A-PARM: Adaptive Division of Sub-cells in the PARM for Efficient Volume Ray Casting

Sukhyun Lim and Byeong-Seok Shin

Inha University, Dept. Computer Science and Information Engineering,
253 Yonghyun-dong, Nam-gu, Incheon, 402-751, Rep. of Korea
slim@inhaian.net, bsshin@inha.ac.kr

Abstract. The PARM is a data structure to ensure interactive frame rates on a PC platform for CPU-based volume ray casting. After determining candidate cells that contribute to the final images, it partitions each candidate cell into several sub-cells. Then, it stores trilinearly interpolated scalar value and an index of encoded gradient vector for each sub-cell. Since the information that requires time-consuming computations is already stored in the structure, the rendering time is reduced. However, it requires huge memory space because most precomputed values are loaded in the system memory. We solve it by adaptively dividing candidate cells into different sub-cells. That is, we divide a candidate cell in which the gradient is strictly changed into a large number of sub-cells, and vice versa. By this approach, we acquire moderate images while reducing the memory size.

1 Introduction

Volume visualization is a research area that deals with various techniques to extract meaningful and visual information from volume data [1], [2]. Volume datasets can be represented by *voxels*, and adjacent eight voxels form a cube called a *cell*. One of the most frequently applied techniques is direct volume rendering, producing high-quality 3D rendered images directly from volume data without intermediate representation. Volume ray casting is a well-known direct volume rendering method [1]. In general, it is composed of two steps [3]: after a ray advances through transparent region (*leaping step*), the ray integrates colors and opacities as it penetrates an object boundary (*color computation step*). Although volume ray casting produces high-quality images, the rendering speed is too slow [1], [2], [3].

Several software-based acceleration techniques for direct volume rendering have been proposed to reduce rendering time. Yagel and Kaufman exploited the use of a ray template to accelerate ray traversal, using spatial coherence of the ray trajectory [4]. However, it is difficult to apply their method to applications involving perspective projection. The shear-warp method introduced by Lacroute and Levoy rearranges the voxels in memory to allow optimized ray traversal (shearing the volume dataset) [5]. Although this method can produce images from reasonable volumes at quasi-interactive frame rates, image quality is sacrificed because bilinear interpolation is used as a convolution kernel. Parker et al. demonstrated a full-featured ray tracer on a workstation with large shared memory [6]. Unfortunately, this method requires 128 CPUs for interactive

rendering and is specialized for isosurface rendering. Although Knittel presented the interleaving of voxel addresses and deep optimizations (by using the MMXTM instruction) to improve the cache hit ratio [7], it generates only 256×256 pixel images. If we want a high-resolution image, the 256×256 image can be magnified by using graphics hardware. Mora et al. proposed a high performance projection method that exploits a spread-memory layout called object-order ray casting [8]. However, this also does not support perspective projection. Grimm et al. introduced acceleration structures [9]. They used a gradient cache, and a memory-efficient hybrid removal and skipping technique for transparent regions. This method achieves fast rendering on a commodity notebook PC, but it is not applicable to perspective projection.

We call a cell of which the enclosing eight voxels are all transparent as a *transparent cell*, and a cell with eight opaque voxels is an *opaque cell*. The *candidate cell* means a cell that contributes to the final images. After dividing each candidate cell into N_{sc}^3 cells, where N_{sc} is a representative on one axis to divide one candidate cell into several cells, we call the resulting cell as a *sub-cell*. That is, a single candidate cell contains N_{sc}^3 sub-cells. Fig. 1 shows the structure of a candidate cell.

Fig. 1. Structure of a single candidate cell. A candidate cell is composed of several sub-cells. In this example, one candidate cell is composed of 125 sub-cells (that is, $N_{sc}=5$).

Recently, Lim and Shin proposed the PARM (Precomputed scAlar and gRadient Map) structure to accelerate the color computation step of the volume ray casting [10]. This structure stores the information required to compute color values of the CPU-based volume ray casting. At preprocessing time, after determining candidate cells that contribute to the final images, it divides each candidate cell into several sub-cells. Then, it stores trilinearly interpolated scalar value and an index of encoded gradient vector for each sub-cell. During the rendering stage, when a ray lies on an arbitrary sample point in a candidate cell after skipping over transparent region, a color value can be determined without time-consuming computations because most values to compute a color value are already stored in the PARM structure. However, because it stores the values in the system memory, it is a burden to the memory when the number of candidate cells is increased.

If a sampling rate is reduced in the candidate cell in which the gradient is strictly changed, we acquire aliased or deteriorated images. Although the previous PARM structure assigns fixed numbers of sub-cells for entire candidate cells, our method adaptively divide the candidate cells into different numbers of sub-cells according to

the gradient variations. So, we call it *A-PARM* (the initial ‘A’ means that ‘adaptive’). We explain our data structure in detail in Section 2, and experimental results are presented in Section 3. Finally, we conclude our work.

