Efficient Parallel Splatting Algorithm

Chih-Hsien Huang, Jen-Shiuh Liu, and Don-Lin Yang

Department of Information Engineering and Computer Science
Feng Chia University, Taichung, Taiwan 407, ROC
Email: {cshuang, liuj, dlyang} @iecs.fcu.edu.tw

ABSTRACT

Splatting is a technique for reconstruction in feed-forward volume rendering, which consumes large portion of time in the entire rendering process. In this work, we exploit object space coherence in the splatting process. Sparse-matrix data structure is introduced to take advantage of object space coherence in order to save computation time and memory storage. With the spatial data structure, no computation is performed for those uninteresting voxels during the splatting process. It is known that volume rendering may take several seconds to hours to render a 3D dataset on a typically single-processor. We also present data-parallel volume rendering algorithms in order to achieve real time applications. Experiments are conducted to assess our proposed schemes. Results show that the proposed sparse-matrix algorithm performs very well.

1. INTRODUCTION

Computer graphics and scientific visualization are fast growing applications domains. There are many volume visualization techniques that help users to extract meaningful information from simulation or experimental results [2]. Volume rendering enable users to visualize scalar and vector fields defined on three-dimensional grids. The task for volume rendering is to render 3D models on 2D graphic displays. Some typical data sources for volume rendering are Magnetic Resonance Imaging (MRI), Computed Topography (CT), Ultrasound, and etc.

Indirect volume rendering is an early volume visualization technique, which converts the 3D voxel representation into 2D graphic primitives (such as curves and lines) and then displays them on conventional graphic devices [6]. Indirect volume rendering suffers from disadvantage such as hard to yield a satisfactory result for certain studies [22]. Direct volume rendering is a new technique, which does not employ intermediate graphic primitives that do not really exist in the volume datasets. In general, direct volume rendering is to map each voxel with its color and opacity in the volume dataset to pixels on the image plane. One major advantage of direct volume rendering is that which enables us to visualize internal structure of any volume datasets much more easier.

By Westover's definition [19], direct volume rendering can be divided into two broad categories: feed-backward and feed-forward mapping methods. The difference between them is the mapping direction. In feed-backward mapping methods, the renderer first calculates effect of each voxel onto pixels on the image plane, stores it in an image buffer, and finally sums up contribution from all voexls to make the final image. Splatting [18] is one of the most popular feed-forward techniques. In feed-backward mapping methods, a ray from each pixel on the image plane is casted into the volume dataset. The graphic value of each pixel is calculated by integrating the color and opacity along a ray. Ray casting is a one of the most popular feed-backward techniques [5,15].

It is known that the computation cost of direct volume rendering increases exponentially as the size of the volume datasets increases. Therefore, it is an important issue to reduce the computation work in order to have fast volume rendering. Existing studies [4,20] have reported that about 70%~95% voxels of volume data are uninteresting, which means that they play no role in constructing the final image. For CT or MRI, many uninteresting voxel points may come from air, which can be identified and hence need not be processed after some classification procedure. It is clearly that by rendering only interesting, according to users' choice, voxel points can save computation time. Many works have employed this idea to speed up the volume rendering. One of such is to exploit object space coherence in the volume by using spatial data structure. The idea of using spatial data structures is to encode the location of interesting voxels in a compact way in order to enable fast rendering. The k-d tree [16], pyramids [3,4,21] and run-length encoding [7,13] are examples among them. Levoy has developed a ray-casting algorithm [4] using octree to explicit object space coherence, which can achieve a 3 to 5 times speedup over a regular ray-casting algorithm. Several studies have reported that, by using spatial data structure, feed-forward volume rendering algorithm is more effective than feed-backward one because of traversal order [3,4]. Hence designing of spatial data structures is an important issue in obtaining efficient feed-forward volume rendering

