PERGAMON

Computerized Medical Imaging and Graphics 23 (1999) 193–200

Computerized Medical Imaging and Graphics

www.elsevier.com/locate/compmedimag

Interactive voxel surface rendering in medical applications

T.-Y. Lee[1,*], T.-L. Weng, C.-H. Lin, Y.-N. Sun

Visual System Laboratory, Department of Computer Science and Information Engineering, National Cheng Kung University, Tainan, Taiwan, ROC

Received 19 March 1999; accepted 19 May 1999

Abstract

Semi-boundary (SB) data structure is a compact voxel surface representation of the structure from the medical images. It represents only the boundary of the extracted structure and only an opaque object boundary involved in a 3D dataset can be visualized. Its computational complexity is in proportion to the number of SB voxels. In this paper, we propose schemes to reduce the number of projections in two ways. First, in conjunction with neighboring code, we exploit a set of visibility tables to cull some of the invisible SB voxels. Second, we exploit three pass rotations and an incremental approach to quickly determine the projection position for each SB voxel during rendering. With these two combinations, we significantly improve SB rendering performance. As a result, we can achieve an interactive rendering speed on general purpose workstations for our medical applications. © 1999 Elsevier Science Ltd. All rights reserved.

Keywords: Semi-boundary; Neighboring code; Visibility look-up tables; Three-pass rotations; Incremental approach

1. Introduction

Volume data are available from many kinds of sources, for example, scanned by MRI (Magnetic Resonance Imaging) or CT (Computed Tomography), or simulated by CFD (Computational Fluid Dynamics) programs. Visualization is a powerful technique to enable us to view 3D structure from these images. There are two main approaches in visualization: surface and volume rendering. The former represents the 3D structure of images in various forms such as voxel, surface, polygon and so forth. On the other hand, the latter does not compute an explicit model of the structure but is able to directly view volume as a semi-transparent cloudy material. Generally, volume rendering techniques are both compute and storage intensive while they produce high quality images. In the past, there has been considerable work done in overcoming these two issues [1–4].

Udupa et al. invented a semi-boundary (SB) data structure that enables interactive rendering and flexible manipulations [2]. The SB data structure only includes those voxels on the boundary of objects. So, it saves space. In contrast to the commonly used ray-casting paradigm, it uses voxel projection during rendering. Therefore, it eliminates the need for rendering-time interpolation and thus it is fast. This approach is quite similar to splatting [5]. However, there is no voxel composite in SB approach. In words, it cannot allow a voxel with transparent density. Additionally, the SB only performs projection for voxels on the boundary of objects instead of all voxels (i.e. spatting). Therefore, the SB is reasonably faster than splatting. But, the SB splats voxels with no filter kernel and thus reduces image quality. In the past, Hanrahan and Laur [3], and Muller and Yagel [4] proposed schemes to reduce the number of voxels to be splatted. Later, Udupa et al. modified SB structure and termed it as a new name, *Shell* [6]. The shell allows voxel composite and thus transparent object rendition is allowed.

The SB rendering performs voxel projection and uses Z-buffer to resolve visibility. Its computational complexity is in proportion to the number of SB voxels. In this paper, we propose schemes to improve the original SB rendering in the following respects. We exploit the use of neighboring code in conjunction with the proposed visibility tables to remove some of the invisible voxels, thereby reducing the number of voxels drawn to the screen. The three pass rotations and an incremental algorithm are presented to quickly determine the projection position for each voxel during rendering. In this paper, the SB voxels are projected in the *far-to-near* order eliminating the need for Z-buffer. The SB has an inherent aliasing problem (i.e. gaps or holes). To resolve it, we use a low-cost 2D filter to post-process the rendered image. Finally, a simple *matting* scheme is

* Corresponding author. Tel.: 886-6-2757575. ext. 62531; fax: 886-6-2747076.

E-mail address: tonylee@mail.ncku.edu.tw (T.-Y. Lee)

[1] http://vision.csie.ncku.edu.tw/~tlee

PII: S0895-6111(99)00015-4

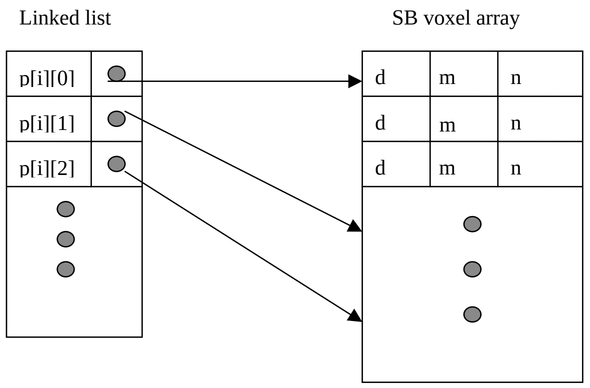


Fig. 1. SB (Semi-Boundary) data structure.

exploited to merge multiple objects and thus the proposed design also can be used to apply different levels of translucency to different objects.

