JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 15, 845-858 (1999)

Short Paper

Optimized Semi-Boundary (SB) Rendering Scheme*

TONG-YEE LEE, TZU-LUN WENG AND YUNG-NIEN SUN
Visual System Laboratory
Department of Computer Science and Information Engineering
National Cheng Kung University
Tainan, Taiwan 701, R.O.C.
E-mail: tonylee@mail.ncku.edu.tw

Volume rendering is a technique for visualizing 3D arrays of sampled data. Volume rendering, though capable of performing very effective visualization, is very computationally intensive. Semi-boundary (SB) is a compact data structure used to encode extracted surface from volume data. In this paper, we describe an incremental algorithm to shorten rendering of SB nodes. We define two projection rules to guarantee proper projection order and, thus, eliminate the requirement for a Z-buffer. Using these two rules, we save data storage as well as rendering time. As a result, the proposed method can achieve interactive rendering performance for a large volume pelvis data set with a size of $512 \times 512 \times 245$. Experimental results show that our achieved rendering rate is less dependent on viewing angles.

Keywords: volume rendering, semi-boundary (SB), look-up tables, incremental rendering, medical visualization

1. INTRODUCTION

Three-dimensional arrays of digital data representing spatial volumes are commonly used in medical applications, such as sequences of two-dimensional images derived from CT or MRI scanners. Physicians use these images in surgical diagnosis and therapy planning. However, the amount of data is always very large; therefore, it is inconvenient to interpret it. Several compact storage mechanisms have been proposed to reduce the volume size, such as semi-boundary (SB) [2] and run-length coding [1]. Volume rendering techniques have been used to visualize 3D volume data to aid understanding. Generally, volume rendering methods can be categorized as either indirect or direct volume rendering methods. The former methods first convert 3D scalar or vector data into geometric primitives and then render them using the traditional graphics pipeline. The later directly utilize the original data points in the final rendering of the data.

Direct volume rendering can generate high quality images, but it is very computationally intensive. In the past, many methods have been proposed to reduce its computational cost [1-3, 7-9]. Many techniques have been proposed to accelerate volume

Received March 19, 1998; accepted April 9, 1999.
Communicated by Jang-Ping Sheu.
*This project was supported by the NSC, R.O.C., under grant NSC-87-2213-E006-012.

rendering in the image order, such as ray casting [5], and in the object order, such as splatting [6]. Several special data structures, such as Octree [11] and K-d tree [12], have been proposed to quickly skip empty regions. These special data structures, although capable of accelerating volume rendering, complicate the innermost loop of rendering with special data traversal code and potentially reduce the benefit of time savings. On the other hand, the encoding of empty space significantly reduces rendering computation as well as volume storage. Lacroute et al. proposed a very efficient scheme, called shear-warp [1], that precisely skips empty space using run-length encoding. This method transforms volume data into sheared-object space using optimization, which exploits coherence in both the volume and the image. However, the shear-warp scheme requires three copies of the volume data on memory. This constraint limits the size of the rendered volume data. In this paper, we adopt the semi-boundary (SB) data structure [2] to encode non-empty regions. The drawback of SB rendering is that its computational complexity is in proportion to the number of SB nodes. The main contribution of this paper is to present an incremental algorithm to speed up rendering of SB nodes. As a result, we achieve interactive rendering performance comparable to that obtained on state-of-art graphics workstations. To further shorten the rendering time and reduce the memory requirement, we define two rules to guarantee correct projection and a set of visibility look-up tables to cull invisible SB nodes.

In section 2, we briefly review the SB data structure proposed in [2], describe our two-dimensional SB data structure and introduce visibility look-up tables. The incremental rendering schemes and two projection rules will be presented in section 3. Our implementation details and experimental results will be discussed in section 4. Finally, some concluding remarks and future work will be given in section 5.