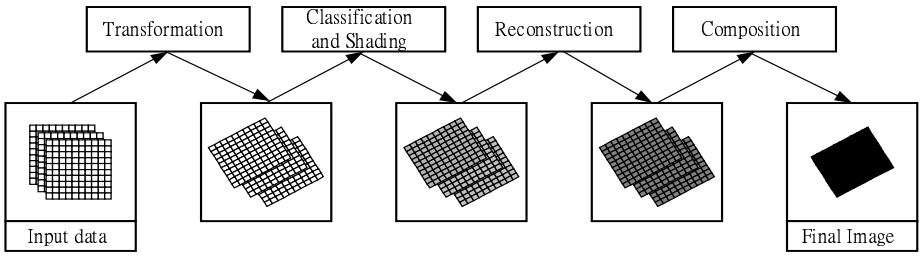


Figure 1. The four stages of a feed-forward volume rendering pipeline.

Splatting is a technique for reconstruction in feed-forward volume rendering, which consumes large portion of time in the entire rendering process. In this work, we exploit object space coherence in the splatting process. We propose to use spatial data structure that efficiently records location of interesting voxels. With the spatial data structure, no computation is performed for those uninteresting voxels during the splatting process. Voxels in a 3D space can be viewed as elements in a 3D matrix. To exploit the object space coherence, we employ three different sparse-matrix (SM) representations in our proposed spatial data structure, which are Compressed Row Storage (CRS), Bit-Map (BM), and Run Length Encoding (RLE). Volume rendering is normally processed in a pipeline fashion. With the introduction of spatial data structure, we add a new stage called sparse-matrix encoding before the reconstruction stage. The sparse matrix representations will reduce both rendering time and storage requirement. It is known that volume rendering may take several seconds to hours to render a 3D dataset on a typically single-processor [10]. We also present data-parallel volume rendering algorithms in order to achieve real time applications.

The rest of this paper is organized as follow. In Section 2, we give a brief review of feed-forward volume rendering process and some splatting algorithms. In Section 3, we present three sparse-matrix representations to exploit object space coherence for fast splatting. Data-parallel volume rendering algorithm based on SM splatting is presented in Section 4. Experimental results are reported in Section 5. Finally, we give our concluding remarks in Section 6.

2. PRELIMINARIES

In this section we review feed-forward volume rendering pipeline and some splatting algorithms. Figure 1 shows the four stages of a typical feed-forward volume rendering pipeline [23]. Task performed in each stage is described as follow.

Transformation is the first stage, which converts input data in object space to image space. Each voxel is transformed by multiplying its coordinate vector with the transformation matrix, which is constructed from viewing parameters such as position of the eye. A value of opacity is assigned to each voxel based on user's choice in the classification stage. A good classification function can help users to extract meaningful information from the data for visualization. To get more realistic images, shading is necessary. Phong shading algorithm [9] is the most popular one, which requires surface normal determined by gray-level gradient scheme [11]. Reconstruction is achieved by splatting each voxel onto a 2D image plane. Splatting can be viewed as throwing a snowball (voxel's contribution) onto a glass window (image plane). All pixels lying in the footprint extent of a voxel are contributed by its splat through a change in color and opacity. In the la st stage of the rendering pipeline, the composition rules [12] are responsible for summing up color and opacity contributions from each pixel of 2D image plane for display. Once the composition process is complete, the feed-forward volume rendering pipeline produces a final image.

Splatting, proposed by Westover [17,18,19], is an object order traversal algorithm, where voxels of volume are properly splatted onto a 2D image plane. In the original splatting method [19], the voxels are sorted slice by slice where each slice is most orthogonal to the viewing direction. During the reconstruction, each voxel in a slice is represented by a 3D reconstruction kernel, which is pre-integrated into a 2D footprint, weighted by the voxel value and then accumulated to an image buffer. The contribution of each voxel to the 2D image plane is calculated by convolving with a 3D reconstruction kernel that distributes the discrete voxel value to all pixels lying in the footprint extent. This reconstruction process is illustrated in Figure2. The idea to use spatial data structure to skip uninteresting voxels in order to speed splatting has been reported in the past.

