실시간 볼륨 광선 투사법을 위한 자료구조

임석현[0], 신병석
인하대학교 미디어연구실
slim@inhaian.net bsshin@inha.ac.kr

A Data Structure for Real-time Volume Ray Casting

Sukhyun Lim[0], Byeong-Seok Shin
Media Lab, Dept. Computer Science & Information Engineering, Inha University

Abstract

Several optimization techniques have been proposed for volume ray casting, but these cannot achieve real-time frame rates. In addition, it is difficult to apply them to some applications that require perspective projection. Recently, hardware-based methods using 3D texture mapping are being used for real-time volume rendering. Although rendering speed approaches real time, the larger volumes require more swapping of volume bricks for the limited texture memory. Also, image quality deteriorates compared with that of conventional volume ray casting. In this paper, we propose a data structure for real-time volume ray casting named PERM (Precomputed dEnsity and gRadient Map). The PERM stores interpolated density and gradient vector for quantized cells. Since the information requiring time-consuming computations is stored in the PERM, our method can ensure interactive frame rates on a consumer PC platform. Our method normally produces high-quality images because it is based on conventional volume ray casting.

1. Introduction

Direct volume rendering produces a projected image directly from volumetric data without intermediate representation. Volume ray casting is a well-known volume rendering method [1]. Although it produces high quality images, the rendering speed is too slow because of several time-consuming computations. To avoid these calculations, we propose a data structure, named PERM, for real-time volume ray casting. After determining candidate cells, each cell is quantized into the number of N_{qc}^3 defined by the user. N_{qc} is the quantized basis. Figure 1 shows the structure of single candidate cell. A candidate cell contains the N_{qc}^3 quantized cells. We calculate the density and the gradient vector per quantized cell in preprocessing time. In rendering stage, the color value can be acquired without time-consuming computation since most of the data required for volume ray casting are stored in the PERM. As a result, the PERM can reduce rendering time.

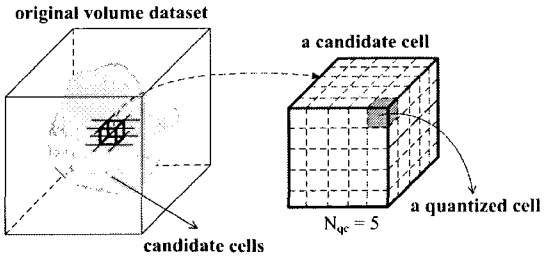

Figure 1: Structure of quantized cells. A candidate cell is composed of several quantized cells.

The next section gives a brief description of previous work, and we contemplate conventional volume ray casting in section 3. In section 4, we explain the PERM in detail. In the fifth section, we present the method of reducing the number of candidate cells. The method of diminishing the memory space is proposed in the sixth section. Experimental results are shown in section 7. Finally, we summarize and conclude our work.

2. Related Work

Several acceleration techniques have been proposed to reduce rendering time in volume visualization. Yagel and Kaufman proposed the use of the ray template to accelerate ray traversal [2]. They used a spatial coherence of the ray trajectory, but it is difficult to apply their method to perspective projection. The shear-warp method proposed by Lacroute and Levoy rearranges the voxels in the memory to allow optimized ray traversal (shearing of the volume data set) [3]. Although this method can produce images from reasonable volumes at quasi-interactive frame rates, image quality is sacrificed because bilinear interpolation as a convolution kernel is used.

Texture mapping is a widely supported technique in 3D graphics hardware [4, 5]. Cabral et al. popularized texture mapping for volume visualization [6]. Recently, GPU-based methods have been proposed to use high-parallel vertex and fragment shading code [7, 8]. Pre-integrated volume rendering is a technique to improve the quality of 3D texture mapping [9, 10, 11]. Since much of the necessary computation is performed in advance, it can generate high-quality images compared with conventional 3D texture mapping. Although it is available on recent graphics accelerators, the limited amount of texture memory restricts the accuracy of the classifications. Therefore, classifications containing high gradients do not render [11]. Moreover, it cannot incorporate lighting due to space constraints [10].

Udupa and Odhner proposed a data structure for volume rendering [12]. Based on a list of nontransparent voxels and a 2D array of pointers to that list, it achieves fast rendering. However, it has been restricted to parallel projection. Although it has been extended to provide a digital perspective, the rendering speed decreases [13]. Sobierarjski and Avila proposed a two-step algorithm, which projects boundary cells in the image plane using graphics hardware to identify the relevant parts of the rays [14]. Wan et al. used a projection template to reduce the computation involved in projecting boundary cells on the distance buffer [15]. It was more effective for small volumetric data and only supported parallel projection.

3. Contemplating Volume Ray Casting

Three elements influence rendering speed in volume ray casting. First, trilinear interpolation is necessary to calculate an accurate density value at each sample point. Trilinear interpolation is composed of seven linear interpolations [16]. Since it performs seven multiplications and fourteen additions, computation time is long.

Second, when the calculated density value is nontransparent, it evaluates the gradient vector. Neighboring gradient vectors are estimated to compute the gradient vector for a sample point. There are several methods of estimating the gradient vector [17]. Generally, all methods randomly access the memory and take a long time to compute. A simple way to estimate the gradient vector is to use the central difference. However, this also takes a long time owing to randomness in the memory reference pattern.