2. TWO-DIMENSIONAL SEMI-BOUNDARY (SB) DATA STRUCTURE AND VISIBILITY LOOK-UP TABLES

We adopt the SB data structure to encode extracted surface from volume data. To render SB nodes in an incremental manner, we organize all of the SB nodes in a two-dimensional parallel linked-list data structure. For completeness, we will briefly describe the concept of the SB rendering algorithm. The whole volume of data can be represented by C, a collection of voxel elements, c_s. In [2], segmentation was done simply via a threshold to extract object from volume data. The SB nodes are the collection of 1-voxel c_s used to describe the surface of the object. All SB nodes are passed to the standard graphics pipeline to perform surface rendering. To efficiently implement our rendering scheme, we also include a normal vector look-up table and introduce a set of visibility tables. The normal vector table is widely used to reduce the required storage of normal vectors of voxels. The neighbors of a 1-voxel c can be defined as follows:

$$n(c) = \{d \mid \text{ for some } j,\ 1 \le j \le 3, \mid c_j - d_j \mid = 1 \text{ and } c_i = d_i \text{ if } i \ne j\}.$$

To further reduce the storage requirement for a SB node, we record $n(c)$ in a neighboring configuration code defined by $\Delta(c) = \sum_{d \in n(c)} g_s(d) \cdot m(d)$, where $m(d) = 2^0, 2^1, 2^2, 2^3, 2^4, 2^5$, where g_s is a segmentation function and it will return either, 0 or 1. This configuration code is denoted as m. Later in this section, we will show how this code can be used in combination with visibility tables to cull invisible SB nodes.

After conversion from volume data to the SB structure, the resulting SB nodes are scattered over 3D space (not in an organized manner). To facilitate rendering of the SB nodes, the whole SB data structure will be organized as a two-dimensional (M by N) linked list (as shown in Fig. 1) in this paper. We can imagine that there are M by N virtual rays (originating from the YZ plane) cast along the X-axis. Each virtual ray (linked list) is a collection of SB nodes located on this ray. In Fig. 1, $P[i][j]$ is the starting SB node on each linked-list. Each SB node contains the following information: d is the *distance* between it and the starting SB node, m is its neighboring configuration 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. If $P[i][j]$ is null , this list will be skipped when rendering computation is performed. We will show later how use of this arrangement can facilitate rendering of SB nodes in an incremental manner.

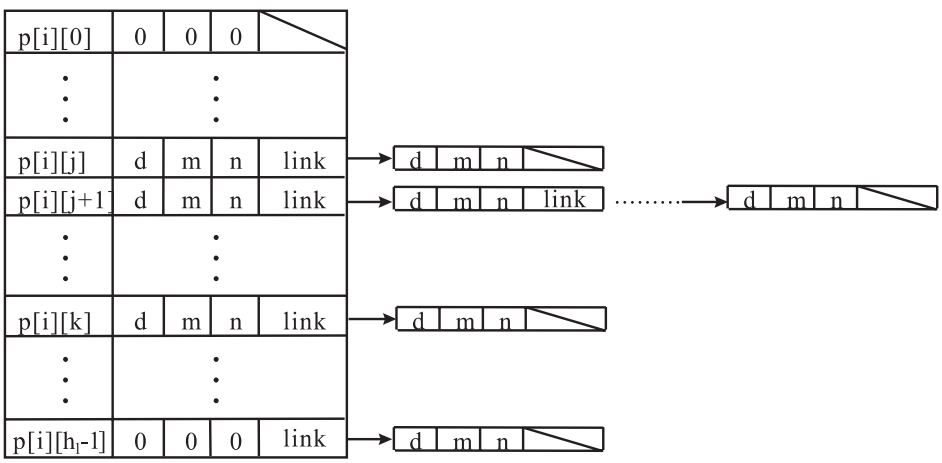


Fig. 1. Two-dimensional SB (Semi-Boundary) data structure.

For any given viewing orientation, we can determine whether an SB node is potentially visible or not based upon its neighboring code. Similar to back-face culling, in general, a half number of SB nodes can be eliminated via this *visible* testing. For this purpose, we create a set of visibility look-up tables. The visibility of an SB node is tested as follows. For a 1-voxel SB node c, D is the down facet of c, T is the top facet, L is the left facet, R is the right facet, B is the back facet, and F is the front facet (as shown in Fig. 2). One of the visibility look-up tables created is shown in Table 1. This table will be used when the rotation angle about the Y-axis is $0 < \theta < 90$. In this case ($0 < \theta < 90$), there are eight combinations (ϕ, rotation angle about the X-axis) for *visible* testing. Similarly, the other visibility tables for rotation about the Y-axis can be created in the same way when $90 \le \theta \le 360$.