In Section 2, we describe data structure to store SB voxels and introduce visibility tables. The schemes proposed to determine SB projections quickly are presented in Section 3. Our experimental results will be discussed in Section 4. Finally, some concluding remark and future work is given in Section 5.

2. Data structure and visibility tables

For a $N \times N \times N$ volume data, $V[0\ldots N-1, 0\ldots N-1, 0\ldots N-1]$ let p and q be two distinct voxels with coordinates (p_x, p_y, p_z) and (q_x, q_y, q_z), respectively. Then, p and q are 6-*adjacent* if $|p_x - q_x| + |p_y - q_y| + |p_z - q_z| = 1$. Selecting a binary segmentation function (a threshold, for example), each voxel is classified either as a 0-*voxel* or as a 1-*voxel*. The union (denoted as C) of all 1-*voxels* represents the structure we wish to visualize, manipulate, and analyze. Instead of handling all 1-voxels, Udupa et al. introduced a semi-boundary (SB) voxels defined as follows. In this method, the neighbors of a voxel are all its 6-*adjacent* neighbors. For a 1-voxel, say c_i, if it has at least a 0-*voxel* neighbor, then c_i will be termed as a SB voxel. The union of all SB voxels is denoted as S. It can be shown that S and C is equivalent [2], the advantage being that typically S can be represented more compactly (i.e. save space) in the computer than C.

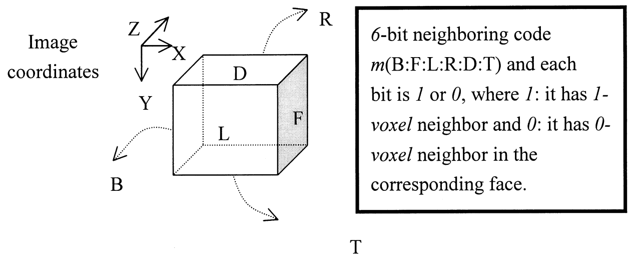


Image coordinates
Z
X
Y
R
D
L
F
B
T
6-bit neighboring code m(B:F:L:R:D:T) and each bit is 1 or 0, where 1: it has 1-voxel neighbor and 0: it has 0-voxel neighbor in the corresponding face.

Fig. 2. The six faces of each SB node and image coordinate system.

Table 1
One visibility look-up table used for $0 < \theta < 90°$ and $0 \le \phi \le 360°$. (*Note*: D: down; T: top; L: left; R: right; B: back; F: front)

ϕ (degree)	Visible face
0	L,B
0 ~ 90	L,T,B
90	T
90 ~ 180	T,F,R
180	F,R
180 ~ 270	D,F,R
270	D
270 ~ 360	D,L,B

To cull invisible SB voxels, Udupa et al. proposed a 6-bit number that encode segmented information (i.e. 1: 1-*voxel* or 0: 0-*voxel*) for all its 6-*adjacent* neighbors. This 6-*bit* number is called the neighboring code of a SB voxel. Additionally, in our implementation, we pre-compute the normal of each SB voxel and encode the normal as an index, n, to the normal vector look-up table [7]. In this manner, we can save more space to store SB voxels.

Conceptually, all SB voxels are organized as a two-dimensional ($M \times N$) linked lists. In words, this structure represents that there are M by N virtual rays (originating from the YZ plane) cast along the X-axis. For each virtual ray (linked list), it is a collection of SB nodes located on this ray. In this structure, the $P[i][j]$ is a dummy node and termed as the origin of each ray (i.e. linked-list). It records the number of SB nodes located on this list, and it contains a pointer to the first SB node along the ray. Each SB node contains the following information: d is the *distance* from the origin $P[i][j]$, m is its neighboring code, n is an index to the normal vector table, and a *link* is a pointer to the next SB node in the same linked list. In implementation, we would like to build this data structure with an array of pointers and an array of SB voxels as shown in Fig. 1. In this structure, each element of the pointer array, say $P[i][j]$, contains a pointer to the first SB node stored in the SB voxel array, and the number of SB voxels along the same virtual ray. In this manner, the SB nodes are stored in continuous memory locations and then can be drawn in storage order during rendering. Therefore, it potentially reduces memory access overhead. To the contrast of this approach, the main disadvantages of ray casters is that they do not access the volume in storage order since viewing rays may traverse the volume in an arbitrary order.

During rendering, a SB node, say c_i, is potentially visible if any of its six facets facing the viewing direction are not blocked by 1-*voxels* that are 6-*adajacent* to c_i. Recall that neighboring code is a 6-*bit* number to encode segmented information (i.e. 1: 1-*voxel* or 0: 0-*voxel*) for all its 6-*adjacent* neighbors. To quickly determine whether a SB voxel is potentially visible, we create a set of visibility look-up tables in conjunction with the neighboring code. The visibility of a SB node is tested as follows. The six faces of a SB node and the image coordinate system are illustrated in Fig.