Third, to calculate an accurate gradient vector on a sample point, eight gradient vectors are interpolated by trilinear interpolation. Since the gradient vector is composed of three components (x-, y- and z-axes), trilinear interpolation is performed three times. Based on the gradient vector, the color value at each sample point is calculated.

4. PERM

To improve rendering performance of volume ray casting, the computation speed of the three elements listed in section 3 should be accelerated. The obvious way is to precompute the values in a preprocessing step, and to read them in the rendering step. For this reason, we propose a novel data structure, the PERM.

4.1. PERM

To generate our data structure, we determine candidate cells from the entire volume. A simple way is to use an opacity transfer function. Levoy proposed this technique to determine voxel materials in certain classes of volumes [18]. Let the changing position of the opacity value from a transparent to a nontransparent region be t_d. Indices of the candidate cells generated based on t_d are stored in the *Index Volume* (IV_i). If the density value of the volume dataset is less than t_d, it has a null value. In equation (1), $d(v)$ is the density of voxel v, and N_{cc} is the number of candidate cells.

$$IV_i = \begin{cases} index & if\ d(v) > t_d \\ null & otherwise \end{cases} \quad (1)$$
$$(\ where\ \ 1 \le index \le N_{cc}\ )$$

Next, we calculate the required values per quantized cell. Three items are required. The first item is the density value calculated by trilinear interpolation (D_{tri}), and is stored in the *Density Buffer* (DB_i). The gradient vector, interpolated from neighboring gradient vectors, is stored in the *Gradient Buffer* (GB_i). The last item is the sign element required to represent a negative value of the gradient vector (*SIGN*). This value is stored in *Sign Buffer* (SB_i). In equation (2), GX_{tri}, GY_{tri}, and GZ_{tri} represent each axis of the gradient vector.

$$\begin{aligned} DB_m &= D_{tri} \\ GB_{3m} &= GX_{tri} \\ GB_{3m+1} &= GY_{tri} \\ GB_{3m+2} &= GZ_{tri} \\ SB_m &= SIGN \\ (\ where\ \ m &= IV_i \times {N_{qc}}^3\ ) \end{aligned} \quad (2)$$

In general, the volume dataset is composed of positive (unsigned) values. If the data type of the gradient vector is the same as that of the volume dataset, we cannot represent a negative value. The simple method is to assign twice the size to each axis. However, this becomes a problem when there are numerous candidate cells. We can solve this problem by adding a sign field composed of a 1 byte data structure, and assigns 2 bits for each axis. For example, if the voxel size is 1 byte, a 6-byte data structure is required (2 bytes per axis). However, when we add a sign field only 4 bytes are required (the gradient vector is 3 bytes, and the sign field is 1 byte).

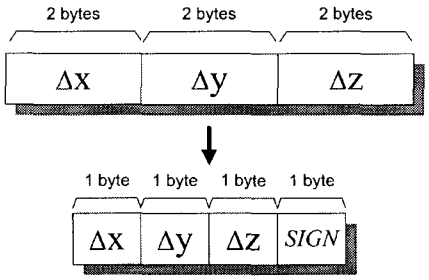

Figure 2: When we add *SIGN* filed, only 4 bytes are required. We assume that voxel size is 1 byte.

The memory requirement is denoted by equation (3). Let $\varphi(\{ds\})$ be the function returning the size of the $\{ds\}$ structure, $b_{\{ds\}}$ the number of byte(s) to assign $\{ds\}$, and $S = N_x \times N_y \times N_z$ the total volume size. The size of the *Index Volume* is equal to that of the volume size since it stores the indices of the candidate cells.

$$\begin{aligned} \varphi(PERM) &= \varphi(IV) + \varphi(DB) + \varphi(GB) + \varphi(SB) \\ &= S \times b_{iv} + (N_{cc} \times {N_{qc}}^3 \times b_{db}) + (N_{cc} \times {N_{qc}}^3 \times b_{gb} \times 3) \\ &\quad + (N_{cc} \times {N_{qc}}^3 \times b_{sb}) \\ &= S \times b_{iv} + N_{cc} \times {N_{qc}}^3 \times (b_{db} + 3b_{gb} + b_{sb}) \end{aligned} \quad (3)$$

4.2. Mapping to the Nearest Quantized Cell

The practical problem is to derive the mapping function between an arbitrary sampling point and the nearest quantized cell; when a fractional position does not exactly match with the position of the quantized cell, a mapping function is required. Although this is solved using arithmetic operations, the computation time is increased. To alleviate unnecessary calculations, we propose a *mapping lookup table* (*M_LUT*). The index of the table is divided into the *quantized basis for M_LUT* (*QBM*) and its return value is the quantized position. This table is used continuously unless the number of quantized cells changes. Equation (4) shows the structure of the *M_LUT*, and figure 3 shows an example when the *QBM* is 1000. We assume that N_{qc} is 5, and the sampling position is x_s, y_s, and z_s.