In total, eight neighboring tables are created. Using these tables, if the visible facets of an SB node, say *c*, are linked to other voxels, then *c* is not visible. Therefore, only un-culled SB nodes will be further used to perform the rendering process.

Table 1. Visible facet table for $0 < \theta < 90$ (*Y*-axis).

ϕ	Visible facet
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

When the rotation angle (θ) about the *Y*-axis is $0 < \theta < 90$, there are eight combinations (ϕ, the rotation angle about the *X*-axis) for *visibl*e testing.
D: down, T:top, L:left, R:right, B:back, F:front

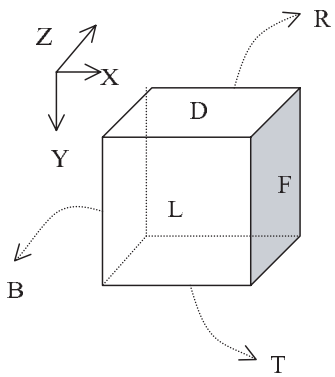


Fig. 2. Label of the six facets of each SB node.

3. INCREMENTAL RENDERING ALGORITHM

In this paper, our main contribution is to propose an incremental algorithm to speed up rendering of SB nodes. As discussed in the previous section, SB nodes are organized into two-dimensional parallel linked lists. Based upon this arrangement, we can render SB nodes in an incremental manner. Our method derives from the intuitive fact that the parallel lines are still parallel after a parallel projection transform. Note that our method does not

work correctly for the perspective projection. However, for medical applications, the image is always viewed via a parallel projection. Therefore, the proposed scheme is still very promising in medical applications. The scheme is described as follows:

As shown in Fig. 3, let X', Y', and Z' be axes for reference frame O, where its origin is located at the center of the volume data, and the world reference frame W is defined by the X, Y and Z axes. The volume data is parallel projected into an image plane that is on W's XY plane. Viewing an object in any orientation can be achieved through a sequence of rotations. We find these three rotation angles regarding the reference frame O's X',Y' and Z' axes, say the angles α about the X' axis, β about the Y' axis, and γ about the Z' axis, respectively. In the proposed scheme, the image projection plane is on the X-Y plane, and the link-list orientation of the SB data structure is created along the X-axis. All link-lists can be thought of as parallel lines along the X-axis. With this arrangement, after a sequence of rotations (first rotation about the X'-axis and second about the Y'-axis) plus a parallel projection, all the link-lists are still parallel to the X-axis on the image plane. This means that all the link-lists will be projected on the scan-line of the image plane. If only X'-axis rotation is required, it is obvious that the projected lines will still be parallel, and that they will also have the same length. However, if Y'-axis rotation is required, each parallel line has to be scaled by the value of cos(β) as shown in Fig. 4.

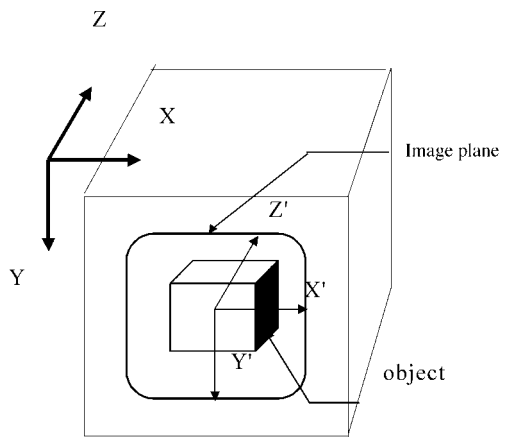


Fig. 3. If $\gamma = 0$, no rotation required for the image plane.

When the angle γ is not zero, the parallel lines in the SB are still kinds of parallel projection onto the image plane. However, the image plane must rotate with the same angle γ after projection to obtain correct results (illustrated in Fig. 5). Following the above concept, we can only project each starting node of all the link-lists on the YZ plane. For the remaining nodes in each list, we multiply each node's *depth* by a constant value cos(β) to obtain its relative offset to the starting node and thus obtain its exact location on the image plane without the need for a parallel projection transform. In contrast, the original SB algorithm must perform a parallel projection for all nodes in the SB data structure.