In [3], a pyramid data structure is proposed to represent volume data, which allows a hierarchical splatting for progressive refinement. After transformation and shading, an octree is constructed, where each node contains the average RGBA value of all its children and a value indicating the average error associated with that

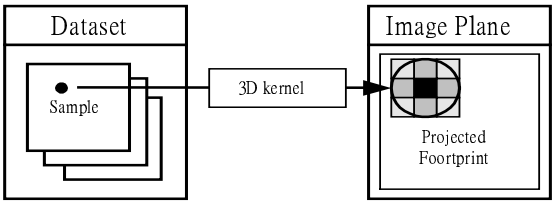


Figure 2: Splatting Process

node. Then reconstruction is started at the root of the octree. For each level, if the checked node is completely filled with data and its average error is below a user-defined limit, then an appropriate splat size (corresponding to the tree node) is projected on to the 2D image plane. Therefore the object space coherence areas such as constant-value and empty areas are rendered with a lower resolution to speed up the rendering time of splatting. If the node's average error is below a user-defined limit or the node is not completely filled with data, the algorithm has to down one level and continue the work until the final image is completed. One disadvantage of this approach is that it trades off image quality for speed.

Our proposed SM splatting has two features compared to the hierarchical splatting: (1) It is simpler in implementation. Only one extra stage is added to the original pipeline, and (2) No image quality is traded for speed.

3. SPARSE-MATRIX DATA STUCTURE FOR SPLATTING

Sparse-matrix data structure is introduced to take advantage of object space coherence in order to save computation time and memory storage. The original volume data can be viewed as a 3D matrix, or a set of 2D matrices (slices). In our proposed work, only voxels with opacity in certain range will be considered interesting or valid. We use three different schemes to exploit object space coherence. As the sparse-matrix encoding algorithm traverses voxels of volume, it encodes interesting voxels into a special sparse-matrix data structure for later processing. It is known that there are many ways to represent a sparse-matrix. To explore different sparse-matrix representation schemes and ease our software development, we devise a common structure for all three schemes studied in this work.

3.1 Data Structure for Sparse-Matrix

Figure 3 displays the common structure for the sparse-matrix volume. There are three structured array components: 1) encoded data array (EA), 2) interesting voxels array (IA), and 3) slice pointers array (PA). The EA records positions of interesting voxels that need to be processed during the splatting. Data (opacity and color) of

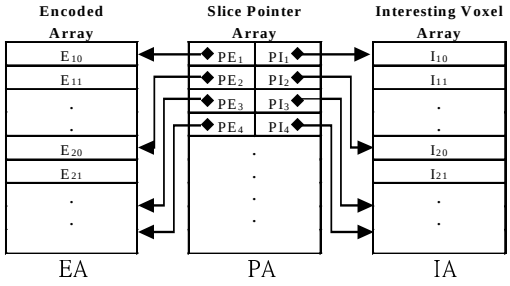


Figure 3: Common data structure for sparse matrix

each interesting voxel is stored in IA. The volume data is divided into slices. Each element of PA has two pointers: one points to the position and the other points to the data of the first interesting voxel. This will enable us to do fast random access on any slice, which is an important feature for interactive rendering.

Volume data in each slice can be viewed as a 2D matrix. In our common data structure we intend to use two separate arrays to record volume data. By doing this, we can easily explore different sparse matrix data structures with minimum software overhead. This is because we need to properly construct EA according to different representation schemes. However, in all schemes the IA remains the same.

3.2 Sparse-Matrix Representation Schemes

In this section we examine three different sparse-matrix representation schemes. They are: Compressed Row Storage (CRS), Bit-Map (BM), and Run Length Encoding (RLE).