$$\begin{aligned} &M\_LUT\,[qi_x][qi_y][qi_z] \\ &= \lfloor qi_x / t \rfloor + if(qi_x \,\%\, t < t/2)\ 0:1 + \\ &\quad N_{qc} \times (\lfloor qi_y / t \rfloor + if(qi_y \,\%\, t < t/2)\ 0:1) + \\ &\quad N_{qc} \times N_{qc} \times (\lfloor qi_z / t \rfloor + if(qi_z \,\%\, t < t/2)\ 0:1) \\ &\quad (\ where\ \ qi_x = \lfloor QBM \times (x_s - \lfloor x_s \rfloor) \rfloor, \\ &\qquad\qquad qi_y = \lfloor QBM \times (y_s - \lfloor y_s \rfloor) \rfloor, \\ &\qquad\qquad qi_z = \lfloor QBM \times (z_s - \lfloor z_s \rfloor) \rfloor, \\ &\qquad\qquad t = QBM / N_{qc}\ ) \end{aligned} \quad (4)$$

4.3. Reducing the Size of *Index Volume*

When there are numerous candidate cells, we can assign the *Index Volume* as four or more bytes. In this case, the total memory is

increased. We use a summed-area table to solve the problem. This table contains the accumulated number of candidate cells along the z-axis. The *Index Volume* stores only the indices of the candidate cells per z-axis. When we use this method, the memory size decreases to about half the original *Index Volume*.

M_LUT[0][0][0] ~ M_LUT[100][0][0] = 0
M_LUT[101][0][0] ~ M_LUT[300][0][0] = 1
M_LUT[301][0][0] ~ M_LUT[500][0][0] = 2
M_LUT[501][0][0] ~ M_LUT[700][0][0] = 3
⋮
M_LUT[0][100][0] ~ M_LUT[0][300][0] = 5
M_LUT[0][301][0] ~ M_LUT[0][500][0] = 10
⋮
M_LUT[0][0][101] ~ M_LUT[0][0][300] = 25
M_LUT[0][0][301] ~ M_LUT[0][0][500] = 50
⋮
M_LUT[1000X0.48][1000X0.0][1000X0.25] = 3+5X(0+1X5) = 27
(*where* x_s=100.48, y_s=100.0, z_s=100.25)
⋮

Figure 3: An example of *M_LUT*

For example, if the volume size is 1,000,000 (i.e., N_x, N_y, and N_z are 100 respectively), $100\times100\times100\times4$ bytes of memory are required to store the *Index Volume* (assuming that the number of quantized cells of all candidate cells is covered by 4 bytes). However, when we use the summed-area table, $100\times100\times100\times2$ bytes are required for the *Index Volume* and 100×4 bytes for the table (assuming that the number of quantized cells of the candidate cells per slice is less than 2 bytes). To determine the index of a candidate cell, this method performs one addition between the result of the *Index Volume* and that of the summed-area table. Therefore, it scarcely influences computation speed.

4.4. Structure of the PERM

Figure 4 shows an example of indexing the *Density Buffer, Gradient Buffer,* and *Sign Buffer* through the *Index Volume*. In the preprocessing step, after generating the *Index Volume* to store the indices of candidate cells, we determine the *Density Buffer*, *Gradient Buffer*, and *Sign Buffer*. In the rendering stage, when a ray locates a sample point, it reads the nearest quantized cell using *M_LUT*, and reads the *Index Volume* to index each buffer. Using these two values, it reads the density value, gradient vector, and sign field. These values are used to determine the opacity and color value.

5. Reducing the Number of Candidate Cells

When the number of voxels of which the density value is greater than t_d is numerous, a lot of memory is needed. When an accumulated opacity value becomes 1.0, ray traversal at the pixel is complete. Therefore, if we can generate the boundaries when the accumulated opacity value becomes 1.0, we reduce the memory space for storing the PERM. Although our approach detecting the boundary is similar to [12, 13, 14, 15], the extraction method is different from other methods.

To reduce the number of candidate cells, we consider two properties. Property (1) occurs when the fully opaque boundary generated in the previous step encloses the current cell. This means that a ray cannot come into the current cell since the accumulated opacity has already become 1.0. We do not consider the current cell even though it is not fully opaque. Figure 5(a) illustrates a 2D example of this property. The shaded circle represents an opaque voxel, the empty circle is a transparent one, and the shaded region depicts the boundary generated in the previous step. Although cells C_1 and C_2 are not fully opaque, the ray cannot enter C_1 and C_2 because the boundary generated in the previous step ($C_{i-3} \sim C_i$) enclosing C_1 and C_2 is fully opaque.

Even if property (1) is not satisfied, the accumulated opacity becomes 1.0 when translucent cells continuously occur (figure 5(b)). Rendering at each pixel can be terminated even when a fully opaque cell does not exist. This is property (2). To prevent excess generation, we define the iterative threshold t_e. When $t \geq t_e$ in the t-th boundary, the generation process is complete.

Using these properties, we can reduce the number of candidate cells. First, we peel the volume until an outermost (innermost) boundary remains. Next, we perform morphological thickening or thinning. If the current voxel satisfies property (1), the cell is skipped.

