Frontiers of Information Technology & Electronic Engineering
www.jzus.zju.edu.cn; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

GPU-based multi-slice per pass algorithm in interactive volume illumination rendering*

Dening LUO‡, Yi LIN, Jianwei ZHANG

College of Computer Science, Sichuan University, Chengdu 610065, China

E-mail: onexinoneyi@hotmail.com; Yilin@scu.edu.cn; zhangjianwei@scu.edu.cn

Received May 3, 2020; Revision accepted July 9, 2020; Crosschecked June 8, 2021

Abstract: Volume rendering plays a significant role in medical imaging and engineering applications. To obtain an improved three-dimensional shape perception of volumetric datasets, realistic volume illumination has been considerably studied in recent years. However, the calculation overhead associated with interactive volume rendering is unusually high, and the solvability of the problem is adversely affected when the data size and algorithm complexity are increased. In this study, a scalable and GPU-based multi-slice per pass (MSPP) volume rendering algorithm is proposed which can quickly generate global volume shadow and achieve a translucent effect based on the transfer function, so as to improve perception of the shape and depth of volumetric datasets. In our real-world data tests, MSPP significantly outperforms some complex volume shadow algorithms without losing the illumination effects, for example, half-angle slicing. Furthermore, the MSPP can be easily integrated into the parallel rendering frameworks based on sort-first or sort-last algorithms to accelerate volume rendering. In addition, its scalable slice-based volume rendering framework can be combined with several traditional volume rendering frameworks.

Key words: Volume rendering; Volume illumination; Volumetric datasets; Multi-slice per pass
https://doi.org/10.1631/FITEE.2000214 **CLC number:** TP391

1 Introduction

Medical imaging and engineering applications are the two most explored domains among the numerous visualization fields associated with volumetric datasets. The computed tomography (CT) and magnetic resonance imaging (MRI) datasets of medical imaging are intuitively visualized with a detailed three-dimensional (3D) structure. In mechanical engineering, some intermediate results associated with the data flow or structural data analysis are simulated. Therefore, the visualization of volumetric datasets has always been a major research hotspot.

Several volume rendering methods have been proposed with the development of the data visualization techniques, and some of these methods have been used in practical applications, including ray casting (Kruger and Westermann, 2003), slice-based volume rendering (Fernando, 2004), and splatting (Laur and Hanrahan, 1991). Such methods have advantages and disadvantages in case of different specific applications. Fortunately, the volume rendering methods are still being studied, optimized, and improved to achieve improved real-world applications; volume illumination addresses mainly the visual perception of depth as well as the problems associated with the spatial structure.

Generally, volume illumination requires a complicated procedure involving illumination computing and data processing. Particularly, several significant challenges are associated with interactive volume rendering when the data size and algorithm complexity increase tremendously. Currently, understanding

‡ Corresponding author

* Project supported by the Sichuan Provincial S&T Projects, China (Nos. 2020YFG0327 and 2020YFG0306) and the China Scholarship Council (No. 201806240168)

ORCID: Dening LUO, https://orcid.org/0000-0003-4359-5975

the manner in which the GPU and parallel rendering techniques can be used to resolve performance issues has become a major challenge.

To resolve the aforementioned issues, we propose a scalable and GPU-based multi-slice per pass (MSPP) volume rendering algorithm to balance the calculation performance and volume illumination of slice-based volume rendering. The specific and promising contributions are as follows:

1. MSPP can be used to build a scalable and effective slice-based volume rendering framework. MSPP is based on the traditional graphics pipeline and can be easily combined with other polygonal algorithms, thereby offering considerable interactivity without affecting the rendering quality. Furthermore, various geometry slices of the volumetric datasets associated with different applications can be managed and updated in a flexible manner.

2. The MSPP algorithm dynamically renders an arbitrary number of slices in a single pass of GPU to improve the calculation efficiency by avoiding frequent draw calls. This algorithm can ensure improved utilization of the GPU and reduce the alternation of CPU and GPU, especially in case of slice-based multi-pass applications.

3. The MSPP algorithm can effectively produce global volume shadow and translucent effects of volume illumination for the real-world volumetric datasets, obtaining a better perception of the shape and depth of volumetric datasets. Meanwhile, interactive volume illumination can be achieved in real time.

4. The scalability and flexibility of the MSPP algorithm make it easier for integration into parallel volume rendering, and the flexibility of an arbitrary number of slices per pass can help balance the performance and the effect.

2 Related work

Volume visualization has always been one of the most exciting areas in scientific visualization (Beyer et al., 2015), specifically in extracting meaningful information from volumetric datasets through the techniques of interactive graphics and imaging. Also, it is concerned with the representation, manipulation, and rendering of volumetric datasets (Çalışkan and Çevik, 2015). Meanwhile, with the development of display devices, volumetric datasets from numerous simulation and sampling devices such as MRI, CT, positron emission tomography, ultrasonic imaging, confocal microscopy, supercomputer simulations, geometric models, laser scanners, depth images estimated by stereo disparity, satellite imaging, and sonar can be efficiently visualized and demonstrated on Web (Wangkaoom et al., 2015; Mwalongo et al., 2016), mobile (Hachaj, 2014), or virtual reality (Hänel et al., 2016) platforms or devices.

Volume rendering has been developed for many years, and a large number of interactive rendering techniques have been proposed and practically applied (Jönsson et al., 2014). Direct volume rendering has become more important due to its effectiveness compared with indirect volume rendering, which is a method to reconstruct geometry from volumetric datasets (El Seoud and Mady, 2019). Therefore, different methods have been proposed to achieve direct volume rendering, such as ray casting (Kruger and Westermann, 2003; Stegmaier et al., 2005), slice-based (Bordoloi and Shen, 2005), shear-warp (Lacroute and Levoy, 1994), and splatting (Laur and Hanrahan, 1991) methods. In summary, a clear and fast representation of the 3D structures and internal details of volumetric datasets is the key task of volume visualization.

Texture-based volume rendering, which depends only on the image size instead of the scene complexity, includes two principal methods: slicing and cell projection. The 3D texture slicing volume rendering is the simplest and fastest slice-based approach on GPU (Rodríguez and Alcocer, 2012). It approximates the volume-density function by slicing the datasets in the front-to-back or back-to-front order and then blends the slices using hardware-supported blending. The view-aligned 3D texture slicing (Engel et al., 2001) is a classic method, and it takes advantage of functionalities implemented on graphics hardware like rasterizing, texturing, and blending (Fernando, 2004). Meanwhile, various composition schemes (Zhang Q et al., 2011) have been used with particular purposes, including first, average, maximum intensity projection, and accumulation.