3.2.1 CRS Scheme

CRS [1] is a common scheme to represent sparse matrices. Figure 4 illustrates an example, which shows a sparse matrix O is represented by EA and IA. The EA indicates positions of those interesting voxels. Each voxel value is stored in IA. In CRS the EA consists of two arrays: element i in INDR array points to the starting address of the first non-zero element in the i-th row; value in COL indicates the column index of the voxel.

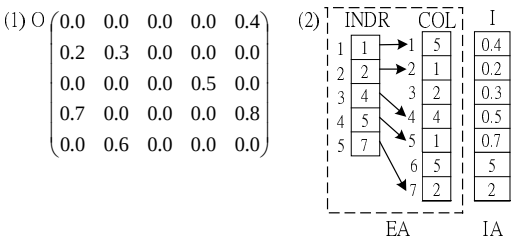


Figure 4: A sparse matrix and its CRS representation.

3.2.2 BM Scheme

The bit-map is also called binary matrix [14] scheme,

where one is used to represent whether a matrix element is zero or not. For a matrix with n elements, the EA consists of a string of exactly n bits one for each element. The interesting voxels are stored in IA. Figure 5 is an example of bit-map scheme, which shows a sparse matrix O is transformed to binary matrix, then bit map and finally to EA with 8 bits per byte. Since IA is the same as the previous CRS scheme, it is not shown in Figure 5.

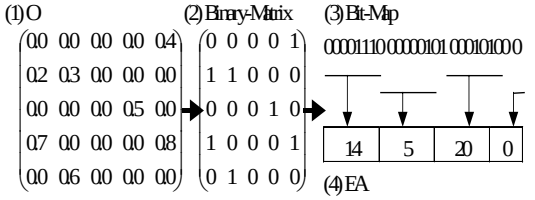


Figure 5: Bit-map representation: (1) sparse matrix O,(2) binary matrix, (3) BM, and (4)EA for BM.

3.2.3 RLE Scheme

One bit is required to represent each voxel in bit-map scheme. Continuity is one characteristic for most volume datasets. By continuity we mean that neighbors of an interesting voxel are mostly interesting. Therefore, the run length encoding scheme will take less memory storage to represent the binary matrix. Another effect of RLE is that it allows us to skip uninteresting voxels more easily during the splatting. The only difference between BM and RLE schemes is the representation in EA. The binary matrix can be viewed as a string consisting of alternate runs of interesting and uninteresting voxels. Hence, run length of uninteresting and interesting voxels is recorded in EA alternately. Each run length is represented by an 8-bit byte, which results in a maximum length 255. A run is split into several runs if its length exceeds 255. Figure 6 is an example that shows a sparse matrix O is transformed into binary matrix then into RLE.

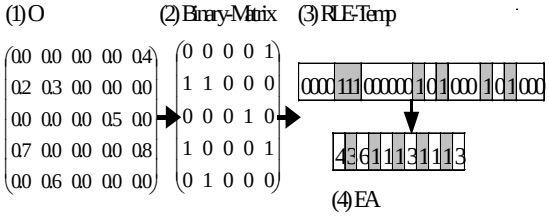


Figure 6: RLE representation:(1) sparse matrix O, (2) binary matrix, (3) RLE, and (4) EA for RLE.

4. PARALLEL SM SPLATTING ALGORITHM

In general, a volume rendering process takes several seconds to hours to render a 3D volume data on a typically single-processor, which may not be acceptable in certain applications. Parallel processing is one direction for fast volume rendering. Sparse data structure has been proposed to speed splatting. In this section, we present data-parallel volume rendering algorithm based on sparse data structure. The data parallel volume rendering algorithm is based on Single Program Multiple Data (SPMD) model, where each processor executes the same program code on different dataset. Our proposed