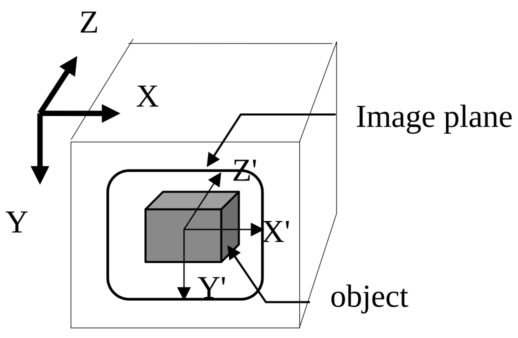


Fig. 3. If $\gamma = 0$, no rotation is required for the image plane.

2. For example, the down face is termed as D, and the top face is termed as T, respectively. The image plane is fixed at the X–Y plane. Therefore, to achieve a given viewing orientation, we can first rotate SB nodes along the X- and the Y-axis in turn, project SB nodes and then rotate the image plane along the Z-axis. In other words, we do not need to consider rotation along the Z-axis during visibility test. One of the visibility look-up tables created is shown in Table 1. This table will be used when the rotation angle θ about the Y-axis (i.e. clockwise) is between 0 and 90°. In this case ($0 < \theta < 90°$), there are eight combinations (ϕ, rotation angle about the X-axis in a clockwise direction) for visibility test. For example, if $0 < \phi < 90°$, the visible faces of a SB node, say c_i, are L, T and B, respectively. If the corresponding bits of c_i's 6-*bit* number in L, T, and B are all 1s, then c_i is determined to be invisible from the current viewing orientation. Similarly, the other visibility tables for rotation about the Y-axis can be created in the same way for $90 \le \theta \le 360$. In total, there are eight look-up tables required. Notice that in their original SB paper [2], Udupa et al. mentioned look-up tables, too: however, there are no details about them.

3. Quick determination of SB projections

3.1. Three pass rotations

Like the original SB paper [2], we perform voxel orthographic projection to render SB nodes in this paper. The coordinate systems used in this paper are shown in Fig. 3. The axes of object (volume) coordinates are labeled X, Y and Z. The axes of image coordinates are labeled X', Y' and Z', and its origin is located at the center of the volume coordinate system. To render an image for a given view, we can achieve it as follows. First, we rotate object (i.e. SB voxels) about the X'-axis by α degree and then about the Y'-axis by β degree. Second, we project transformed object on the image plane (i.e. the X'–Y' plane). Finally, we rotate image plane about the Z'-axis by γ degree to obtain the correct rendered image. Recall that each linked list of SB nodes is thought as a virtual ray cast along the X-axis. In words, all lists are parallel lines along the X-axis. For the first rotation (i.e. about the X'-axis), all parallel lines will be still parallel and also preserve their length after the projection. However, after the second rotation (i.e. about the Y'-axis), they do not preserve length after projection, and their length will be scaled by a factor of $\cos(\beta)$ as shown in Fig. 4.

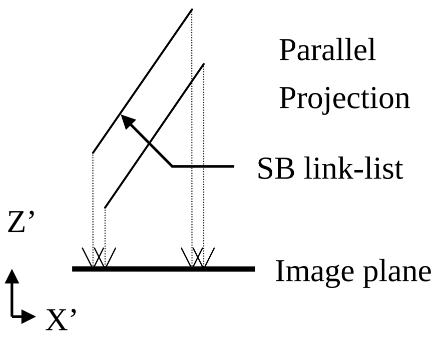


Fig. 4. If $\beta \neq 0$ the length of a link-list is scaled by $\cos\beta$.

When the angle γ is not zero, we perform the third rotation of the image plane about the Z'-axis to obtain a correct result (illustrated in Fig. 5). Based upon the above rationality, it is only required to project the origins (i.e. $P[i][j]$s) of all lists rather than all SB voxels. For the remaining SB nodes on each list, we multiply each node's *d value* (i.e. distance from the origin by a factor of $\cos(\beta)$ plus the projected location of the origin to obtain their exact projected location. In contrast, the original SB algorithm must compute parallel projection for all SB nodes. Next, we will show an incremental approach to further reduce the number of origin projections.

3.2. Incremental calculation of projection

Three coordinates of a SB node, (X, Y, Z) are transformed into the image system coordinates (X', Y', Z') by a rotation matrix M. The $m_1 \sim m_9$ are elements of the rotation matrix M:

$$\begin{bmatrix} X' \\ Y' \\ Z' \end{bmatrix} = \begin{bmatrix} m_1 & m_2 & m_3 \\ m_2 & m_5 & m_6 \\ m_7 & m_8 & m_9 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{1}$$