The transfer function (TF) (Ljung et al., 2016; Ma and Entezari, 2018) for volume rendering is a central topic in scientific visualization. It is used to present more details of volumetric datasets, such as implementing volume classification (Khan et al., 2018) and volume illumination (Schlegel et al., 2011).

Perceptually motivated techniques for visualization add additional insights when displaying the volumetric datasets, because global illumination techniques can obviously improve spatial depth and shape cues and thus provide better perception (Zhang YB and Ma, 2013; Preim et al., 2016). Meanwhile, Schott et al. (2009) provided the directional occlusion approach that provides perceptual cues similar to ambient occlusion. Later approaches (Šoltészová et al., 2010; Patel et al., 2013) present soft shadows without precomputation, but do not bring considerable performance benefits (Angelelli and Bruckner, 2015).

Besides, the central topics in scientific visualization are parallel volume rendering (Beyer et al., 2015; Usher et al., 2017), remote rendering, and portable display (Noguera and Jimenez, 2016) in the future. The combination of slice-based algorithms and these central topics is also an important research hotspot in the future. Therefore, considering the better perception of volumetric datasets and the flexibility and scalability of the growing data, MSPP is proposed to satisfy the increasing practical application requirements of interactive volume rendering.

3 Algorithm

Volumetric datasets are usually a 3D array of volume elements or voxels, so volume rendering is the reconstruction process of displaying each point in a volume. Voxels can represent various physical characteristics, such as density, temperature, velocity, and pressure. Typically, the volumetric datasets store densities that are obtained using a cross-sectional imaging modality such as CT or MRI scans. A 3D texture, which is a simple array of two-dimensional (2D) textures, is obtained by accumulating these 2D slices along the slicing axis, and the densities store different material types.

For a long time, volume rendering implementations are almost exclusively based on slice-based methods where axis- or view-aligned texture slices are blended to approximate the volume rendering integral. Slice-based techniques can easily be combined with polygonal algorithms. In addition, slice-based techniques require only a few rendering passes and offer an excellent interactivity level without sacrificing the rendering quality.

First, slice-based volume rendering requires determining the slicing axis of volumetric datasets according to different application purposes and algorithm processes. Specifically, the selection of the slicing axis needs to be taken into account in the algorithm. Second, many slices are high-efficiently managed and updated in various interactive scenarios. Most importantly, the proposed MSPP will be highlighted to reduce the calculation alternation of multi-pass CPU and GPU and thus significantly improve the rendering performance. It is an effective solution to improve performance when a large number of slices are applied to complex practical volume visualization. In the working pipeline of MSPP, several steps can be efficiently implemented in interactive volume rendering methods, such as translucent effect, volume shadow, and TF applications.

Volume rendering is usually designed on the basis of the whole rendering framework, which includes the process of volumetric datasets, specific algorithm flow, and data updates. The rendering framework of MSPP is shown in Fig. 1. Slice-based volume rendering initializes volumetric datasets into 3D texture (1) and manages geometry slices (2) according to the algorithm processing. With the flexibility of the MSPP rendering framework, slice-based algorithms can be easily integrated and replaced, such as 3D texture slicing (3), half-angle slicing (Kniss et al., 2002) (4), and MSPP applications (5). Moreover, an interactive volume rendering, including geometry slice update (6) and algorithm update (7), needs to

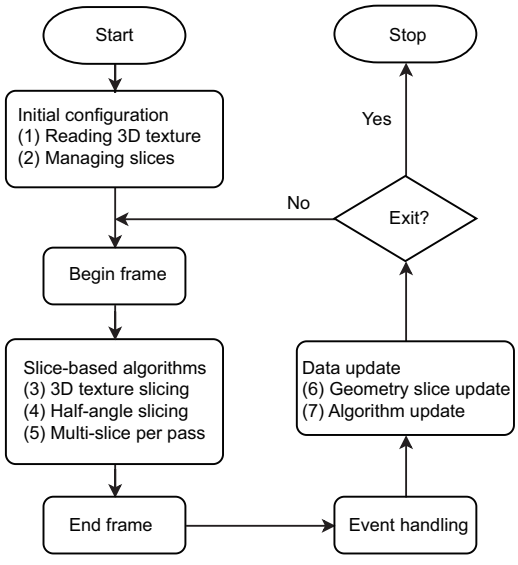

Fig. 1 The multi-slice per pass (MSPP) rendering framework. In the initial configuration, the slice-based algorithms and data updates are involved in the graphics pipeline

be designed and refined in the graphics pipeline.

3.1 Slice-based volume rendering

Slice-based volume rendering sequentially accumulates and blends all the slices generated from volumetric datasets along the slicing axis. In general, slice-based methods use a cube or a cylinder as a geometry base, dividing it into a certain number of slices on the slicing axis of the baseband, then mapping the corresponding value of the whole volumetric datasets onto each slice according to the geometry position, and finally blending all the slices sequentially with the front-to-back or back-to-front order.

The choice of slicing axis related to volume rendering methods and the volume illumination model is very crucial for slice-based volume rendering. Here, we discuss the three types of cube-based slicing axis methods (i.e., axis-aligned slicing, view-aligned slicing, and half-angle slicing), as well as the application of management of geometry slices for MSPP.

3.1.1 View-aligned 3D texture slicing

The easiest way of doing slice-based volume rendering is the axis-aligned quad slicing; that is, all the quad slices are consistently regular and are blended sequentially slice by slice (Fig. 2a). Although the cube-based slicing method is fast and straightforward, one of the major problems is that a transparent blend occurs when the viewer's direction is inconsistent with the slicing axis (Fig. 2b). In this case, the billboard technique solves this problem by forcing the quad slices always facing the viewer's direction (Fig. 2c).

View-aligned 3D texture slicing (Fig. 1, (3)) is an interactive volume rendering algorithm that can generate much better results because the slicing axis is always consistent with the viewer's direction. Although it improves the refinement of geometry, the algorithm becomes much more complicated. The slices from any viewpoint to the cube are perpendicularly sliced. Each slice obtains 1–3 triangle meshes from the nearest to farthest vertices of a cube, as shown in Fig. 3.

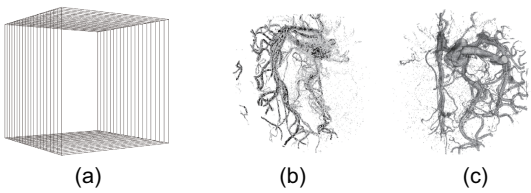

Fig. 2 Volume rendering of axis-aligned quad slices: (a) a set of regular quad slices; (b) the blend problem occurs when the viewer's direction is inconsistent with the slicing axis; (c) using the billboard technique to force the quad slices always facing the viewer's direction

Algorithm 1 shows the process of the view-aligned 3D texture slicing on CPU, in which the intersections of a unit cube and the geometry slices are found, which are perpendicular to the viewer's direction. The whole process is recomputed for each frame as the object or viewer changes (line 1). The minimum and maximum distances of unit cube vertices are calculated by doing a dot product of the viewer's direction vector with each unit cube vertex (line 2). The perpendicular plane from the nearest to the farthest vertices is used in a certain step to intersect with all the edges of the unit cube and to obtain the intersection parameters λ (line 3). The intersection points are found and triangular primitives are generated for each slice (lines 4 and 5). Finally, the management and update of geometry slices (Fig. 1, (2) and (6)) are done (line 6). Fig. 3 shows the result of mapping the 3D texture of volumetric datasets onto geometry slices.