2 A-PARM (Adaptive-Precomputed scAlar and gRradient Map)

Our method focuses on reducing the memory size for the PARM structure while preserving moderate image quality. Through the next section, we can recognize the method in detail.

2.1 PARM

The obvious way to accelerate the color computation step of the CPU-based ray casting is that we precompute required values in preprocessing step and refer to the values in rendering step. For that, we proposed the PARM structure in our previous work [10]. The first generation step of the PARM is to determine candidate cells from the entire volume [10]. Then, we compute required values. However, because computing the positions of sample points is not feasible because the points can lie in an arbitrary position in a candidate cell, we compute the required values per sub-cell after dividing each candidate cell into N_{sc}^3 sub-cells. The first requirement value is trilinearly interpolated scalar value, and this value is used to determine the opacity value.

The next value is a trilinearly interpolated gradient vector. However, since a vector is composed of three components (x-, y- and z-element), we require three bytes per a sub-cell. So, we apply the *gradient encoding method* proposed by Rusinkiewicz and Levoy [11]. This method maps the gradient vectors on a grid in each of the six faces of a cube. In conventional method [11], they exploited $52 \times 52 \times 6 = 16,224$ different gradient vectors. However, we increase the grid size as $104 \times 104 \times 6$ (approximately 2 bytes) to reduce visual artifacts. It leads to a mean error of below 0.01 radian. As a result, after computing trilinear interpolated gradient vector, we store an index.

Fig. 2. By the gradient encoding method, we store an index of the gradient vector instead of three components of interpolated vector

The ray-traversal procedure when we exploit the PARM is as follows: first, a ray is fired from each pixel on an image plane. After traversing transparent cells using the

conventional space-leaping methods, it reaches a candidate cell. Second, after selecting the nearest sub-cell, it refers to the precomputed scalar value from the PARM. Because the scalar value on a sample point is already interpolated, it does not require time-consuming trilinear interpolation. If the value is within the transparent range, the ray advances to the next sample point. Otherwise (that is, if the scalar value is regarded as nontransparent), the ray also refers to the encoded gradient vector index. To decode representable vector from the index quickly, we use a lookup table [11]. Lastly, a color value is determined from the referred gradient vector. Those steps continue for all rays in the image plane.

2.2 A-PARM

When we exploit the PARM structure, we accelerate the color computations of the CPU-based volume ray casting. However, because most required values in the PARM are calculated in preprocessing stage, it requires long preprocessing time and huge memory size. One of the challenging issues of the volume rendering is to acquire high-quality images. When we exploit the PARM structure, the simple way to satisfy it is that we can increase the number of sub-cells. However, unfortunately, the preprocessing time is increased according to increase the number of them, and we can also require large memory space.

The obvious way to solve it while increasing the image quality is to adaptively divide candidate cells into different numbers of sub-cells. In our method, therefore, we increase the number of them in which the gradient is strictly changed. That is, if the gradient is over the user-defined threshold (τ), we increase the number of them, and if the gradient is below τ , we reduce the number of them. The steps are as follows; at first, we estimate three components of the gradient vector $\nabla f(\mathbf{x}) = \nabla f(x, y, z)$ for all candidate cells. To determine the candidate cells, we used original method [10]. We assume that a volume dataset is composed of N voxels, and each voxel $\mathbf{v}$ is indexed by $\mathbf{v}(x, y, z)$ where $x, y, z = 0, \dots, N-1$. Eq. (1) shows our approach when we use the central difference method [3]. It is the most common approach for gradient estimation in volume graphics [3]. If one of the eight voxels in a candidate cell is satisfied with the Eq. (1), we increase the number of sub-cells.

$$\nabla f(x, y, z) \approx \frac{1}{2} \begin{pmatrix} f(x+1, y, z) - f(x-1, y, z) \\ f(x, y+1, z) - f(x, y-1, z) \\ f(x, y, z+1) - f(x, y, z-1) \end{pmatrix} > \tau . \quad (1)$$

In implement aspect, the PARM is composed of two data structures; *INDEX BUFFER* and *DATA BUFFER*. The *DATA BUFFER* stores precomputed values for each sub-cell. And, to refer the values from it, a data structure stored on the indices of the candidate cells is required, and this is called *INDEX BUFFER*. Because the N_{sc} value is already determined (once again, N_{sc} is a representative on one axis to divide one candidate cell into several cells), we refer expected values from the *DATA BUFFER* quickly. That is, if a ray lies on a sub-cell S_j ($0 \leq S_j < N_{sc}$) in a candidate cell, we refer an index C_i of the candidate cell from the *INDEX BUFFER*. Then, we acquire a precomputed scalar value and an index of the encoded gradient vector from the *DATA BUFFER* by referring the index of $N_{sc} \times (C_i - 1) + S_j$.