The SB nodes are virtually organized as parallel rays originating from the YZ plane. The X coordinate of each origin $P[i][j]$ is zero, so (1) can be written as (2) for each origin

$$\begin{bmatrix} X' \\ Y' \\ Z' \end{bmatrix} = \begin{bmatrix} m_1 & m_2 & m_3 \\ m_2 & m_5 & m_6 \\ m_7 & m_8 & m_9 \end{bmatrix} \begin{bmatrix} 0 \\ Y \\ Z \end{bmatrix} \tag{2}$$

$$\begin{aligned} X' &= m_2 Y + m_3 Z \\ Y' &= m_5 Y + m_6 Z. \end{aligned} \tag{3}$$

For the origins of two nearby parallel lists, we know that

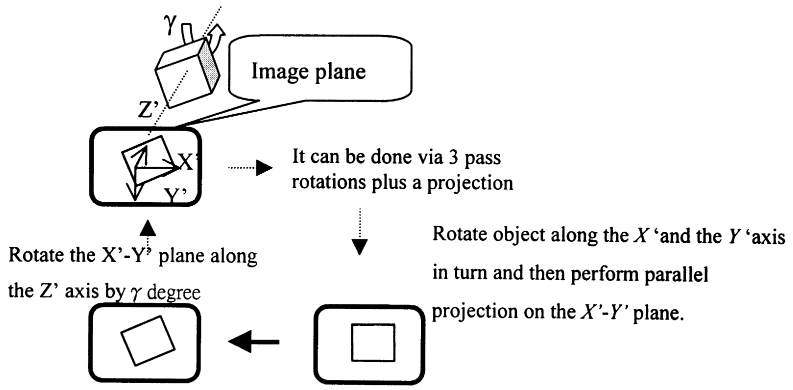

γ
Image plane
Z'
X'
Y'
It can be done via 3 pass rotations plus a projection
Rotate object along the X' and the Y' axis in turn and then perform parallel projection on the X'-Y' plane.
Rotate the X'-Y' plane along the Z' axis by γ degree

Fig. 5. If $\gamma \neq 0$, then rotate the image plane after projection.

$\Delta Y \neq \Delta Z$ and their value is either 0 or 1. This implies that $\Delta X'$ is either m_2 or m_3, and $\Delta Y'$ is either m_5 or m_6. During rendering, we can select a list, say $S1$, and compute the projection of its origin using Eq. (1). On the basis of $S1$, the origin of the neighboring list $S2$ can be simply calculated by two additions as shown in Fig. 6. In this manner, the projection of the origin for $S3$ can be decided by $S2$, incrementally.

3.3. Visibility solving

The Z-buffer can be exploited to resolve visibility. For all SB voxels, we can draw them on the screen in the *far-to near* order and thus eliminate the need for a Z-buffer. Recall that we perform two pass rotations (i.e. α and β about the X'- and the Y'-axis of image coordinate system) before projection. After these two rotations, the local object coordinate system, labeled by x, y, and z-axis, will be transformed. Then, the *far-to-near* project order can be easily determined by finding an axis, say P, among the x, y and z-axes, and P is most parallel to the projection direction $-Z'$. For example, in Fig. 7, $\alpha = 90$ and $\beta = 0$, then $+z$ and $-Z'$ directions are most parallel. Therefore, we will project SB nodes plane by plane; in words, start from plane at $z = 0$ to plane at maximum $+z$ to achieve the *far-to-near* project order.

4. Experimental results and discussions

We have implemented both SB and the proposed algorithms on a SUN SPARC-20 workstation to evaluate and analyze their rendering performance. The test set is a $512 \times 512 \times 245$ pelvis CT data and the rendered image is at $512 \times$