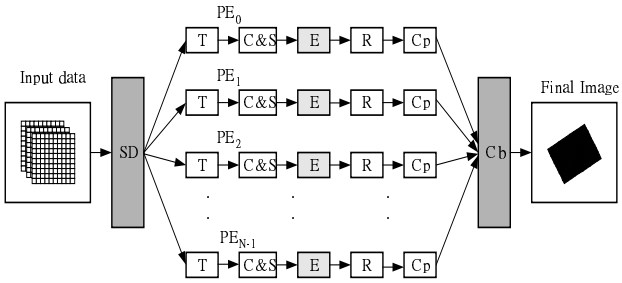


Figure 7: Parallel rendering pipeline with SM Splatting.

```
/*Decomposition stage for parallel volume rendering*/
01  On each processor Pi with slices S0, S1,……,Sj
/*Depend on shading or not*/
02    Initialize light and shade table if need
03    Precompute normal vector and gradient sizes for
      Shading
/*Encoding input volume into sparsematrix volume*/
04    for each slice
05      Transfer to image space
06      Encoding slice by Sparse-Matrix technique
        according to user-defined
/* SM splatting algorithm with decoding the sparse-matrix
volume*/
07    for each slice
08      for each meaningful voxel based on EA array
09        Compute an appropriate footprint extent
10        Project footprint onto the 2 D image plane
11          for each pixel lying in the footprint extent
12            Weighted by the voxel value and
              accumulated to an image buffer
13      Composite successive slices to produce the local
        Image
/*Combining stage using hierarchical scheme*/
14    for i =1to log(N)
15      if (PE's State == "send")
16        Send local image information to its partner PE
17      else
18        Receive and Combine image information from
            its partner PE
19    if ( Pi = display processor )
20      Display final image
```

Figure 8: Pseudo-code for a parallel volume rendering process.

algorithm requires few data to be communicated among processors. Figure 7 illustrates the parallel volume rendering pipeline, which is similar to the traditional rendering pipeline except with a few changes.

The Slab Decomposition (SD) is introduced to distribute object space data to all the processors. Suppose that we have n processors, then the entire volume is partitioned into n most equal sized slabs, where the i'th slab is assigned to the i'th processor. Each processor receives a slab of several slices data, then processes sequentially the four pipeline stages of volume rendering slice by slice. The four stages are Transformation (T), Classification and shading (C&S), and Reconstruction (R) and Composition (Cp). After these, each processor has a local image corresponding to the assigned slab. Combination (Cb) is introduced to glue n local images to form a single final image for display. To take advantage of spatial coherence, the Encoding (E) stage is added before reconstruction (splatting), which employs our proposed sparse matrix data structure to represent volume data. It is clear that data communication only occurs in slab decomposition and combination stages. Pseudo-code for parallel volume rendering process is presented in Figure 8.

5. PERFORMANCE RESULTS

Some experiments are performed to assess our proposed SM parallel volume rendering.

5.1 Experiment Platform

We have implemented parallel splatting algorithm based on our proposed sparse-matrix data structures on an IBM SP2 machine. The system consists of 80 IBM POWER2 CPU with 66.7 MHz clock rate, 128KB 1st-level data cache, 32KB 1st-level instruction cache, and 128MB of main memory. Nodes are connected through a low-latency, high-bandwidth interconnection network called the High Performance Switch (HPS).

We have used C language with MPI [8] message-passing library to implement our algorithm. This makes our parallel SM splatting algorithm portable to many distributed memory or shared memory multiprocessor systems.

5.2 Test Dataset

We select three different volumetric datasets for performance evaluation. Table 1 summarizes their size and sparse ratio (SR). The SR is defined as percentage of uninteresting to the total number of voxels in the dataset. As SR increases, the number of uninteresting voxels increases, and hence the amount of work taxed by splatting decreases. We use a voxel's opacity value to define its interestingness. A voxel with its opacity value greater(less) than the user-defined threshold is called (un)interesting. Opacity value for our test datasets is between 0 and 255. SR_*x* denotes that the user-defined threshold is *x*. Hence, SR_10 indicates the percentage of uninteresting voxels with opacity less than 10.

