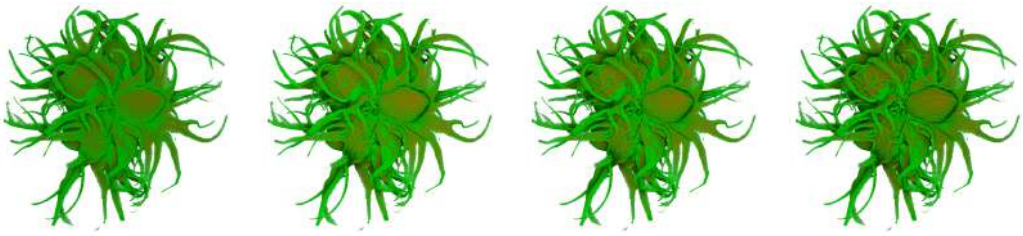




(a): Monte Carlo (b): Image (c): Object $128^3$ (d): Object $64^3$

(e): Monte Carlo (f): Image (g): Object $256^3$ (h): Object $128^3$

Figure C.4: Foot dataset comparison with Monte Carlo.



(a): Monte Carlo (b): Image (c): Object $256^3$ (d): Object $128^3$

(e): Monte Carlo (f): Image (g): Object $256^3$ (h): Object $128^3$

Figure C.5: Hazelnut dataset comparison with Monte Carlo.



(a): Monte Carlo (b): Image (c): Object $256^3$ (d): Object $128^3$

(e): Monte Carlo (f): Image (g): Object $256^3$ (h): Object $128^3$

Figure C.6: Flower dataset comparison with Monte Carlo.