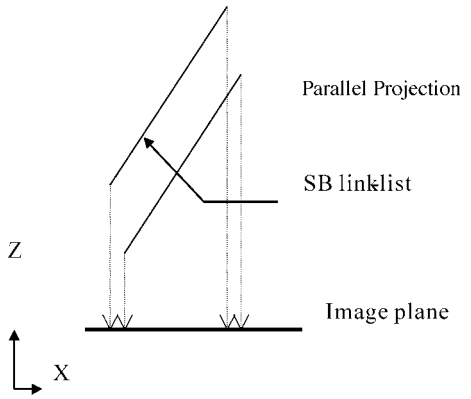


Fig. 4. If $\beta \neq 0$, the length of the link-list is scaled by $\cos\beta$.

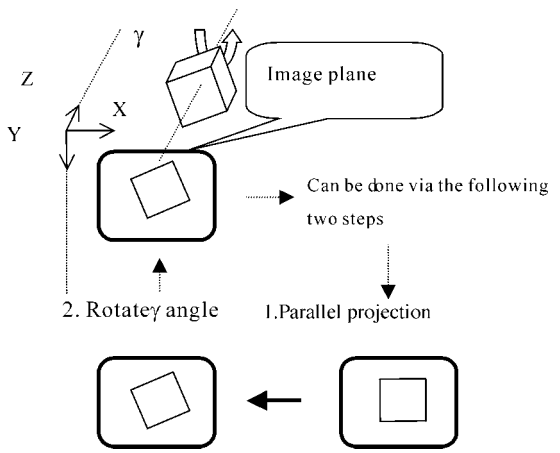


Fig. 5. If $\gamma \neq 0$, then rotate the image plane after projection.

Next, we will show how to parallel project all SB nodes via two rules either in front-to-back or back-to-front order. In this procedure, we do not require that the *Z*-buffer determine the hidden surface removal (i.e., it is storage efficient). Furthermore, we can save time in both storage access and depth computation (i.e., without computing and comparing the *Z* value). The shear warp algorithm [1] needs three copies of volume data to make sure that its algorithm is projected in front-to-back order. This requirement will greatly limit its usefulness for a large data set. To make sure parallel projection is in front-to-back or back-to-front order, we define two rules in our implementation. Rule *R*1 guarantees front-to-back projection and rule *R*2 insures back-to-front projection.

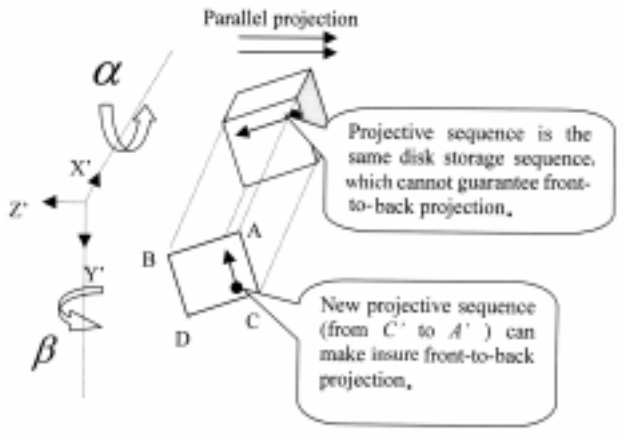


Fig. 6. Front-to-back or back-to-front projection.

***R*1:** The original projective sequence, identical to the disk storage sequence, cannot guarantee front-to-back projection. The coordinates of points *A'*, *B'*, *C'*, and *D'* are (*Z'min*,*Y'min*) , (*Z'max*,*Y'min*) , (*Z'min, Y'max*) , and (*Z'max*,*Y'max*). *Z'min, Z'max, Y'min*, and *Y'max* are the minimum and maximum coordinates of the *Z'* axis and the *Y'* axis of all SB nodes. *A, B, C* and *D* are the points after rotation. Point *F* is a point selected from *A, B, C* and *D* whose *Z*-coordinate is a minimum. *S* is a point selected from *A, B, C* and *D* whose *Z*-coordinate is a minimum except for point *F*. If point *F* is point *C* as shown in Fig. 6, and if point *S* is point *A*, then the new projective sequence (from *C'* to *A'*) is front-to-back projection when $0 \le \beta \le 180$.