To manage a large number of geometry slices of volumetric datasets more effectively, all the geometry slices need to be managed by a root node. Its children are organized and updated according to the different application requirements in the MSPP

Algorithm 1 Process of generating geometry slices on CPU

1: Obtain the viewer's direction and normalize it
2: Calculate the minimum and maximum distances of unit cube vertices in the viewer's direction
3: Calculate all the possible intersection parameters (λ)
4: Find the intersection points of each geometry slice
5: Generate triangular primitives
6: Update geometry slices

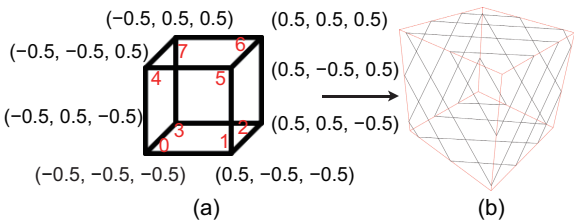

Fig. 3 The result of view-aligned 3D texture slicing: (a) meshes of geometry slices; (b) the visualization result of volumetric datasets

rendering framework. For example, view-aligned 3D texture slicing has only one geometry node that manages the vertices of all the slices, whereas half-angle slicing has child geometry nodes (having the same number of slices) that manage each slice. In contrast, each geometry node of our proposed MSPP manages the vertices of a cluster of slices according to the number of slices in one pass.

3.1.2 Half-angle slicing

The slicing axis of half-angle slicing is different from that of the view-aligned 3D texture slicing. As the name suggests, the slicing axis is the half-angle direction formed by the viewpoint and the light direction, instead of slicing the cube of volumetric datasets as perpendicular to the viewpoint direction. The advantage is that the same slices can simulate light absorption slice by slice, producing volume shadow or other volume rendering effects. Therefore, the same set of geometry slices can be rendered from the viewpoint of both the viewer and the light. Also, the slicing axis needs to be changed according to the sign of the dot of the light and the viewer's direction vectors.

The illumination attenuation of volumetric datasets is accumulated from the viewpoint of light. The half-angle slicing axis is shown in Fig. 4, in which $\boldsymbol{c}$ and $\boldsymbol{l}$ are unit vectors of the camera (viewer) and light, respectively, and $\boldsymbol{s}$ is the final slicing axis. Meanwhile, the following equation shows the calculation method of the half-angle vector, which will be used in half-angle slicing and MSPP:

$$\begin{cases} \theta = \boldsymbol{c} \cdot \boldsymbol{l}, \\ \boldsymbol{s} = \begin{cases} \boldsymbol{c} + \boldsymbol{l}, & \theta \geq 0, \\ -\boldsymbol{c} + \boldsymbol{l}, & \theta < 0. \end{cases} \end{cases} \tag{1}$$

3.2 Volume illumination

Volume illumination can present the depth perception of volumetric datasets, and it is different from the surface illumination mode. Usually, the volume illumination mode (Rostamzadeh et al., 2013) in its differential form is solved by integrating along the light direction from the starting point $s = s_0$ to the end point $s = D$ as

$$I(D) = I_0 \mathrm{e}^{-\int_{s_0}^{D} K(t)\mathrm{d}t} + \int_{s_0}^{D} q(s) \mathrm{e}^{-\int_{s}^{D} K(t)\mathrm{d}t} \mathrm{d}s, \tag{2}$$

where optical properties K and q are respectively the absorption coefficient and the source term describing emission. In most practical applications, simplified models are often used because the complete equation of light transportation is computationally intensive. The emission–absorption model, which is the most common in volume rendering, is used in this study. For the emission–absorption model, the accumulated color C and opacity A with the number of slices n are computed as follows:

$$\begin{cases} C = \sum_{i=1}^{n} C_i \prod_{j=1}^{i-1} (1 - A_j), \\ A = 1 - \prod_{j=1}^{n} (1 - A_j), \end{cases} \tag{3}$$

where C_i and A_j are the color and opacity assigned by the TF to the data value at samples i and j, respectively. Opacity A_j approximates the absorption, and opacity-weighted color C_i approximates the emission and absorption along the ray segment between samples i and $i + 1$.

For iterative computations of the discretization volume integration, the blend function is different for either the front-to-back or back-to-front order, shown respectively as

$$\begin{cases} C_{\mathrm{dst}} \leftarrow C_{\mathrm{dst}} + (1 - \alpha_{\mathrm{dst}}) C_{\mathrm{src}}, \\ \alpha_{\mathrm{dst}} \leftarrow \alpha_{\mathrm{dst}} + (1 - \alpha_{\mathrm{dst}}) \alpha_{\mathrm{src}}, \end{cases} \tag{4}$$

$$C_{\mathrm{dst}} \leftarrow (1 - \alpha_{\mathrm{src}}) C_{\mathrm{dst}} + C_{\mathrm{src}}. \tag{5}$$

Front-to-back means that rendering proceeds in the front-to-back order to the eye and back-to-front means that rendering proceeds in the back-to-front order to the eye. Variables with subscript src (as for "source") describe quantities introduced as inputs from the optical properties of the dataset (e.g., through a TF). In contrast, variables with subscript dst (as for "destination") describe output quantities that hold accumulated colors C and opacities α.

For illumination attenuation, the illumination intensity of the slice attenuates proportionally under the light source, as shown in Fig. 5. The second-layer slice is the blended illumination intensity of the first-layer slice. The volume shadow is calculated by the illumination attenuation slice by slice.