Volume Datasets	Volume Dimensions	Density_Threshold		
		SR_0	SR_10	SR_30
Cube	256×256×110	98%	98%	98%
Head	256×256×113	0%	47%	70%
Engine	256×256×110	27%	77%	81%

Table 1: Test Datasets.

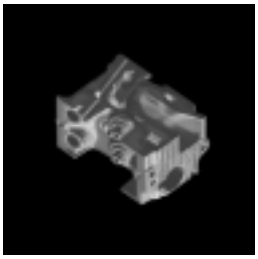
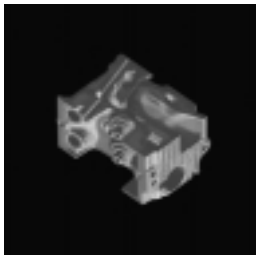
No sparse matrix used	Using CRS with SR_10
	

Figure 9(a): Images of test datasets: Engine

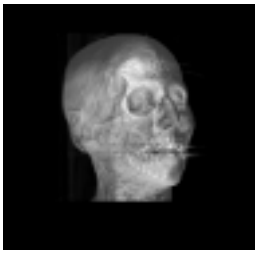
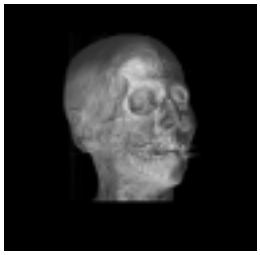
No sparse matrix used	Using CRS with SR_10
	

Figure 9(b): Images of test datasets: Head

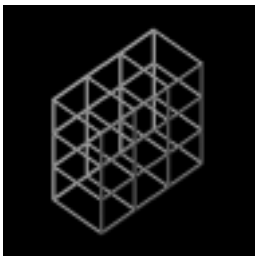
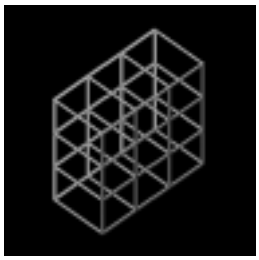
No sparse matrix used	Using CRS with SR_10
	

Figure 9(c): Images of test datasets: Cube

In our experiment, volume dataset may become sparse by varying user-defined threshold. A gray-level image of size 256×256 is generated for each dataset using

Volume Datasets		SR	Original Splatting	SM splatting					
Type	Threshold			CRS		Bit-Map		RLE	
			Ave. (Sec)	Ave.	Perf.	Ave.	Perf.	Ave.	Perf.
Cube	10 / 30	98%	4.081	1.100	73%	1.454	65%	1.275	68%
Head	10	47%	7.644	5.460	29%	5.862	24%	5.958	23%
	30	70%	7.102	3.467	51%	4.067	43%	4.296	40%
Engine	10	77%	6.490	2.891	55%	3.294	49%	3.530	46%
	30	81%	5.518	2.503	54%	3.098	44%	3.365	40%

Table2: A comparison of original and three SM Splatting methods.

parallel projection in the rendering process. Figure 9(a), 9(b), 9(c) shows the rendering images of the original dataset and with SR_10, respectively. It can be seen that there is almost no observable difference.

5.3 Experimental Results

Sparse ratio is a determinant factor for our proposed SM splatting algorithm. A comparison of the original and three SM splatting algorithms is summarized in Table 2.