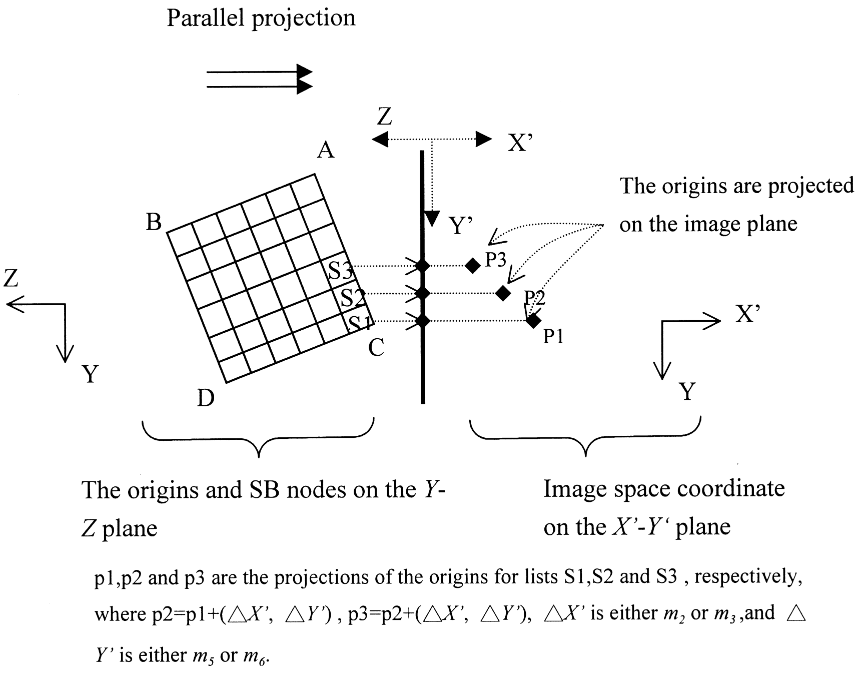

Parallel projection
Z
X'
A
B
Y'
The origins are projected on the image plane
S3
S2
S1
P3
P2
P1
C
D
Z
Y
X'
Y
The origins and SB nodes on the Y-Z plane
Image space coordinate on the X'-Y' plane
p1,p2 and p3 are the projections of the origins for lists S1,S2 and S3, respectively, where p2=p1+(△X', △Y'), p3=p2+(△X', △Y'), △X' is either m2 or m3, and △Y' is either m5 or m6.

Fig. 6. Decide the projection of the origin in an incremental manner.

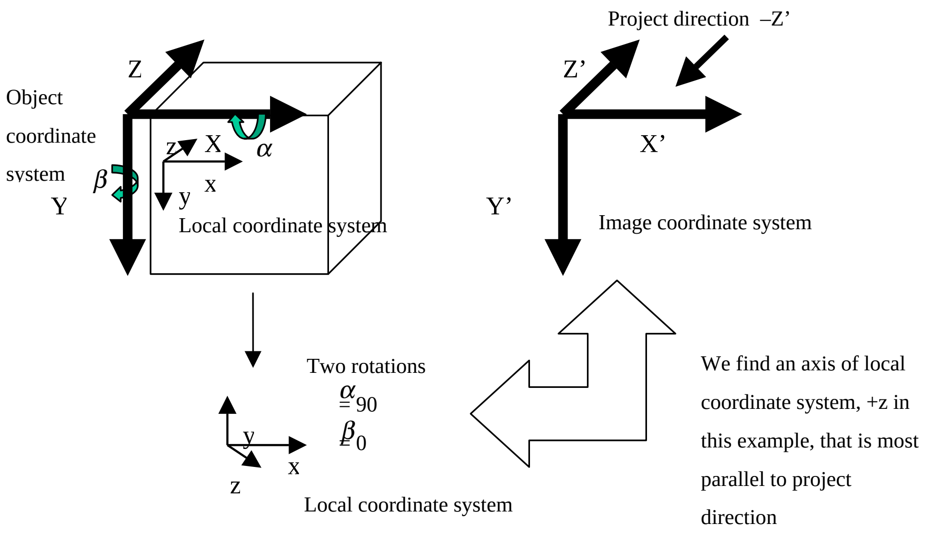


Fig. 7. An example of determining the *far-to-near* project order.

512 resolution. The various overheads spent in both algorithms are classified as follows: (1) *looping*: overheads spent on data access overheads such as advancing pointers and traveling SB voxel array; (2) *culling*: overheads spent on visibility culling; (3) *projection*: overheads spent on drawing SB nodes on the screen and (4) *miscellaneous cost*: such as visibility solving, shading and so forth.

In the first experiment, the results are obtained from rendering 180 consecutive frames, rotating object about the *Y*-axis. The neighboring code in conjunction with look-up tables and the pre-computed normals can improve rendering performance. Therefore, to fairly compare the proposed algorithms with the original SB scheme, both methods will include these options in this study. On the average, the rendering time per frame taken by the SB and the proposed method are 1567 and 716 ms, respectively. The proposed method is faster than the original SB by a factor of 2.19. We can certainly achieve much better improvement in the case of the SB scheme without visibility culling. We should point out that there is approximately a half number of SB voxels culled in our study. To further understand the performance difference, a breakdown of the total execution time for both methods is shown in Table 2. This table shows the exact timings and the percentages for various overheads. For the original SB, the major cost is due to projection (about 60%). By contrast, this cost is reduced significantly in the proposed algorithms (about 5.5 times faster). For miscellaneous cost, since there is no Z-buffer in the proposed scheme, it is slightly faster. In future, we would like to shorten timing in looping for better improvement. Finally, we show a rendered image of the CT pelvis data in Fig. 8.

In contrast to ray casting, the proposed method is view independent and its computational complexity is not in proportion to the size of volume, but to the number of total SB voxels. Fig. 9 shows a rendered image for another case of study. In this study, on the average, it requires 412 ms to render an image. Fig. 10(a) and (b) both show that there is no jump in rendering time when object is rotated about a fixed axis in both studies. In other words, the rendering is view independent versus viewing angle. We should point out that, on the average, there are 2.5×10^5 visible SB nodes in (a) but approximately 1.4×10^5 in (b). Therefore, the rendering cost of (a) is approximately larger than that of (b) by a factor of two.