3.3 MSPP for volume shadow

A large number of passes are required for half-angle slicing for volume shadow. Thus, the draw calls

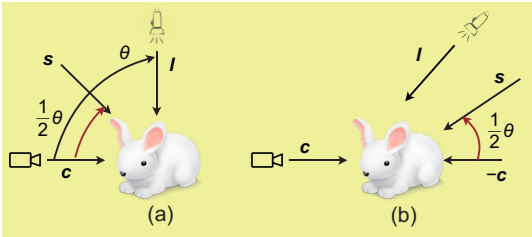

Fig. 4 The half-angle slicing axis: (a) the dot product of the viewer and light vectors is non-negative; (b) the negative direction of the viewer if the dot product is less than zero (*c*: unit vector of the camera (viewer); *l*: unit vector of the light; *s*: the final slicing axis)

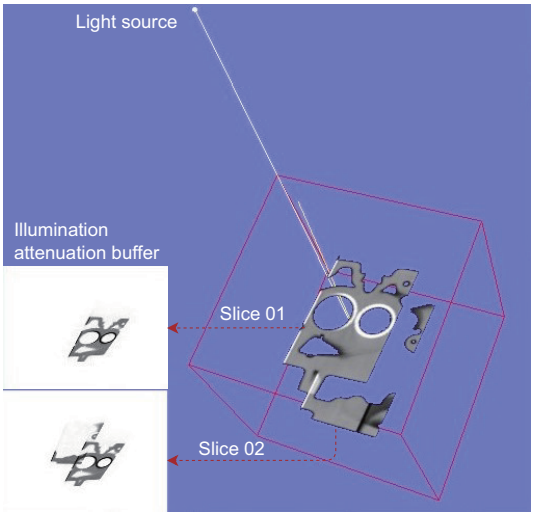

Fig. 5 Illumination attenuation slice by slice under the light source

and memory overhead are huge. The algorithm complexity is $O(n \cdot 2m)$ (n is the number of slices and m is the number of samples of each slice), given that each slice is simultaneously sampled from the viewpoint of both the viewer and the light source. The same set of slices is sampled twice, and all the passes are alternately calculated and rendered on CPU and GPU to cause frequent draw calls so that the performance will be significantly degraded.

In contrast, our proposed MSPP (Fig. 1, (5)) makes sure that multiple slices could be rendered in one pass to reduce alternate calculating and rendering on CPU and GPU. That is to say, n slices per pass can reduce the number of passes to $1/n$. Meanwhile, MSPP takes measures to calculate volume shadow without loss, and it can approximate the global illumination process quickly.

MSPP has mainly three major steps, as shown in Algorithm 2. The first step is to obtain the $\text{LightBuffer}_{0,1,\ldots,n-1}$ of n slices per pass from the viewpoint of the light source. The second step is to render each slice of n slices from the viewpoint of the viewer to calculate volume shadow according to illumination attenuation from $\text{LightBuffer}_{0,1,\ldots,n-1}$. Then n slices are blended to EyeBuffer by the rendering order according to Eq. (4) or (5). The third step accumulates the illumination attenuation of all rendered slices for the next pass to preserve the continuity of illumination attenuation slice by slice.

All geometry slices are proceeded by Algorithm 1, and the viewer's direction is the half-angle vector in Eq. (1). Meanwhile, once the direction of the viewer or the light is changed, these geometry slices are managed and efficiently updated to achieve real-time interaction.

Algorithm 2 MSPP for volume shadow on GPU

Initialize n, $\text{LightBuffer}_{0,1,\ldots,n-1}$(1,1,1,1), EyeBuffer(0,0,0,0), and AttenuationBuffer(1,1,1,1)
for int i=0; i <TotalPasses; i++ **do**
  // Step 1: render n slices per pass into
  // $\text{LightBuffer}_{0,1,\ldots,n-1}$
  Bind $\text{LightBuffer}_{0,1,\ldots,n-1}$ as the render target
  for each sample **do**
    for each slice per pass **do**
      Evaluate sample color α
      Multiply α by illumination color and output to the corresponding light buffer
    end for
  end for
  // Step 2: render n slices per pass and accumulate
  // into EyeBuffer
  Bind EyeBuffer as the render target
  Bind $\text{LightBuffer}_{0,1,\ldots,n-1}$ as texture
  Bind AttenuationBuffer as texture
  for each sample **do**
    for each slice per pass **do**
      Compute $\text{LightBuffer}_{0,1,\ldots,n-1}$ texture coordinates
      Evaluate sample color
      Read illumination intensity from $\text{LightBuffer}_{0,1,\ldots,n-1}$ and blend with illumination attenuation of the previous slices
      Multiply the color by illumination intensity
    end for
    Blend all slices and accumulate into EyeBuffer
  end for
  // Step 3: accumulate illumination attenuation of
  // all rendered slices
  Accumulate illumination attenuation slice by slice into AttenuationBuffer
end for

From the viewpoint of the light source, the conventional "over" blending (similar to the normal painting operation) in half-angle slicing is not needed. The n geometry slices are rendered simultaneously from the viewpoint of the light source into $\text{LightBuffer}_{0,1,\ldots,n-1}$, which independently store the illumination intensity of the corresponding slice. The illumination intensity is determined only by multiplying the sample color of volumetric datasets α by illumination color.

When n geometry slices are rendered from the viewpoint of the viewer, the former $\text{LightBuffer}_{0,1,\ldots,n-1}$ are used as textures to find out the illumination intensity of the corresponding slice. Similar to the shadow mapping algorithm, the shadow texture lookup coordinates $\boldsymbol{C}$ are calculated by matrix transformation to the light source space as follows:

$$\boldsymbol{C} = \boldsymbol{L}_{\mathrm{p}} \cdot \boldsymbol{L}_{\mathrm{v}} \cdot \boldsymbol{E}_{\mathrm{vi}} \cdot \boldsymbol{E}_{\mathrm{mv}} \cdot \boldsymbol{V}, \tag{6}$$

where $\boldsymbol{V}$ is the object space vertex position, $\boldsymbol{L}_{\mathrm{p}}$ and $\boldsymbol{L}_{\mathrm{v}}$ are the projection matrix and the view matrix respectively from the viewpoint of the light source, and $\boldsymbol{E}_{\mathrm{mv}}$ and $\boldsymbol{E}_{\mathrm{vi}}$ are the model-view matrix and the view inverse matrix respectively from the viewpoint of the camera. Each slice color is determined by multiplying the sample color by the illumination intensity blended with illumination attenuation of previous slices in AttenuationBuffer by the "over" blending. All slices' colors are blended and accumulated into EyeBuffer according to Eq. (4) or (5). The blending equation changes as θ (Eq. (1)) changes. If θ is non-negative, use Eq. (4) as the blend function; otherwise, use Eq. (5).

In step 3, it is easy to accumulate attenuation slice by slice into AttenuationBuffer. However, the question that arises here is how to distinguish each slice in the shader of MSPP. The easiest way is by marking the vertices of each slice without adding more vertex attributes. In the four components (x, y, z, w) of each vertex, the fourth component w can be used to mark the slice number. Unfortunately, the vertex will be normalized in the rendering pipeline, so the first three components need to be multiplied by the slice number to ensure correctness in the shader. Meanwhile, the coordinates of the vertices in the shader need to be divided by the component w to restore the correct position.