For each threshold value, there are four splatting times (original and three SM) and three relative performance gains listed. The time reported for SM methods includes both encoding to SM structure and splatting. The performance gain (Perf.) is defined as follow:

$$Performance\_Gain(\%) = \frac{Time(Original\_Splatting) - Time(SM\_Spaltting)}{Time(Original\_Splatting)} \times 100$$

It can be seen that the SM methods have a performance gain of 23~73%. In the best case (Cube with threshold 10), the CRS scheme results in a speed up about 3.7. In the worse case (Head with threshold 10), the RLE scheme still results in a speed up about 1.3. These values indicate that the SM method provides a new technique that speeds the splatting without trading off image quality. The size of footprint table is also a major factor that affects the performance of SM methods.Table 3 and Figure 10 shows that as the footprint size increases the performance gain also increases for both head and engine datasets. This is because large footprint table requires more work to do. By exploring sparse property, we save more time on splatting.

After examining the effects of sparse ratio and footprint table size, we move our attention to parallel implementation. We run parallel splatting with 1,2,4,8,16 and 32 computing nodes. Two forms of speedup are defined in order to give us further insight of parallel SM splatting. Figure 11 displays speedup values relative to original splatting with single node. On the other hand, figure 12 shows speedup values for each splatting algorithm relative itself with single node. More precisely, S_n and S'_n are defined as follow:

Volume Datasets	Footprint Size	Original Splatting	SM splatting					
			CRS		Bit-Map		RLE	
		Ave. (Sec)	Ave.	Perf.	Ave.	Perf.	Ave.	Perf.
Head_30	2×2	7.102	3.467	51%	4.067	43%	4.296	40%
	3×3	12.81	5.086	60%	5.651	56%	5.740	55%
	4×4	20.47	7.287	64%	8.035	61%	7.971	60%
	5×5	31.52	9.197	71%	10.65	67%	10.06	68%

Table3: A comparison original and SM methods with different footprint size.

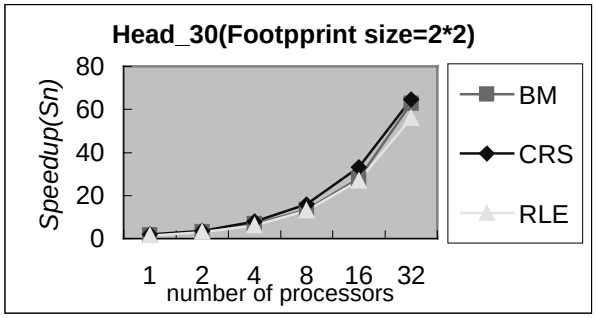


Figure 11: Speedup (S_n) relative to the original splatting algorithm.

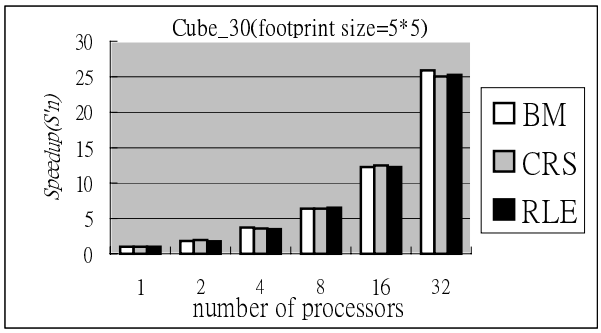


Figure 12: Speedup (S'_n) relative to algorithm itself.

$$S_n = \frac{T_1}{T_n}$$

where T_1 is the execution time of the original splatting run on one processor, and T_n is the execution time of a SM splatting run on n processors. From figure 11, we can see that the CRS scheme has the best speedup in general.

$$S'_n = \frac{T'_1}{T'_n}$$

where T'_1(respectively, T'_n) is the execution time of a SM splatting run on one(respectively, n) processor(s). Figure 12 shows that all SM methods scale well as processor number increases.

6. CONCLUSIONS

In this work, we exploit object space coherence in the splatting process. Sparse-matrix data structure is introduced to take advantage of object space coherence in order to save computation time and memory storage. With the spatial data structure, no computation is performed for

those uninteresting voxels during the splatting process. We also present data-parallel volume rendering algorithms in order to achieve real time applications. Experiments are conducted to assess our proposed schemes. Results show that the proposed sparse-matrix algorithm performs very well.