The facility to display multiple objects at the same frame is a common routine in visualization. For example, we render a MR head and a SPECT tumor simultaneously to unveil their spatial relationship in pre-surgical planning. We

Table 2
Detailed timing breakdowns for both schemes

Method	Item	Looping	Culling	Projection	Others
SB scheme	Time (ms)	324	68	947	228
	Percentage	21%	4%	60%	15%
Our scheme	Time(ms)	324	68	172	152
	Percentage	21%	10%	24%	21%

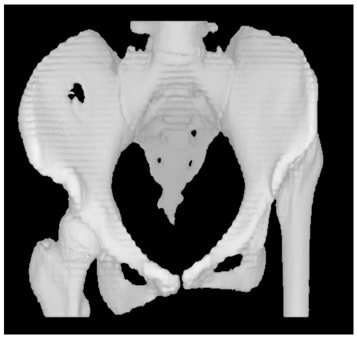


Fig. 8. A rendered image of the pelvis data ($512 \times 512 \times 245$).

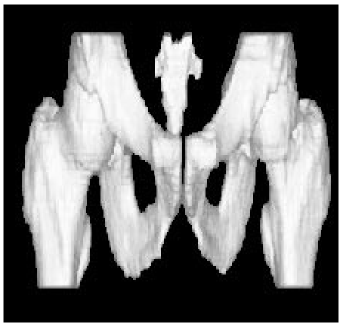
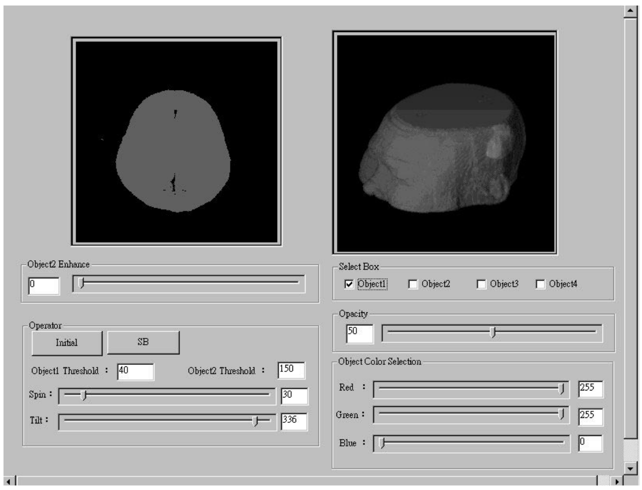


Fig. 9. Another study of rendering pelvis data ($320 \times 270 \times 167$).



Object2 Enhance: 0

Select Box: [✓ Object1] Object2 Object3 Object4

Opacity: 50

Operator: Initial SB

Object1 Threshold : 40 Object2 Threshold : 150

Spin : 30

Tilt : 336

Object Color Selection

Red : 255

Green : 255

Blue : 0

Fig. 11. A snapshot of the *matting* GUI; many parameters can be manipulated. In this example, a MR head and a SPECT tumor are displayed simultaneously.

exploit our packages to segment and register both data sets. Therefore, there are two sets of SB voxels created and registered. Then, we generate an image for each set and merge two images by a simple *matting* method as follows. The color $C = [\mathrm{R{\cdot}G{\cdot}B}\alpha]$ at each point of a desired composite will be a combination of the color C_f of the foreground and color C_b of the background at the corresponding points. The combination is simply calculated by $C = C_\mathrm{f} + (1 - \alpha_\mathrm{f})C_\mathrm{b}$. Fig. 11 shows a snapshot of our matting GUI where the user can interactively manipulate several parameters such as color and opacity (i.e. α). In this example, the MRI head is chosen as the background image.

The image quality is essential to good visualization. In the proposed method, there are two main aliasing effects. The first inherently comes from the SB method; it creates gaps and holes. The second is caused from the final image rotation in the proposed algorithms. To reduce these defects, we exploit a low-cost median filter on the final image. Fig. 12 shows the rendered results with various sizes of filters. We can see there are many small gaps or holes scattered over the image for the non-filter case. Generally, we exploit a 3×3 median filter in experiments, and we can obtain acceptable image quality without sacrifice of rendering performance. The larger filter kernel can avoid gaps easily, but excessive overlap among neighboring pixels will blur the image. In contrast to this approach, the splatting algorithm computes the contribution of a voxel to the image by convolving the voxel with a filter. The splatting computes filter convolution for all voxels. However, our approach only computes filter convolution on those pixels drawn by visible SB voxels (i.e. it is cheaper).

Finally, we show an example of exploiting the proposed scheme in pre-surgical planning. In this example, we demonstrate a surgical simulation of the Chiari osteomy as shown in Fig. 13. First, the object would be rotated to the desired orientation so that the cutting plane is perpendicular to the viewing plane. Next, a straight line inclined about 15° above the horizontal line and at a tangent to the top of the femoral head is drawn to determine the location and the orientation of the cutting plane, as shown in Fig. 13 (left). Finally, the upper coxal bone is moved outward along the plane with suitable offset to increase the cover area ratio of the acetabulum, as shown in Fig. 13 (middle). The cut and moved object is displayed without the femoral bone (as shown in Fig. 13 (right)) to illustrate the final operation of the Chiari osteomy. Both rendered image and three orthogonal views of volume data are used to demonstrate the final results.