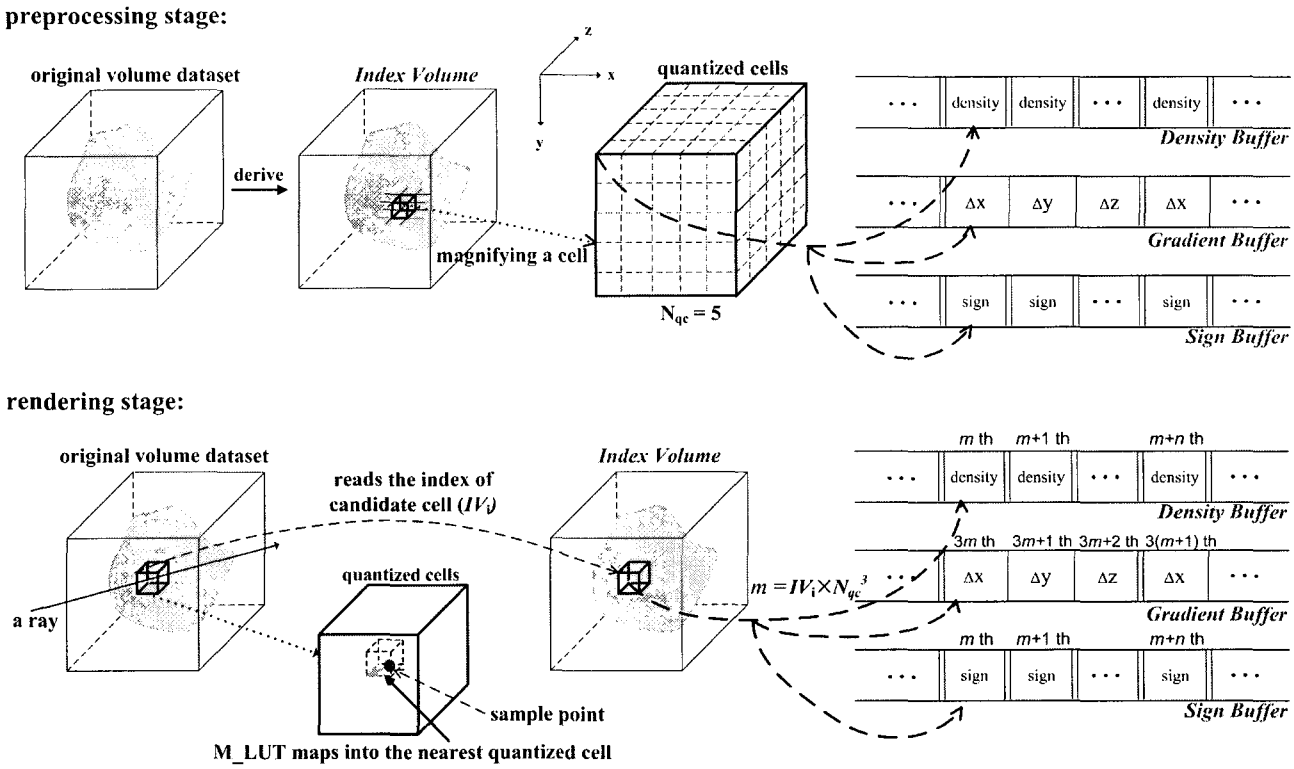

Figure 4: The data structure of the PERM. It is composed of *Index Volume*, *Density Buffer*, *Gradient Buffer*, and *Sign Buffer*.

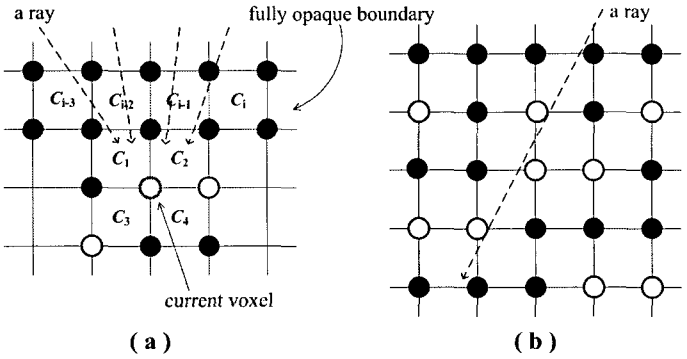

Figure 5: Since C_1 and C_2 are enclosed by the fully opaque boundary generated in the previous step (from C_{i-3} to C_i), a ray cannot come into C_1 and C_2. Accumulated opacity can become 1.0 when a translucent cell is repeated continuously.

The generated boundary can satisfy the closure condition. If the current voxel cannot fulfill the property, a ray might penetrate the boundary. These steps repeat until the property (2) is satisfied. Figure 6 shows an example to reduce the number of candidate cells. The total memory can obviously be reduced by using this method.

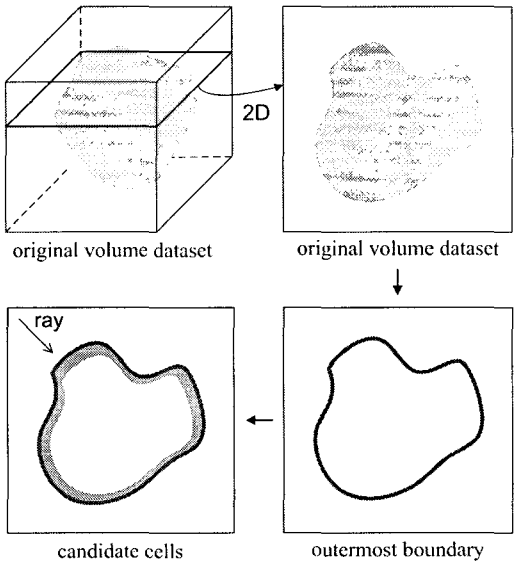

Figure 6: An example of reducing the number of candidate cells. The morphological thinning is performed from the outermost boundary.