***R*2:** In *R*1, if point *F* is the point selected from *A, B, C* and *D* whose *Z* coordinate is a maximum, and if point *S* is the point selected from *A, B, C,* and *D* whose *Z* coordinate is a maximum except for point *F*, then the new projective sequence is back-to-front projection when $180 < \beta < 360$.

The scan-line number and starting position of each linked-list can be decided without a rotation matrix. Next, we will describe how to obtain these values using an incremental algorithm, also. Let *X'*, *Y'*, and *Z'* be the coordinates of objects, and let *X*, *Y*, and *Z* be the coordinates after rotation. $m1 \sim m9$ are the elements of the rotation matrix:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} m1 & m2 & m3 \\ m4 & m5 & m6 \\ m7 & m8 & m9 \end{bmatrix} \bullet \begin{bmatrix} X' \\ Y' \\ Z' \end{bmatrix}. \tag{1}$$

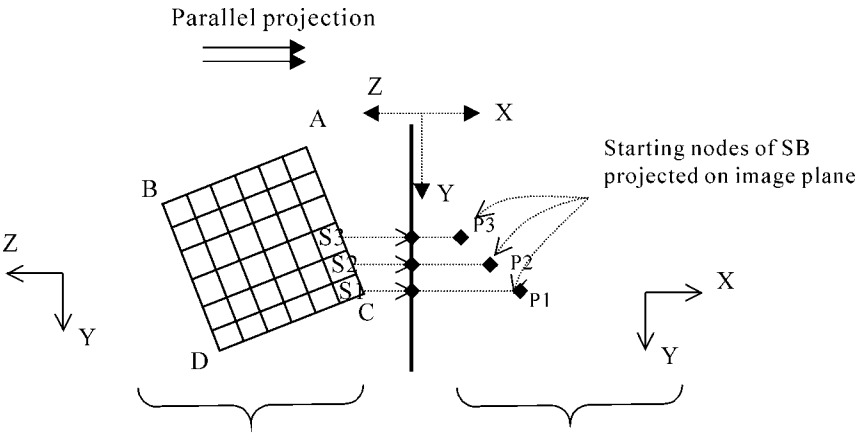
The *X'* coordinate of the starting position of each link-list is a *C* constant, so (1) can be written as (2)

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} \mathrm{m}1 & m2 & m3 \\ m4 & m5 & m6 \\ m7 & m8 & m9 \end{bmatrix} \bullet \begin{bmatrix} C \\ Y' \\ Z' \end{bmatrix}, \tag{2}$$

$$X = m2 * Y' + m3 * Z' + C * m1, \tag{3}$$

$$Y = m5 * Y' + m6 * Z' + C * m4. \tag{4}$$

Y' or Z' is the index of the inner loop or outer loop, depending on the projection sequence. Therefore, ΔX can be determined by $m2$ or $m3$,and ΔY can be determined by $m5$ or $m6$. For a given linked list $S1$, the starting position and scan-line number are known in advance. On the basis of $S1$, the starting position and scan-line number for the neighboring linked list $S2$ can be decided using an incremental method as shown in Fig. 7 and equations (3) and (4). The projection point of $S3$ can be decided by $S2$. The exact image position of each node is the nearest integer.



p1,p2 and p3 are the projected points of SB nodes S1, S2 and S3, respectively, where p2=p1 +(ΔX, ΔY), p3=p2 + (ΔX, ΔY), ΔX can be determined by $m2$ or $m3$, and ΔY can be determined by $\Delta m5$ or $\Delta m6$.

Fig. 7. Determining the scan-line number and starting position using an incremental algorithm.

To fill in some details, a pseudo-code version of the proposed algorithm is listed below (Fig. 8 on next page):

In the above procedure, Part I sets up all the initial parameters and all the related lookup tables. Part II checks the visibility of each SB node, Part III-A performs parallel projection, III-B does incremental rendering, and Part IV performs phong shading.