Another question is that each slice of MSPP outputs correct light attenuation messages into $\text{LightBuffer}_{0,1,\ldots,n-1}$. First, one slice per pass is determined by the slice number mark (MarkNo) of the vertices and the current pass (PassNo). Second, once a slice is calculated in the fragment shader, the current slice is computed and output to the corresponding buffer:

$$|\text{MarkNo} - n \cdot \text{PassNo} - i| < \text{Threshold}, \tag{7}$$

where "Threshold" is the threshold value and i is one slice of n slices.

4 Results and analysis

To make the proposed algorithm usable across all platforms and to implement all the algorithms involved in this study, we use OpenSceneGraph (version 3.6.4). All the tests are performed on different hardware platforms. Related experiments are implemented on a laptop with 8 GB NVIDIA GeForce RTX 2070 (with Max-Q Design), an Intel Core i7-8750H processor, and 16 GB of RAM. Experiments are also implemented on a Mac computer (running OS X El Capitan version 10.11.6) with a 3.2-GHz Intel Core i5 processor, 16 GB 1600 MHz of RAM, and 2 GB NVIDIA GTX 680MX.

The experimental data of the following tests are from the volumetric datasets of UZH VMML (https://www.ifi.uzh.ch/en/vmml/research/datasets.html) and open scientific visualization datasets (https://klacansky.com/open-scivis-datasets/). Those files in .raw format of volumetric datasets are loaded from the disk and parsed in real time.

Half-angle slicing for volume shadow acquires a large number of passes to complete shadow computing, so the rendering performance is relatively weak. Assuming that volume data are divided into n slices and extra passes are excluded, $2n$ passes (twice the number of slices) are needed to calculate the global volume illumination. It will result in an extremely low frame rate and difficulties for real-time interaction. On one hand, GPU is highly demanded in the massive number of passes for each frame. On the other hand, CPU is used to handle alternate passes and external updates of events and data. Once the interactive operations occur, the calculation overhead will be largely increased and spent on much more geometry slices and rendering updates.

Therefore, in the first experiment we compare

view-aligned slicing, half-angle slicing, and MSPP (including 4, 8, and 16 slices per pass) without shadow under the same volume data and presentation methods to verify the feasibility of MSPP and to study the effect of the number of slices per pass. That is, the volume shadow is not considered, and the translucent effect with the TF is consistent for each method. Different numbers of slices for Engine data (resolution: 256^3; file size: 16.8 MB) are designed to test their performance (Table 1).

The view-aligned slicing has no complicated operations, and the performance is the best, equivalent to rendering all the geometry slices in a rendering pass. However, here, we study mainly the feasibility based on MSPP. In general, as the number of slices increases, the performance of each algorithm will gradually decrease, because more geometric vertices are processed in each pass. Meanwhile, it can be seen that MSPP with different numbers of slices per pass outperforms half-angle slicing, and the time spent increases as the number of slices of the volume data increases. Even with the maximum overhead of 512 slices, the time spent of the four slices per pass is close to a quarter of that of half-angle slicing.

Specifically, view-aligned slicing has only one pass, so the rendering efficiency is the highest; half-angle slicing has n passes for n slices, and calculation resources are heavily occupied; MSPP greatly reduces the number of passes with n slices of the volume data, which is n/m depending on the number of slices per pass (m). However, a more significant number of slices per pass is not good, because the internal processing time of one pass will affect the whole time spent; thus, the number of slices per pass should have a balance for specific applications, as can be seen from rows C–E in Table 1. The experiment shows that the rendering performance of volumetric datasets can be significantly improved by reducing the number of passes. At the same time, volume shadow is not considered, and only limited textures are used for the experiment.

Table 1 Performance comparison of the three methods without shadow in different numbers of slices for Engine data

Case	Average time spent (ms)					
	16	32	64	128	256	512
A	1.00	1.04	1.05	1.06	1.15	1.61
B	1.25	1.77	2.78	4.81	9.58	19.34
C	1.05	1.06	1.34	1.85	2.87	5.28
D	1.06	1.06	1.10	1.33	1.85	2.93
E	1.08	1.07	1.15	1.33	1.69	2.30

A: view-aligned slicing; B: half-angle slicing; C: MSPP with 4 slices per pass; D: MSPP with 8 slices per pass; E: MSPP with 16 slices per pass. The data are divided into six different numbers of slices from 16 to 512 increased by a power of 2

Meanwhile, the unified resolution of the experiments is 800 × 600 (for both images and videos). If the current density value of Engine data in a shader is more significant than 0.1, these calculations are performed only to remove air and other low-intensity artifacts to produce much better results, as shown in Fig. 6. We can see that each algorithm can present the outline and internal details of the volumetric datasets. The TF is generated as the maximum intensity projection value by looking up in the one-dimensional (1D) color table after interpolation. The geometry meshes become dense as the number of slices increases. The other methods have just some geometry meshes per pass, except for the view-aligned 3D texture slicing, which is the whole geometry mesh of volumetric datasets for each pass. Simultaneously, as the number of slices increases, the rendering effects become clearer. The results of rendering a certain number of slices (e.g., 64 slices or more) are practical. More importantly, MSPP can achieve the effect of half-angle slicing, but faster than it.

In addition to the visual representation of volumetric datasets, the advantage of the slice-based algorithm is that it can simulate illumination scattering effects, such as the volume shadow of global illumination. Thus, in our second experiment we compare the rendering performance and effects of volume shadow algorithms of half-angle slicing (HAS) and MSPP with different numbers of slices per pass (Table 2 and Fig. 7). All the conditions of the experiment are consistent. The global volume shadow of slice-based slicing needs to be computed slice by slice and simulated in the process of illumination attenuation. Compared with HAS, MSPP increases the complexity of illumination simulation.

As can be seen from Table 2, the cost of volume shadow calculation of both HAS and MSPP is very large; additionally, as the number of slices increases, the overhead of each algorithm is increasing and the performance is drastically decreasing. However, MSPP with a different number of slices per pass can improve performance by approximately two times in the same number of slices compared