Although this method can reduce the memory space, it can produce different results. If a cavity exists inside the object and the viewpoint locates inside, we perform morphological thickening from the

innermost boundary. Otherwise, when we observe the object from outside, we execute morphological thinning. Therefore, we should make two boundaries when a cavity exists. After generating the boundaries, we merge them. Figure 7 depicts these situations.

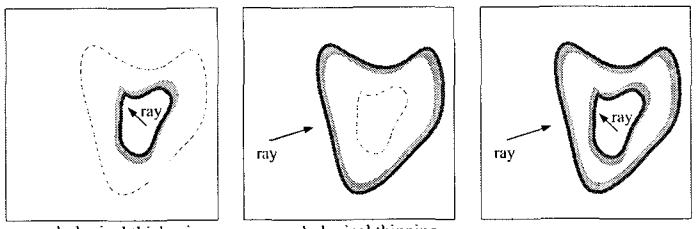

Figure 7: When the object has cavity regions, morphological thickening is performed from the innermost boundary and thinning from the outermost boundary. After generating these boundaries, we might merge them.

6. Unequal Quantization

Although our method accelerates rendering speed, much space is required to store the data structure. We propose a method applying unequal quantization for the density and gradient vector to solve this problem. That is, we set N_{qc} for the density and that for the gradient vector to be unequal.

There are two methods. The first sets the number of quantized cells for density (D_N_{qc}) to be large compared with that for the gradient vector (G_N_{qc}). This method does not deteriorate image quality because the discrepancy between the original sample point and the point taken by *M_LUT* is trivial. Of course, the lighting effect deteriorates since the number of quantized cells for the gradient vector is insufficient. On the other hand, if the number of quantized cells for the gradient vector is set to be large, image deterioration may occur because of the discrepancy between them. In addition, since calculation of the gradient vector is only possible by estimation, the former case shows an accurate result compared with the latter case.

When we use the PERM using unequal quantization, equation (2) transforms into equation (5). The *Index Volume* is identical. Equation (6) shows the memory requirement when we use the unequal quantization based method.

$$\begin{aligned} DB_l &= D_{tri} \\ GB_{3m} &= GX_{tri} \\ GB_{3m+1} &= GY_{tri} \\ GB_{3m+2} &= GZ_{tri} \\ SB_m &= SIGN \\ (\ where\ \ l &= IV_i \times (D\_N_{qc})^3,\ \ m = IV_i \times (G\_N_{qc})^3\ ) \end{aligned} \tag{5}$$

$$\begin{aligned} \varphi(PERM) &= \varphi(IV)+\varphi(DB)+\varphi(GB)+\varphi(SB) \\ &= S \times b_{iv} + (N_{cc} \times (D\_N_{qc})^3 \times b_{db}) + \\ &\quad (N_{cc} \times (G\_N_{qc})^3 \times b_{gb} \times 3) + (N_{cc} \times (G\_N_{qc})^3 \times b_{sb}) \end{aligned} \tag{6}$$

7. Experimental Results

All of the methods were implemented on a PC equipped with a single Pentium IV 3.06 GHz CPU and 1 GB main memory. The first dataset was a CT scan of a Boston teapot with a resolution of 256×256×178, the second dataset was a CT scan of a bonsai with a resolution of 256^3. The third dataset was an aneurysm of a human brain vessel with a resolution of 256^3, and the final dataset was a human tooth with a micro CT with a resolution of 256×256×161.

Table 1 shows the preprocessing time and required memory for different datasets when N_{qc} is 5. The time was about 30 seconds and required memory was 505 MB to store the PERM in the worst case. When we use the unequal quantization based method, even in the worst case, the time was 13 seconds with a required memory of 209 MB.

The rendering time to produce the final image when the images are projected to a perspective view is shown in figure 8, and for a parallel projection is shown in figure 9. All the results are for the mean computation time and include the space-leaping time. Each dataset provides the result when the image size is 128 x 128, 256 x 256, and 512 x 512, respectively. The rendering speed of our method is at least 200% faster than that of the conventional distance-map method. We use the Euclidean distance-map method as a space-leaping method [19]. Total rendering time is also at least 200% faster than the conventional method for parallel projection. However, total rendering speed is slow compared with the perspective method because advance from image space to volume space is required. In perspective projection, it is not necessary to skip over those regions since the center of projection is generally located inside the volume space.

Table 1: Preprocessing time and required memory to generate the PERM for different datasets. N_{qc} is 5.

dataset	N_{cc} (voxels)	candidate cell detect time (secs)	quantized cell generation time (secs)	preprocessing time (secs)	required memory (MB)
teapot	601,190	0.77	19.78	20.55	399
bonsai	754,117	0.84	29.02	29.86	505
aneurism	214,521	0.44	7.14	7.58	168
tooth	304,322	0.53	10.01	10.54	211

Table 2: Preprocessing time and required memory when using unequal quantization based method ($D\_N_{qc}=5$, $G\_N_{qc}=3$).

dataset	N_{cc} (voxels)	candidate cell detect time (secs)	quantized cell generation time (secs)	preprocessing time (secs)	required memory (MB)
teapot	601,190	0.77	8.37	9.14	163
bonsai	754,117	0.84	11.82	12.66	209
aneurism	214,521	0.44	3.06	3.50	84
tooth	304,322	0.53	4.33	4.86	92