Those two buffers are optimal for the original PARM structure. However, since we divide candidate cells into different numbers of sub-cells according to the gradient variations, we cannot exploit them. One of the simple methods is to generate several *INDEX BUFFERS*. However, it is a burden to the system memory because the size of one *INDEX BUFFER* is identical to the volume dataset. Therefore, we require other smart approach.

To solve it, we generate a modified *INDEX BUFFER*, named *M-INDEX BUFFER*. Firstly, from the previous *INDEX BUFFER*, we rearrange indices according to gradient variations, by using the sorting algorithm. We assume that a candidate cell is divided into N_{sc1}^3 , N_{sc2}^3 or N_{sc3}^3 sub-cells according to gradients. And we divide a candidate cell into N_{sc1}^3 sub-cells if the gradient is steep, and when it is moderate we divide it into N_{sc2}^3 sub-cells. Otherwise (that is, if it is gentle), we partition it into N_{sc3}^3 sub-cells. Secondly, after searching all candidate cells from the previous *INDEX BUFFER*, we rearrange indices as the sequence from N_{sc1}^3 to N_{sc3}^3 sub-cells. By this approach, only one size of the *INDEX BUFFER* is required. Of course, it requires additional generation time compared with the conventional method. However, its time is marginal against the increment of memory size because only the $O(N)$ time is necessary. Fig. 3 shows our *M-INDEX BUFFER* generation method.

Fig. 3. (Upper) when we use conventional data structure, we require three *INDEX BUFFERS*. Therefore, it is not adequate for our method. (Lower) although we divide candidate cells into three numbers of sub-cells according to the gradient variations, we can require only one *INDEX BUFFER* size by using the *M-INDEX BUFFER*. The red, blue, and black squares mean candidate cells as the gradients are steep, moderate, and gentle, respectively.

Besides the *M-INDEX BUFFER*, we store total numbers of candidate cells related to the N_{sc1} , N_{sc2} or N_{sc3} , and we call them as $\#N_{sc1}$, $\#N_{sc2}$ or $\#N_{sc3}$ (for example, in Fig. 3, they are 12, 10, and 12, respectively). In ray traversal, if a ray reaches a candidate cell and its index of the *M-INDEX BUFFER* is below $\#N_{sc1}$ (that is, $0 \leq C_i < \#N_{sc1}$), we

acquire expected precomputed structure by referring Eq. (2). When the index of it is over $\#N_{sc1}$ and below $\#N_{sc2}$ (that is, $\#N_{sc1} \leq C_i < (\#N_{sc1} + \#N_{sc2})$), we reference the index by Eq. (3). Otherwise (that is, $\#N_{sc1} + \#N_{sc2} \leq C_i < (\#N_{sc1} + \#N_{sc2} + \#N_{sc3})$), we refer the Eq. (4). Other remaining steps are identical with the previous PARM (see Sect. 2.1).

$$N_{sc1} \times (C_i - 1) + S_j. \quad (2)$$

$$(N_{sc1} \times \#N_{sc1}) + (N_{sc2} \times (C_i - \#N_{sc1} - 1) + S_j). \quad (3)$$

$$(N_{sc1} \times \#N_{sc1} + N_{sc2} \times \#N_{sc2}) + (N_{sc3} \times (C_i - (\#N_{sc1} + \#N_{sc2}) - 1) + S_j). \quad (4)$$

3 Experimental Results

All the methods were implemented on a PC equipped with an AMD Athlon64x2™ 4200+ CPU, 2 GB main memory, and GeForce™ 6600 graphics card. The graphics card capabilities are only used to display the final images. The first and second dataset were an engine block and a bonsai with resolutions of 512^3 , respectively. The third dataset was an aneurysm of a human brain vessel with a resolution of 768^3 . Fig. 4 shows OTFs (Opacity Transfer Functions) for the datasets, and we divide candidate cells into three different numbers of sub-cells (τ) according to the gradient variations. If one scalar value of the three elements in $\nabla f(x, y, z)$ for a candidate cell is over 150, we divide the candidate cell into 64 sub-cells (that is, $N_{sc1}^3 = 4^3 = 64$). When the gradient is from 50 to 149, we partition the cell into 27 sub-cells ($N_{sc2}^3 = 3^3 = 27$). Otherwise, we divide into $N_{sc3}^3 = 2^3 = 8$ sub-cells.

Fig. 4. The OTFs for the engine block, bonsai, and aneurysm dataset, respectively

The rendering times to produce the perspective view are shown in Table 1. Our method is at least 50% faster than the method using the previous hierarchical min-max octree, and this time is almost same with the previous PARM structure (the tolerance is below 5%). The hierarchical min-max octree is widely used data structure for empty-space skipping [1], [7], [10]. Table 2 shows the preprocessing time and required memory for each dataset. Compared with the previous PARM, we can reduce the preprocessing time and memory storage as the amount of 26%.