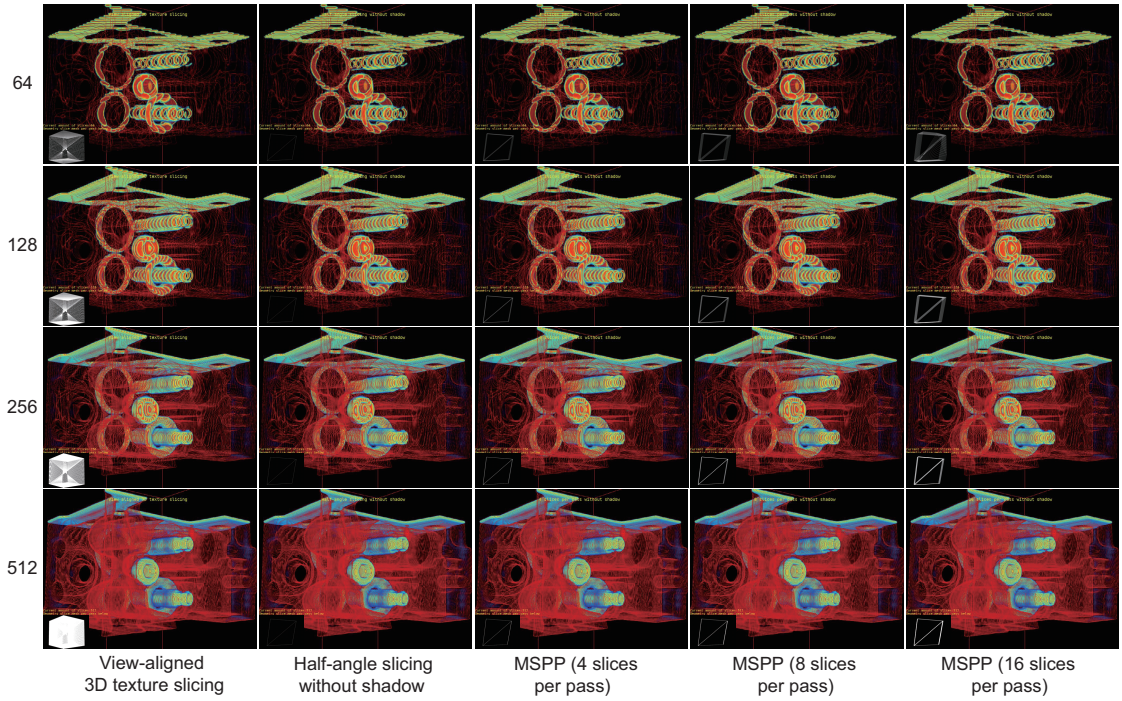

Fig. 6 Rendering effects of view-aligned 3D texture slicing, half-angle slicing without shadow, and MSPP of 4, 8, and 16 slices per pass. All results are presented in the same way as a translucent effect with transfer function (TF) in different numbers of slices. Meanwhile, images at the bottom left are the geometry slice mesh per pass for each method

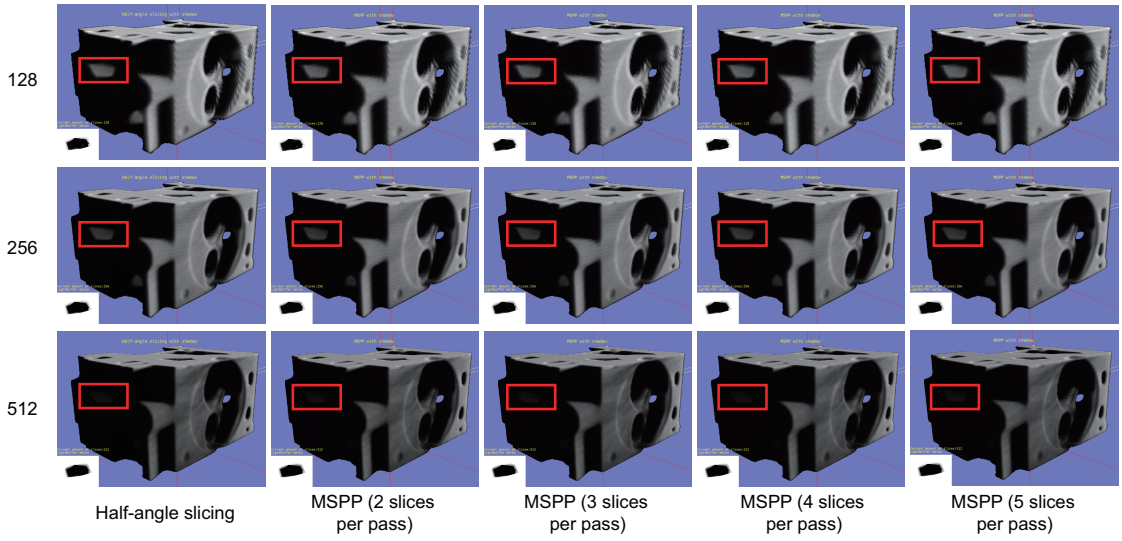

Fig. 7 Effects of volume shadow of half-angle slicing and MSPP with different numbers of slices per pass. The illuminated area within the red frame is the case where the light source is calculated through the space structure of volumetric datasets. Meanwhile, images at the bottom left are the illumination attenuation buffers of the last slice

with HAS. It is mainly because MSPP has reduced the rendering passes from $2n$ (twice the number of slices n) to $2n/m$ (m slices per pass); thus, the overhead of CPU and GPU for interactive events and data updates is significantly reduced.

At the same time, it can be seen that the rendering performance of MSPP does not increase linearly as the number of slices per pass increases, but decreases. The main reason is that the number of shadow buffers required for illumination attenuation

calculation increases as the number of slices per pass increases. Moreover, the storage overhead is increased so that the time spent does not increase with the increase in the number of slices. The memory overheads of HAS and MSPP for volume shadow are shown in Fig. 8. This is consistent with the results of the second experiment about the different numbers of slices. As the number of slices increases, the memory overhead of each method will gradually increase. However, with the increasing number of slices per pass, the memory overhead of MSPP does not decrease linearly but stabilizes gradually. It is because MSPP increases the number of slices per pass. The buffers for illumination attenuation are gradually increased, just in balance with the advantage of reducing the number of passes.

The attenuation buffer accumulates the illumination attenuation slice by slice, similar to half-angle slicing for volume shadow. Although the number of illumination attenuation passes is increased by the number of total passes, because these passes are processed by image-based pass, it does not add too much calculation overhead. The results of MSPP for volume shadow (Fig. 7) are consistent with those of HAS. Meanwhile, as the number of slices increases, multiplying each slice by $(1+\alpha)^n$ (α is an opacity value of the current slice and n is an adjustable factor of illumination attenuation) compensates for the fact that more and more slices are over-dark due to excessive accumulation of illumination attenuation.