Splatting(reconstruction) is one stage in the rendering pipeline. At present, our proposed sparse-matrix scheme only applies to the splatting process. We are investigating on extending the sparse-matrix scheme before the shading process, which should further improve the performance of the entire rendering process.

REFERENCES

[1] Barrett, R., Berry M., Chan, R, P., Demmel, J., Donato, J., Dongarra, J., Eijkhout, V., Pozo, R., Romine, H., "Template for the Solution of Linear System: Building Blocks for Interactive Method," SIAM, 1994.

[2] Kaufman (Eds.), "Volume Visualization," *IEEE Computer Society Pres*s, 1991.

[3] Laur, D., Hanrahan, P., "Hierarchical splattings: A progressive refinement algorithm for volume rendering," *Computer graphics*, 1991.

[4] Levoy, M., "Efficient ray tracing of Volume data," *ACM Transactions on Graphics*, July 1990.

[5] Levoy, M.S., "Volume Rendering: Display of Surface from Volume Data," *IEEE Computer Graphics and Applications*, vol. 8, no. 3, May 1998.

[6] Lorenson, W.E., Cline, H. E, "Marching Cube: A High Resolution 3D Surface Construction Algorithm," *Computer Graphics*, vol.21, no. 3, July 1987.

[7] Montani, C., Scopigno, R.,"Rendering volumetric data using the STICKS representation scheme," *Proceedings of the 1990 Workshop on Volume Visualizatio*n, 1990.

[8] MPI Fourm, "MPI: A message-passing interface standard," may 1994.

[9] Phong, B., "Illumination for Computer Generated Pictures," *Comm ACM*, vol. 18, no. 6,1975.

[10] Neumann, U., "Interactive Volume Rendering on a Multicomputer," *Proceedings of the 1992 Symposium on Interactive 31*.1992.

[11] Pommert, A., Tiede, U., Wiebecke, G., Hoehne, K. H., "Image Quality in Voxel-Based Surface Shading," *Processing of the International Symposium on Computer Assisted Radiology*, 1989

[12] Porter, T., Duff, T., "Composition Digital Images," *Computer Graphics*, 1984.

[13] Reynolds, R. A., Gordon, D., Chen, L. S.," A dynamic screen technique for shaded graphics display of slice-represented objects," *Computer Vision, Graphics, and Image Processing*, 1987.

[14] Robert, J., Linda G., "Data Structures And Their Implementation," Van Nostrand reinhold University Computer Science Series, 1982.

[15] Sabella, P., "A Rendering Algorithm for Visualization 3D Scalar Data," *Computer Graphics*, vol. 22, no. 4, August 1988.

[16] Subramanian, K. R., Fussell, D. S., "Applying space subdivision techniques to volume rendering," *Proc, Visualization*, 1990.

[17] Westover, L., "Interactive Volume Rendering," *Proceedings of the Chapel Hill Workshop on Volume Visualization*, May 1989.

[18] Westover, L., "SPLATTING: A Parallel, Feed-Forward Volume Rendering Algorithm," PhD thesis. University of North Carolina at Chapel Hill, 1991.

[19] Westover, L., "Footprint Evaluation for Volume Rendering," *Computer Graphics, Vol.24, No.* 4, August 1990.

[20] Wilhelms, J., Gelder, V. A., "Octree for fast isosurface generation," *ACM transactions on graphics*, July 1992.

[21] Yagel, R., Shi, Z., "Accelerating Volume Animation by Space Leaping," *Proc. Visualization*, 1993.

[22] Yagel, R., Machiraju, R., "Data-Parallel Volume rendering Algorithms," *The Visual Computer*, 1994.

[23] Yagel, R., Machiraju, R., "Efficient Feed-Forward Volume Rendering Techniques for Vector and Parallel Processors,"*SUPERCOMPUTING'93*, 1993.