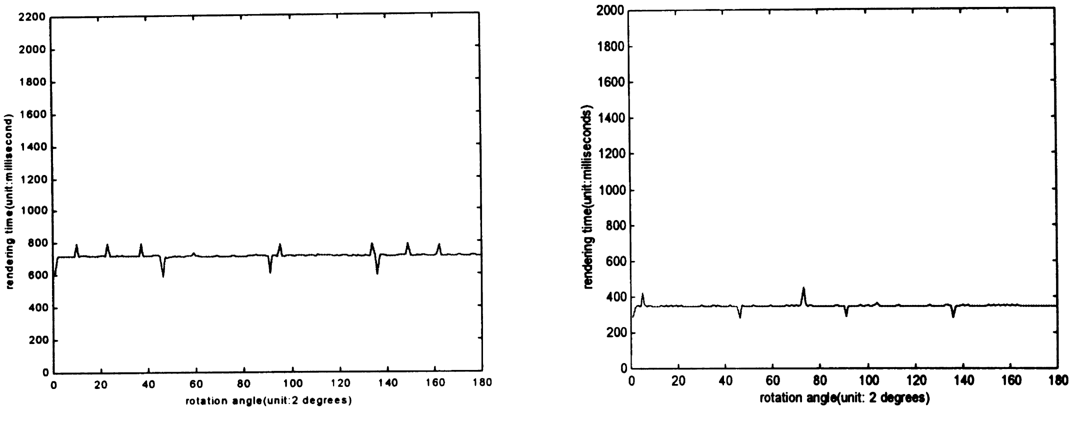


Fig. 10. Plot of rendering time versus viewing angle for both data sets: (a) left: $512 \times 512 \times 245$ and (b) right: $320 \times 270 \times 167$.

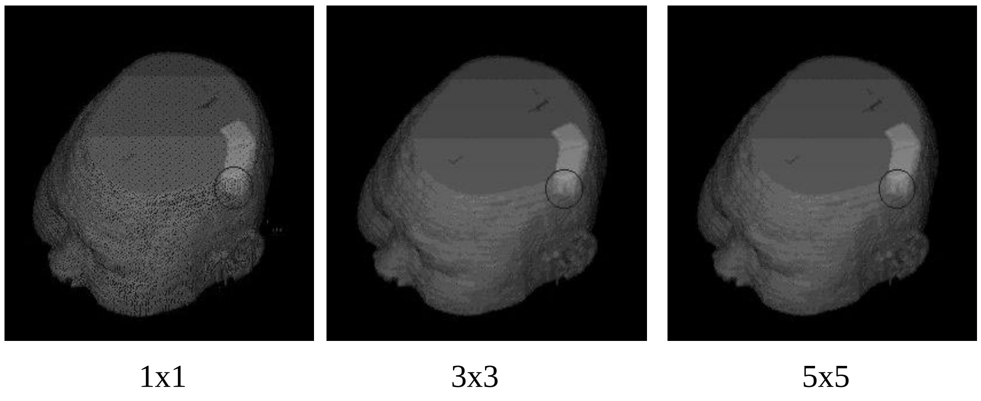


Fig. 12. Rendered results by convolving the pixel with various size of the median filter. In case where no filter is used (i.e. 1×1), there are many holes or gaps scattered over the image.

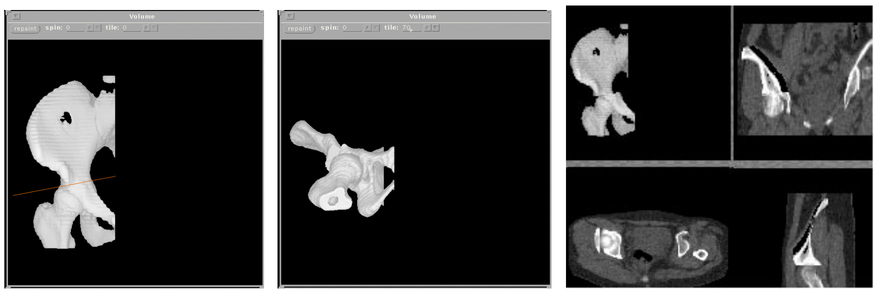


Fig. 13. An example of Chiari osteomy and these operations can be manipulated in an interactive manner.