More test effects of MSPP are shown in Fig. 9, including plants and medical datasets. The tests of

Table 2 Performance comparison of the two methods with volume shadow in different numbers of slices for Engine data

Case	Average time spent (ms)				
	32	64	128	256	512
A	2.84	4.69	8.93	19.13	41.70
B	1.87	3.04	5.25	10.24	19.52
C	1.54	2.29	3.45	7.01	12.22
D	1.47	2.08	3.34	5.81	10.62
E	1.62	2.42	3.91	6.95	14.71
F	1.80	2.72	4.41	7.77	13.86

A: half-angle slicing; B: MSPP with 2 slices per pass; C: MSPP with 3 slices per pass; D: MSPP with 4 slices per pass; E: MSPP with 5 slices per pass; F: MSPP with 6 slices per pass. The volume data are divided into five different numbers of slices from 32 to 512 increased by a power of 2

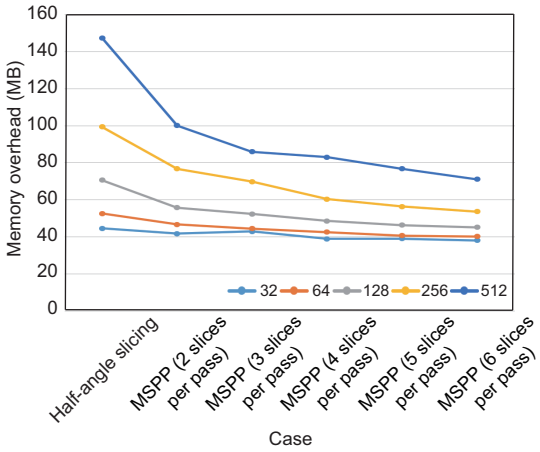

Fig. 8 Memory overhead of half-angle slicing and MSPP for volume shadow in different numbers of slices from 32 to 512 increased by a power of 2

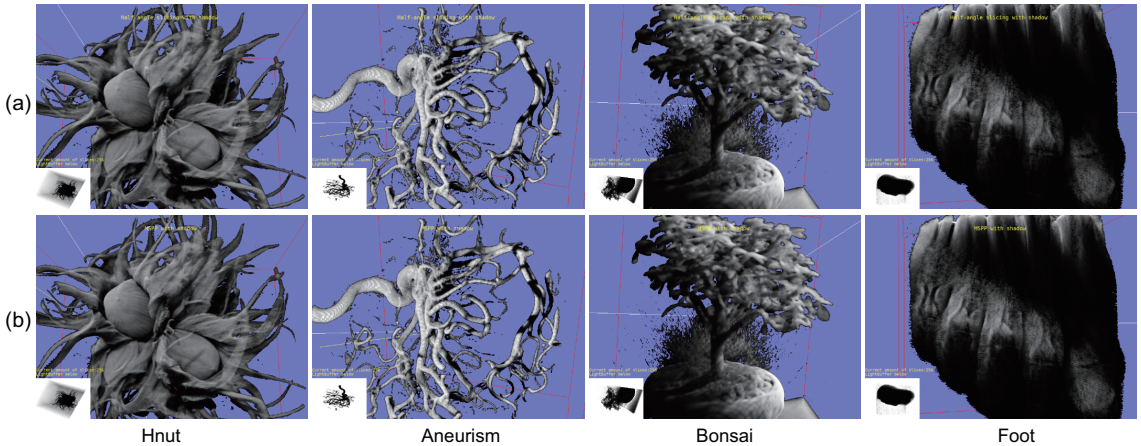

Fig. 9 Effect comparison for volume shadow between half-angle slicing (a) and MSPP (b). The four volumetric datasets from the left to right are Hnut (resolution: 512^3; file size: 128 MB), Aneurism (resolution: 256^3; file size: 16 MB), Bonsai (resolution: 256^3; file size: 16 MB), and Foot (resolution: 256^3; file size: 16 MB). Meanwhile, images at the bottom left are the illumination attenuation buffers of the last slice

real-world volumetric datasets show that MSPP has a wide range of applications. As can be seen from the overall effects, images of MSPP are the same with half-angle slicing in illumination and shadow. There is another problem that slice-based volume rendering can cause artifacts (Belyaev et al., 2019). It is related to the texture mapping in the 3D graphics rendering process. However, there are some ways to reduce slicing artifacts. On one hand, the number of slices and per-slice image resolution can be increased. On the other hand, post-processing anti-aliasing methods can be used to improve image effects.

MSPP can be easily integrated into parallel rendering frameworks based on sort-first or sort-last algorithms to accelerate volume rendering. The geometry slices of volumetric datasets can be flexibly managed and updated for MSPP applications. In the parallel rendering framework, two stages are executed in parallel. One is that each server thread executes rendering tasks to local images, and the other is that all the local images are composed into the final result. In MSPP for parallel volume rendering, slices of an adjacent rendering pass can be simply divided into different rendering servers. These rendering servers accept only slices of a nearby pass, so the locality of the data can be guaranteed.

According to the "over" operator satisfying the conjunction rule, n passes are divided into m subsets. Each subset contains several adjacent passes, like this $\{s_1, s_2, \ldots, s_{k_1}\}$, $\{s_{k_1+1}, s_{k_1+2}, \ldots, s_{k_2}\}$, $\ldots$, $\{s_{k_{m-1}+1}, s_{k_{m-1}+2}, \ldots, s_n\}$, so that all images are composed using the sort-last parallel algorithm as

$$(s_1 \text{ over } s_2 \text{ over } \ldots \text{ over } s_{k_1}) \text{ over } (s_{k_1+1} \text{ over } s_{k_1+2} \text{ over } \ldots \text{over } s_{k_2}) \text{ over } \ldots \text{ over } (s_{k_{m-1}+1} \text{ over } s_{k_{m-1}+2} \text{ over } \ldots \text{ over } s_n). \quad (8)$$

5 Conclusions and future work

In general, the MSPP method of slice-based volume rendering can quickly implement the visualization of volumetric datasets and some volume effects, such as translucent effect with TF and volume shadow. In contrast to other slice-based volume rendering algorithms such as half-angle slicing, MSPP can improve the rendering performance for volume shadow by at least two times and effectively reduce the memory overhead. Meanwhile, the scalability of MSPP can be flexibly applied to more volumetric datasets and rendering applications. Combined with the parallel rendering framework, it also dramatically improves the performance of volume rendering. However, there are also some problems with MSPP to represent the better volume effects, such as the lack of a combination of more methods over the state of the art.

We will further optimize and improve the MSPP method to form a mature framework system and combine it with multi-computer parallel rendering frameworks. Meanwhile, more related volume rendering methods (such as traditional volume rendering ray casting) combined with MSPP will be studied. Furthermore, soft shadow, slicing artifacts, and illumination scattering will be solved.

Contributors

Dening LUO and Jianwei ZHANG designed the research. Dening LUO processed the data and drafted the manuscript. Jianwei ZHANG and Yi LIN helped organize the manuscript. Dening LUO and Jianwei ZHANG revised and finalized the paper.

Compliance with ethics guidelines

Dening LUO, Yi LIN, and Jianwei ZHANG declare that they have no conflict of interest.

References

Angelelli P, Bruckner S, 2015. Performance and quality analysis of convolution-based volume illumination. *J WSCG*, 23(2):131-138.