4. IMPLEMENTATION DETAILS AND EXPERIMENTAL RESULTS

We implemented both SB and our proposed algorithm on a SUN SPARC station-20 with 64M memory, and evaluated their performance in rendering of a $512 \times 512 \times 245$ pelvis volume data for an image resolution of 512×512. The primary sources of the overheads in both algorithms can be organized in the four categories [10] listed below:

```
Procedure for an incremental SB rendering algorithm;
begin
//— — — — — — — — — — — — — — — — — — — — — — — — — — — — — — — — — —
//Part I: initialize parameters and lookup tables
//— — — — — — — — — — — — — — — — — — — — — — — — — — — — — — — — — —
   //obtain image viewing transform configuration
   spin=trans->spin; tilt=trans->tilt;
   cosx=trans->cosx;cosy=trans->cosy;
   sinx=trans->sinx; siny=trains->siny;
   // initialize lookup tables
   Set Normal Lookup( );
   SetVisibill Lookup( );
   decide_projection_sequence( );  /** use rule R1 or R2 to determine projection order **/
   for i: = Z'min to Z'max do begin
      for j:= Y' max to Y'min do begin
      cur = &sb[i][j];
//— — — — — — — — — — — — — — — — — — — — — — — — — — — — — — — — — —
// part III-A: for each linked list, compute the position
// of starting SB node via a parallel projection and also
// decide the projected scan-line number of each linked-list
//— — — — — — — — — — — — — — — — — — — — — — — — — — — — — — — — — —
      X1=X1old + ΔX;
      Y = Yold+ΔY;
      while (curl=NULL)
      {
//— — — — — — — — — — — — — — — — — — — — — — — — — — — — — — — — — —
// Part II: Visibility culling via look-up tables
//— — — — — — — — — — — — — — — — — — — — — — — — — — — — — — — — — —
if this SB node is visible under current viewing orientation
do begin
//— — — — — — — — — — — — — — — — — — — — — — — — — — — — — — — — — —
// Part III-B: computation other nodes along each linked-list
// in an incremental manner.
//— — — — — — — — — — — — — — — — — — — — — — — — — — — — — — — — — —
      X=X1 + cur->depth*cosy;
//— — — — — — — — — — — — — — — — — — — — — — — — — — — — — — — — — —
// Part IV: do phong shading
//— — — — — — — — — — — — — — — — — — — — — — — — — — — — — — — — — —
         perform phong shading of a current SB node;
      }// end of while
         cur:=next link-list node;
      end;// end of j
end;// end of i

if r ≠ 0 do image rotation;
end;//end of procedure
```

Fig. 8. Pseudo-code of the proposed rendering algorithm.

1. Looping: This includes overheads spent on control overheads such as updating loop counters, advancing pointers and traveling the linked-list structure. The method of memory access to data is crucial for determining this overhead [1,10].
2. Neighboring configuration tests: This is time spent on checking the visibility of SB nodes.
3. Parallel projection transformation: This is time spent on parallel projection. For SB rendering, this task is the most time-consuming part of Part III A compared with the other overheads. On the other hand, in our proposed method, we perform III A only for the starting SB nodes and perform III B for the remaining un-culled nodes.
4. Z-buffering and shading: This is time spent on hidden-surface removal and shading of the SB nodes. In our proposed method, the hidden surface removal is replaced by front-to-back or back-to-front projection. Compared with the SB rendering, no Z value comparison is required in our proposed method.

The other miscellaneous overheads, such as lookup table initializations (about 2~3 milliseconds), are less significant than the above listed overheads. From our experiments, the rendering time needed by SB and our proposed method is 1567 and 716 milliseconds, respectively. These results are the average of rendering 180 consecutive frames (1 frame per 2 degrees of rotation). The rotation axis was set on the *Y*-axis. The proposed incremental algorithm is faster than the original SB algorithm by a factor of 2.19 times. Visibility culling will improve rendering performance. Therefore, to fairly compare our method with the original SB scheme, both methods included visibility look-up tables in our comparison study. To further analyze the performance difference, a breakdown of the total execution time for SB and our method is shown in Tables 2 and 3. Both tables show the exact timings and the percentages of the total execution time for each category of overheads.

Table 2. Detailed timing breakdowns for the SB algorithm.

SB algorithm	Timing	Percentage
Looping	324	21%
Visibility culling	68	4%
Part III A only	947	60%
Z-buffering and Shading	228	15%

Table 3. Detailed timing break-downs for the new rendering algorithm.

new algorithm	Timing	Percentage
Looping	324	45%
Visibility culling	68	10%
Part III A and B	172	24%
Shading and front-to-back or back-to-front projection	152	21%

From these two tables, we can see clearly that Part III is significantly improved by our incremental algorithm, which is about 5.5 times faster. Similarly, front-to-back or back-to-front projection is better than Z-buffer. In both tables, the percentage results show that the the overheads are incurred in Part III. Some future work can be done to improve the performance of other time killers, such as looping.