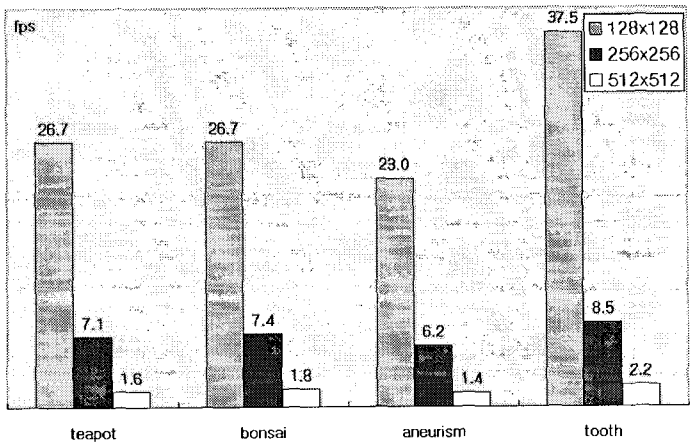

Figure 8: Performance of our approach for different datasets. We use Euclidean distance-map as a space-leaping method. All results are rendered to a perspective projection.

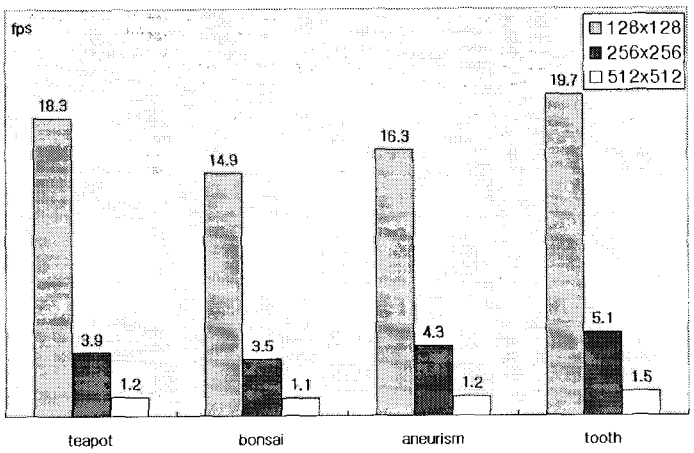

Figure 9: Performance of our approach when the results are projected to a parallel view.

Figure 10 shows the image quality without change of viewpoint and viewing direction. The images in figure 10(a) are the result generated using the conventional volume ray casting method. Images in figures 10(b), (c), and (f) are results generated by our algorithm when N_{qc} is 10, 5, and 3 respectively. We can see visual artifacts according to the decrease in N_{qc}. The images when D_N_{qc} is 3 and G_N_{qc} is 5 are depicted in figure 10(d). Figure 10(e) shows the results when D_N_{qc} is 5 and G_N_{qc} is 3. Visual artifacts exist when D_N_{qc} is lower than G_N_{qc}.

Figure 11 shows the rendering images for different datasets projected to a parallel view and a perspective view when D_N_{qc} is 5 and G_N_{qc} is 3. Image size is 256×256 pixels.

8. Conclusion

The most important issue in volume visualization is to produce high quality images in real time. We propose the volume ray casting method to reduce rendering time compared with conventional methods in any situation. Using the PERM, we can render in real-time on a consumer PC platform without time-consuming calculations such as trilinear interpolation and gradient estimation. The rendering speed of our method does not affect volume size, although that of proposed methods such as 3D texture mapping is dependent on volume datasets. Our experimental results show that the method normally produces high-quality images and requires less rendering time. Future work will be focused on reducing memory space and preprocessing time.

References

[1] LEVOY, M. 1988. Display of Surfaces from Volume Data. IEEE Computer Graphics and Applications, 5, 4, 29–37.

[2] YAGEL, R. and KAUFMAN, A. 1992. Template-based Volume Viewing. In Proceedings of ACM EUROGRAPHICS 1992, 11, 3, 153–167.

[3] LACROUTE, P. and LEVOY, M. 1994. Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation. In Proceedings of ACM SIGGRAPH 1994, 451-457.

[4] WESTERMANN, R. and ERTL, T. 1998. Efficiently Using Graphics Hardware in Volume Rendering Applications. In Proceedings of ACM SIGGRAPH 1998, 169-177.

[5] REZK-SALAMA, C., ENGEL, K., BAUER, M., GREINER, G. and ERTL, T. 2000. Interactive Volume on Standard PC Graphics Hardware Using Multi-textures and Multi-stage Rasterization. In Proceedings of ACM SIGGRAPH 2000, 109-118.

[6] CABRAL, B., CAM, N. and FORAN, J. 1994. Accelerated Volume Rendering and Tomographic Reconstruction using Texture Mapping Hardware. In Symposium on Volume Visualization 1994, 91-98.

[7] LINDHOLM, E., KLIGARD, M.J. and MORETON, H. 2001. A User-Programmable Vertex Engine. In Proceedings of ACM SIGGRAPH 2001, 149-188.

[8] PURCELL, T.J., BUCK, I., MARK, W.R. and HANRAHAN, P. 2002. Ray Tracing on Programmable Graphics Hardware. In Proceedings of ACM SIGGRAPH 2002, 703-712.