Table 1. Rendering improvements for each dataset before/after our method (unit: %). The time for space leaping is included in our method. All results are rendered to a perspective projection.

dataset	engine	bonsai	aneurysm	average
improvements	55	49	46	50

Table 2. Preprocessing time and required memory for each dataset. The previous PARM divides all candidate cells into 64 sub-cells.

dataset	engine	bonsai	aneurysm
the number of candidate cells (voxels)	5,520,128	4,865,792	2,579,232
candidate cell determination time (A) (secs)	7.24	6.01	3.30
generation time of the PARM (B) (secs)	168.02	147.84	76.97
generation time of the A-PARM (C) (secs)	114.09	107.7	58.89
total preprocessing time			
of the PARM (D=A+B) (secs)	175.26	153.85	80.27
total preprocessing time			
of the A-PARM (E=A+C) (secs)	121.33	113.71	62.19
efficiency of the preprocessing (D/E) (%)	30.8	26.1	22.5
required memory of the PARM (F) (MB)	1328	1203	764
required memory of the A-PARM (G) (MB)	907	896	602
efficiency of the required memory (F/G) (%)	31.7	25.5	21.2

Fig. 5 shows the image quality. As you can see, the image quality using our structure is almost same with that using the previous PARM. We compute the *Root Mean Square Error* (RMSE) [12] for accurate comparisons. Compared with the results of the previous PARM, the mean value of RMSE results is about 2.04. This is marginal against reducing the preprocessing time and memory storage.

Fig. 5. Comparisons of image quality: the previous PARM (left) and A-PARM (right) for each dataset. The RMSE results are 1.15, 2.89, and 2.09. The engine is rendered to parallel projection and the bonsai and aneurysm are projected to perspective viewing. And, in case of bonsai, we magnify the soil region for accurate comparisons.

4 Conclusion

The most important issue in volume visualization is to produce high-quality images in real time. To achieve interactive frame rates on a PC platform, the PARM structure is proposed in our previous work. Although it reduces the rendering time by precomputing a trilinearly interpolated scalar value and an index of encoded gradient vector for each sub-cell, the preprocessing time is increased. Moreover the memory storage is also increased since most precomputed values are loaded in the system memory. To solve them, we adaptively divide candidate cells into different numbers of sub-cells according to the gradient variations. The experimental results show that our method reduces rendering time and produces high-quality images while reducing the preprocessing time and memory storage.

Acknowledgment

This work was supported by IITA through IT Leading R&D Support Project.

References

1. Levoy, M.: Display of Surface from Volume Data. *IEEE Computer Graphics and Applications*, Vol. 8, No. 3 (1988) 29-37
2. Kaufman, A.: *Volume Visualization*. 1st ed., Ed. IEEE Computer Society Press (1991)
3. Engel, K., Weiskopf, D., Rezk-salama, C., Kniss, J., Hadwiger, M.: *Real-time Volume Graphics*, AK Peters (2006)
4. Yagel, R., Kaufmann, A.: Template-based Volume Viewing. *Computer Graphics Forum*, Vol. 11 (1992) 153-167
5. Lacroute, P., Levoy, M.: Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation. *Proc. SIGGRAPH 1994* (1994) 451-457
6. Parker, S., Shirley, P., Livnat, Y., Hansen, C., Sloan, P.: Interactive Ray Tracing for Iso-surface Rendering. *Proc. IEEE Visualization 1998* (1998) 233-238
7. Knittel, G.: The UltraVis System. *Proc. IEEE Volume Visualization 2000* (2000) 71-79
8. Mora, B., Jessel, J.P., Caubet, R.: A New Object Order Ray-Casting Algorithm. *Proc. IEEE Volume Visualization 2002* (2002) 203-210
9. Grimm, S., Bruckner, S., Kanitsar, A., Gröller, E.: Memory Efficient Acceleration Structures and Techniques for CPU-based Volume Raycasting of Large Data. *Proc. IEEE Volume Visualization 2004* (2004) 1-8
10. Lim, S., Shin, B-S: PARM: Data Structure for Efficient Volume Ray Casting. *Lecture Notes in Computer Science*, Vol. 4263 (2006) 296-305
11. Rusinkiewicz, S., Levoy, M.: QSplat: A Multiresolution Point Rendering System for Large Meshes. *Proc. SIGGRAPH 2000* (2000) 343-352
12. Kim, K., Wittenbrink, C.M., Pang, A.: Extended Specifications and Test Data Sets for Data Level Comparisons of Direct Volume Rendering Algorithms. *IEEE Trans. on Visualization and Computer Graphics*, Vol. 7 (2001) 299-317