The rendering time depends on the viewing angle while the number of the SB nodes culled in Part II may vary with the viewpoints. Fig. 9 shows the rendering time versus the viewing angle for our algorithm. Recall that un-culled SB nodes will be executed in Part III and, thus, will make difference in rendering time. In this figure, we see some variation in the rendering time as the rotation angle increases. To verify our claim (i.e., that variations are mainly due to the number of un-culled SB nodes), we plot in Fig. 10 the un-culled SB nodes executed by the proposed algorithm. Referring to both Figs.9 and 10, we see clearly that the variations in both figures are similar. Finally, we present one frame of our rendered results for pelvis data in Fig. 11.

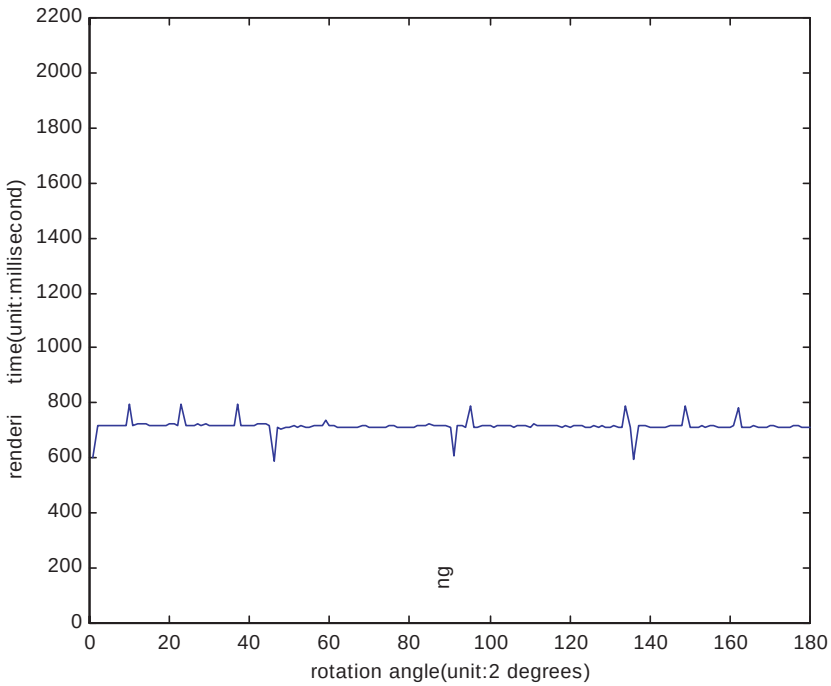


Fig. 9. Rendering time for the proposed rendering algorithm.

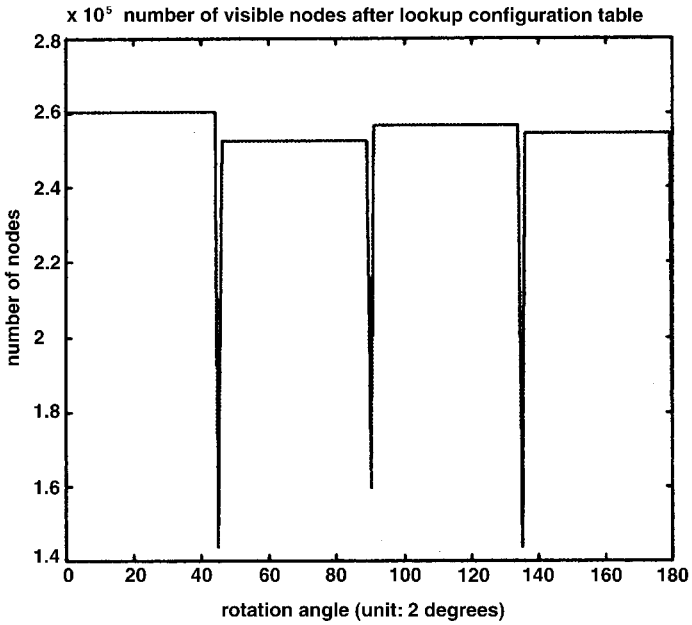


Fig. 10. The number of un-culled SB nodes versus different viewing angles.