[9] ENGEL, K., KRAUS, M. and ERTL, T. 2001. High-quality Pre-integrated Volume Rendering Using Hardware-Accelerated Pixel Shading. In Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware, 9-16.

[10] LUM, E., WILSON, B. and MA, K. 2004. High-Quality Lighting and Efficient Pre-Integration for Volume Rendering. In Proceedings of the Joint Eurographics-IEEE TVCG Symposium on Visualization 2004.

[11] ROETTGER, S., GUTHE, S., WEISKOPF, D., ERTL, T. and STRASSER, W. 2003. Smart Hardware-Accelerated Volume Rendering. In Proceedings of EUROGRAPHICS/IEEE TCVG Symposium on Visualization 2003, 231-238.

[12] UDUPA, J.K. and ODHNER, D. 1993. Shell Rendering. IEEE Computer Graphics and Applications, 13, 6, 58-67.

[13] CARNIELLI, G.P., FALCAO, A.X. and UDUPA, J.K. 1999. Fast Digital Perspective Shell Rendering. In Proceedings of the XII Brazilian Symposium on Computer Graphics and Image Processing, 105-111.

[14] SOBIERARJSKI. L. and AVILA, R. 1995. A Hardware Acceleration Method for Volumetric Ray Tracing, In Proceedings of IEEE Visualization 1995, 27-34.

[15] WAN, M., BRYSON, S. and KAUFMAN, A. 1998. Boundary Cell-Based Acceleration for Volume Ray casting. Computer&Graphics, 22, 6. 715–721.

[16] HILL, S. Tri-linear interpolation. In P.S. Heckbert, Graphics Gems IV. 521–525. Academic Press Professional, San Diego, CA, 1994.

[17] YAGEL, R., COHEN, D. and KAUFMAN, A. 1992. Normal Estimation in 3D Discrete Space. The Visual Computer, 6, 278-291.

[18] LEVOY, M. 1990. Efficient Ray Tracing of Volume Data. ACM Transactions on Graphics, 9, 245–261.

[19] SAITO, T. and TORIWAKI, J.I. 1994. New Algorithms for Euclidean Distance Transformations of an N-Dimensional Digitized Picture with Applications. Pattern Recognition, 27, 11, 1551-1565.