Belyaev SY, Smirnova ND, Smirnov PO, et al., 2019. Fast selective antialiasing for direct volume rendering. Proc SPIE, Medical Imaging: Imaging Informatics for Healthcare, Research, and Applications, 1095407. https://doi.org/10.1117/12.2511887

Beyer J, Hadwiger M, Pfister H, 2015. State-of-the-art in GPU-based large-scale volume visualization. *Comput Graph Forum*, 34(8):13-37. https://doi.org/10.1111/cgf.12605

Bordoloi UD, Shen HW, 2005. View selection for volume rendering. Proc IEEE Visualization, p.487-494. https://doi.org/10.1109/VISUAL.2005.1532833

Çalışkan A, Çevik U, 2015. Overview of computer graphics and algorithms. Proc 23rd Signal Processing and Communications Applications Conf, p.831-834. https://doi.org/10.1109/SIU.2015.7129957

El Seoud MSA, Mady AS, 2019. A comprehensive review on volume rendering techniques. Proc 8th Int Conf on Software and Information Engineering, p.126-131. https://doi.org/10.1145/3328833.3328878

Engel K, Kraus M, Ertl T, 2001. High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. Proc ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware, p.9-16. https://doi.org/10.1145/383507.383515

Fernando R, 2004. GPU Gems: Programming Techniques, Tips, and Tricks for Real-Time Graphics. Addison-Wesley, Boston, USA.

Hachaj T, 2014. Real time exploration and management of large medical volumetric datasets on small mobile devices—evaluation of remote volume rendering approach. *Int J Inform Manag*, 34(3):336-343. https://doi.org/10.1016/j.ijinfomgt.2013.11.005

Hänel C, Weyers B, Hentschel B, et al., 2016. Visual quality adjustment for volume rendering in a head-tracked virtual environment. *IEEE Trans Vis Comput Graph*, 22(4):1472-1481. https://doi.org/10.1109/TVCG.2016.2518338

Jönsson D, Sundén E, Ynnerman A, et al., 2014. A survey of volumetric illumination techniques for interactive volume rendering. *Comput Graph Forum*, 33(1):27-51. https://doi.org/10.1111/cgf.12252

Khan NM, Ksantini R, Guan L, 2018. A novel image-centric approach toward direct volume rendering. *ACM Trans Intell Syst Technol*, 9(4):42. https://doi.org/10.1145/3152875

Kniss J, Premoze S, Hansen C, et al., 2002. Interactive translucent volume rendering and procedural modeling. Proc IEEE Visualization, p.109-116. https://doi.org/10.1109/VISUAL.2002.1183764

Kruger J, Westermann R, 2003. Acceleration techniques for GPU-based volume rendering. Proc IEEE Visualization, p.287-292. https://doi.org/10.1109/VISUAL.2003.1250384

Lacroute P, Levoy M, 1994. Fast volume rendering using a shear-warp factorization of the viewing transformation. Proc 21st Annual Conf on Computer Graphics and Interactive Techniques, p.451-458. https://doi.org/10.1145/192161.192283

Laur D, Hanrahan P, 1991. Hierarchical splatting: a progressive refinement algorithm for volume rendering. *ACM SIGGRAPH Comput Graph*, p.285-288. https://doi.org/10.1145/127719.122748

Ljung P, Krüger J, Groller E, et al., 2016. State of the art in transfer functions for direct volume rendering. *Comput Graph Forum*, 35(3):669-691. https://doi.org/10.1111/cgf.12934

Ma B, Entezari A, 2018. Volumetric feature-based classification and visibility analysis for transfer function design. *IEEE Trans Vis Comput Graph*, 24(12):3253-3267. https://doi.org/10.1109/TVCG.2017.2776935

Mwalongo F, Krone M, Reina G, et al., 2016. State-of-the-art report in web-based visualization. *Comput Graph Forum*, 35(3):553-575. https://doi.org/10.1111/cgf.12929

Noguera JM, Jimenez JR, 2016. Mobile volume rendering: past, present, and future. *IEEE Trans Vis Comput Graph*, 22(2):1164-1178. https://doi.org/10.1109/TVCG.2015.2430343

Patel D, Šoltészová V, Nordbotten JM, et al., 2013. Instant convolution shadows for volumetric detail mapping. *ACM Trans Graph*, 32(5):154. https://doi.org/10.1145/2492684

Preim B, Baer A, Cunningham D, et al., 2016. A survey of perceptually motivated 3D visualization of medical image data. *Comput Graph Forum*, 35(3):501-525. https://doi.org/10.1111/cgf.12927

Rodríguez MB, Alcocer PPV, 2012. Practical volume rendering in mobile devices. Proc 8th Int Symp on Advances in Visual Computing, p.708-718. https://doi.org/10.1007/978-3-642-33179-4_67

Rostamzadeh N, Jönsson D, Ropinski T, 2013. A comparison of volumetric illumination methods by considering their underlying mathematical models. Proc SIGRAD, Visual Computing, p.35-40.

Schlegel P, Makhinya M, Pajarola R, 2011. Extinction-based shading and illumination in GPU volume ray-casting. *IEEE Trans Vis Comput Graph*, 17(12):1795-1802. https://doi.org/10.1109/TVCG.2011.198

Schott M, Pegoraro V, Hansen C, et al., 2009. A directional occlusion shading model for interactive direct volume rendering. *Comput Graph Forum*, 28(3):855-862. https://doi.org/10.1111/j.1467-8659.2009.01464.x

Šoltészová V, Patel D, Bruckner S, et al., 2010. A multidirectional occlusion shading model for direct volume rendering. *Comput Graph Forum*, 29(3):883-891. https://doi.org/10.1111/j.1467-8659.2009.01695.x

Stegmaier S, Strengert M, Klein T, 2005. A simple and flexible volume rendering framework for graphics-hardware-based raycasting. Proc 4th Int Workshop on Volume Graphics, p.187-195. https://doi.org/10.1109/VG.2005.194114

Usher W, Amstutz J, Brownlee C, et al., 2017. Progressive CPU volume rendering with sample accumulation. Proc 17th Eurographics Symp on Parallel Graphics and Visualization, p.21-30. https://doi.org/10.2312/pgv.20171090

Wangkaoom K, Ratanaworabhan P, Thongvigitmanee SS, 2015. High-quality web-based volume rendering in real-time. Proc 12th Int Conf on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology, p.1-6. https://doi.org/10.1109/ECTICon.2015.7207091

Zhang Q, Eagleson R, Peters TM, 2011. Volume visualization: a technical overview with a focus on medical applications. *J Dig Imag*, 24(4):640-664. https://doi.org/10.1007/s10278-010-9321-6

Zhang YB, Ma KL, 2013. Lighting design for globally illuminated volume rendering. *IEEE Trans Vis Comput Graph*, 19(12):2946-2955. https://doi.org/10.1109/TVCG.2013.172