5. Summary

In this paper, we present an interactive voxel surface rendering method. We exploit the proposed algorithms in medical applications. In this method, the object is represented by the SB data structure. The cost of drawing the SB voxels on the screen is in proportion to the number of the SB projections. To reduce this number, we propose three pass rotations and an incremental approach to quickly determine the projection. Moreover, the performance can be further improved by visibility culling. For this purpose, we exploit neighboring code in conjunction with a set of look-up tables. The experimental results show the proposed algorithms perform faster than the original SB method. We found that there is no jump in our rendering time versus viewing angle. Furthermore, we discuss the problems in multiple-object display and aliasing. We provide solutions to them and illustrate real examples. Finally, we demonstrate a surgical simulation in Chiari osteomy. The proposed algorithms can collaborate well with other manipulative operations such as cutting in an interactive manner. Several studies are to be carried out in the near future. We plan to enhance our scheme with the morphing technique to perform surgical simulation for the pelvis system. Some user-friendly manipulative tools will be developed. Additionally, we are collaborating with medical doctors in the hospital of National Cheng-Kung University to design a computer-aided surgical system that can support pre-surgical planning, surgical simulation, and provide both quantitative and qualitative knowledge prior to the actual surgical procedure.

Acknowledgements

This project is supported by NSC-88-2213-E-006-012 and NSC-88-2213-E-006-037, National Science Council, Taiwan, Republic of China.

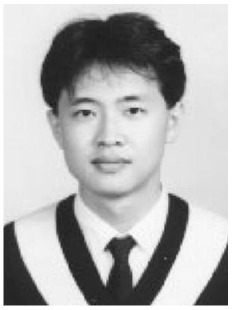
References

[1] Lacroute PG. Fast volume rendering using a shear-warp factorization of the viewing transformation, Technical Report: CSL-TR-95-678, Departments of Electrical Engineering and Computer Science, Stanford University, September, 1995.
[2] Udupa JK, Odhner D. Fast visualization, manipulation, and analysis of binary volumetric objects. IEEE Computer Graphics and Applications 1991;November:53–62.
[3] Hanrahan P, Laur D. Hierarchical splatting: a progressive refinement algorithm for volume rendering, Proceedings of SIGGRAPH'91.
[4] Muller K, Yagel R. Fast perspective volume rendering with splatting by utilizing a ray-driven approach, Proceedings of Visualization'96, San Francisco, CA, September, 1996. pp. 65–72.
[5] Westover L. Footprint evaluation for volume rendering, Computer Graphics (SGIGRAPH'90 Proceedings), Dallas, vol. 24, pp. 367–376.

[6] Udupa JK, Odhner D. Shell rendering. IEEE Computer Graphics and Applications 1993;November:58–67.
[7] Van Gelder K. Direct volume rendering with shading via three-dimensional texture, 1996 Symposium on Volume Visualization, pp. 23–30.

Tong-Yee Lee was born in Tainan county, Taiwan, Republic of China, in 1966. He received his B.S. in computer engineering from Tatung Institute of Technology in Taipei, Taiwan, in 1988, his M.S. in computer engineering from National Taiwan University in 1990, and his PhD in computer engineering from Washington State University, Pullman, in May 1995. Now, he is an Assistant Professor in the Department of Computer Science and Information Engineering at National Cheng-Kung University in Tainan, Taiwan, Republic of China. He was with WSU as a Visiting Research Professor at School of EE/CS during 1996 summer. He has been working on parallel rendering and computer graphics since 1992, and has published more than 60 technical papers in refereed journals and conferences. His current research interests include parallel rendering design, computer graphics, visualization, virtual reality, surgical simulation, distributed and collaborative virtual environment, parallel processing and heterogeneous computing.

Tzu-Lun Weng was born in Taiwan, ROC in 1965. He received the Master degree from National Central University in 1988. During 1988–1995, he worked in CSIST (Chung-Shan Institute of Science and Technology). He is now working toward to the PhD degree in the Department of Computer Science and Information Engineering, National Cheng-Kung University from 1995. His current research interests include computer vision, biomedical image analysis, computer graphics, motion analysis, deformation model, shape modeling and animation.



Chao-Hung Lin was born in Koushung, Taiwan, Republic of China, in 1973. He received his B.S. in computer science/engineering from Fu-Jen University and M.S. in computer engineering from National Cheng-Kung University, Taiwan, Republic of China, in 1997 and 1998, respectively. Now, he is pursuing his PhD degree at Department of Computer Science and Information Engineering, National Cheng-Kung University. Mr. Lin research interests include computer graphics, image processing, virtual reality, visualization and interactive rendering technique.

Yung-Nien Sun received the B.S degree from National Chiao Tung University, Hsinchu, Taiwan, Republic of China, in 1978 and the M.S. and Ph.D degrees from University of Pittsburgh, Pittsburgh, Pennsylvania, in 1983 and 1987, respectively. He was an assistant scientist with the Brookhaven National Laboratory, New York, from 1987 to 1989. Since 1989, he has joined the faculty of the Department of Computer Science and Information Engineering, National Cheng-Kung University, Taiwan, as an associate Professor and became a Professor in 1993. He has been working on image processing and computer vision since 1982 and has published more than 70 papers, half of them in refereed journals. His current research interests are in medical and industrial applications of computer vision technologies. He is a member of IEEE, Sigma-Xi, the Chinese Association of Image Processing and Pattern Recognition, and the Chinese Association of Biomedical Engineering.