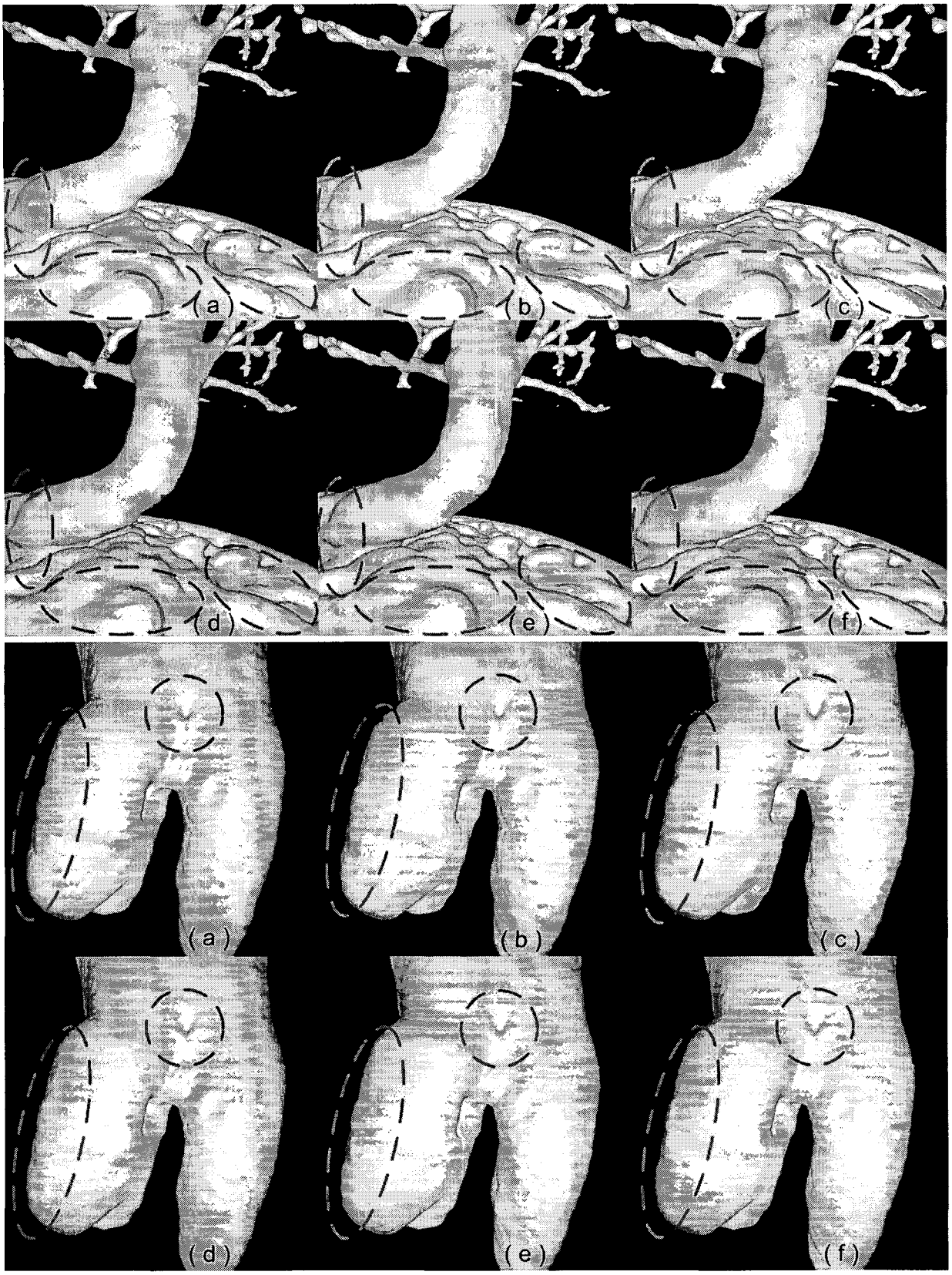

Figure 11: Comparison of image quality: (a) conventional volume ray casting (b) $N_{qc}=10$ (c) $N_{qc}=5$ (d) $D\_N_{qc}=3$, $G\_N_{qc}=5$ (e) $D\_N_{qc}=5$, $G\_N_{qc}=3$ (f) $N_{qc}=3$. The dataset of the bonsai is at the first and second row, and that of the tooth is at the third and fourth row.

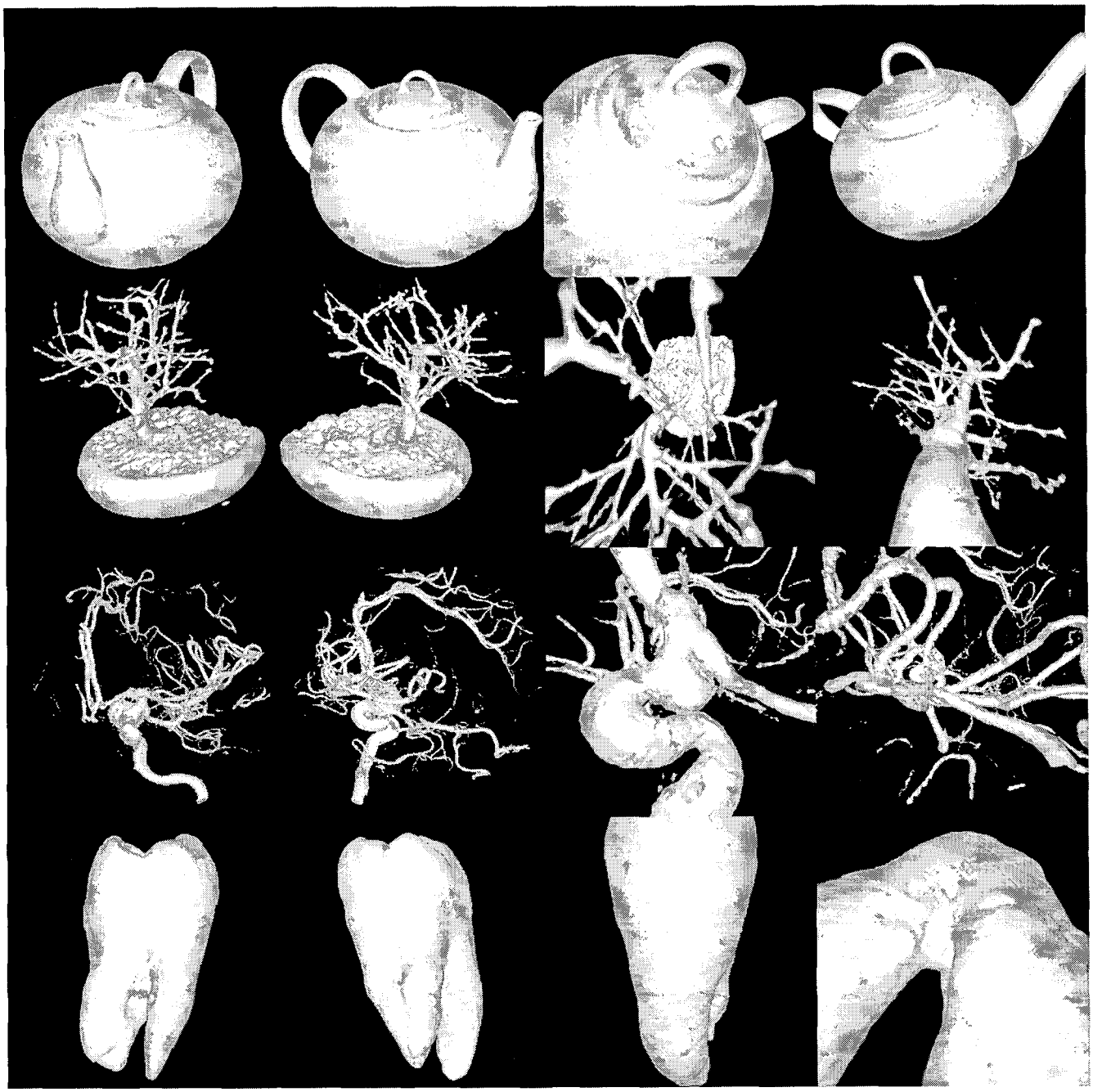

Figure 12: Rendering results in different areas: Images of the first and second column are obtained with parallel projection. Images of the third and fourth column are projected to perspective view.