Fig. 11. A rendering image of a pelvis (data set: 512×512×245).

5. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented an incremental SB rendering technique. This scheme can achieve interactive performance in rendering large volume data, such as $512 \times 512 \times 245$ pelvis data. The incremental algorithm accelerates the computation of parallel projection. We use two simple rules, $R1$ and $R2$, to determine the order of projection and thus reduce the hidden surface time. As a results, no Z buffer is required to perform hidden surface removal. We have experimentally analyzed our algorithm. The algorithm performed better than the original SB algorithm on our pelvis data. Much future work will be done as follows. We plan to enhance our scheme with morphing technology to perform surgical simulation for the pelvis system. Some user-friendly manipulation tools will also be developed soon.

REFERENCE

1. Philippe G. Lacroute, "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, 1995.
2. Jayaram K. Udupa and Dewey Odhner, "Fast visualization, manipulation, and analysis of binary volumetric objects," *IEEE Computer Graphics and Applications*, Vol. 11, No. 6, 1991, pp. 53-62.
3. Jayaram K.Udupa and Dewey Odhner, "Shell rendering," *IEEE Computer Graphics and Applications*, Vol. 13, No. 6, 1993, pp. 58-67.
4. L. M. Luo, C. Hamitouche, J. L. Dillenseger and J. L. Coatrieux, "A moment-based three-dimensional edge operator," *IEEE Transactions on Biomedical Engineering*, Vol. 40. No. 7, 1993, pp. 693-703.
5. M. Levoy, "Display of surfaces from volume data," *IEEE Computer Graphics and Applications*, Vol. 8, No. 3, 1988, pp. 29-37.
6. L. Westover, "Interactive volume rendering," *Workshop on Volume Visualization*, Chapel Hill, NC, 1989, pp. 9-16.
7. D. Cohn and Z. Shefer, "Proximity clouds – an acceleration technique for 3D grid traversal," Technical Report, TR-FC-93-01, Department of Mathematics and Computer Science, Ben Gurion University, Israel, 1993.
8. R. Yagel, "Shell accelerated volume rendering of transparent regions," *The Visual Computer*, Vol. 10, No. 1, 1994, pp. 53-61.
9. K. Zuiderveld, A. Koning, Viergever and A. Max, "Acceleration of ray-casting using 3D distance transforms," in *Proceedings of SPIE*, Vol. 1808, 1992, pp. 324-335.
10. T. L. Weng, T. Y. Lee, Y. N. Sun, "New rendering method: scan-line based semi-boundary algorithm," in *Proceedings of SPIE*, Vol. 3335, 1997, pp. 20-27.
11. M. Levoy, "Efficient ray tracing of volume data," *ACM Transactions on Graphics*, Vol. 9, No. 3, 1990, pp. 245-26
12. K. R. Subramanian and D. S. Fussell, "Applying space subdivision techniques to volume rendering," in *Proceedings of Visualization*, 1990, pp.150-158.

Tong-Yee Lee (李同益) was born in Tainan county, Taiwan, Republic of China, in 1966. He received his B.S. in computer engineering from the Tatung Institute of Technology in Taipei, Taiwan, in 1988, his M.S. in computer engineering from National Taiwan University in 1990, and his Ph.D. 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 the School of EE/CS during the summer of 1996. He has been working on parallel rendering and computer graphics since 1992 and has published more than 60 technical papers in referred journals and at conferences. His current research interests include parallel rendering design, computer graphics, visualization, virtual reality, surgical simulation, distributed & collaborative virtual environments, parallel processing and heterogeneous computing.

Tzu-Lun Weng (翁子倫) was born in Taiwan, R.O.C., in 1965. He received the M.S. 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 the Ph.D. degree in the Department of Computer Science and Information Engineering, National Cheng-Kung University. His current research interests include computer vision, biomedical image analysis, computer graphics, motion analysis, deformation model, shape modeling and animation.

Yung-Nien Sun (孫永年) received the B.S. degree from National Chiao Tung University, Hsin-chu, Taiwan, Republic of China, in 1978 and the M.S. and Ph.D. degrees from the 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. In 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 he 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 referred 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.