A NEW PARALLEL VOLUME RENDERING ALGORITHM

Hyun Chin, R. S. Ramakrishna

Department of Information & Communication
KwangJu Institute of Science and Technology
500-712, 1 Oryong-dong Puk-ku , KwangJu
South Korea
e-mail : avatar93@geguri.kjist.ac.kr , rsr@kjist.ac.kr

ABSTRACT

On screen division of objects for parallel volume rendering is considered in this paper. The suggested algorithm runs on private-memory based parallel computers. The notable characteristic of the algorithm is that it effects data transfers only when it is absolutely necessary.

Keywords: parallel volume rendering, computer graphics, parallel algorithm, screen-divided volume rendering, object-divided volume rendering

1. INTRODUCTION

Volume rendering allows exploration of the inner structure of complex 3D data. It is a highly computation intensive job that demands huge storage space. A number of accelerating techniques have been reported in the literature [5][6][7]. Parallelization is known to be of great utility in volume rendering [5][6].
Volume rendering requires that care be exercised on several counts. Even though volume data are divided and rendered in parallel, the assembling process is beset with massive data transfer, especially on private-memory based parallel systems. Further, due attention must be paid to questions such as which part of volume data is related to which parts of image on the screen. All the nodes in private-memory based parallel computers [1][2] will have to communicate with each other in order to transfer volume data and rendered images if the volume data is divided and distributed among the local memories. [3].

A solution to these problems is presented in this paper. Volume data is divided and rendered independently at each node, in object-divided parallel volume rendering style [5][8][11]. Another approach is to divide the image into a number of small areas. Divided blocks of volume pertinent to a divided-screen area [9][10] will be delivered to a computing node that is assigned to render that area. By estimating the volume blocks corresponding to a small area of the screen, the system can send only the appropriate volume data to the nodes.
Communication cost is incurred only when the final rendered images are transferred.

2. DIVISION OF VOLUME INTO SCREEN-DIVIDED AREAS

Voxel gradient computation, resampling and compositing form the two major phases of volume rendering algorithms that demand massive computation. The first phase is independent of the rendering method – ray casting, splatting, etc. – and the second changes a little with the rendering method. A parallel volume rendering technique subdivides the volume object itself and then distributes the implied computations among processors. The suggested algorithm is based on the same premise.

The algorithm estimates the parts of volume data that correspond to a specific area on the screen through a sequence of actions. Before dividing the job and distributing the sub-jobs to all the nodes, the system must know the projection area (on the screen).

If the projection area of any two blocks has no overlap, then it is possible to render them independently. In that case, only the final image assembly (on the screen) requires significant communication. The basic idea of the algorithm is to reduce the communication by paying due attention to the partitioning of volume data.

The problem arises due to the uncertainty in the rendered area corresponding to each part of the volume object. It is difficult to know in advance where exactly they project.

Two questions may be raised in this regard:

1. Is it possible to estimate which part of volume object will be rendered to which part of image screen?
2. Can parts of the volume object be extracted exactly as the user – or requester – desires?

The proposed method reduces the need for transferring volume data while rendering, by seeking affirmative answers to the two questions. Volume data are divided into a number of small blocks so that it will be easy to extract them. These "micro-blocks" form atomic elements in appropriating volume objects to the nodes of the parallel computer. Also, the rendering screen is divided into a number of small rectangular "micro-areas". Micro-block is an atomic volume element (for dividing the volume), and micro-area is an atomic screen element (for dividing the screen).

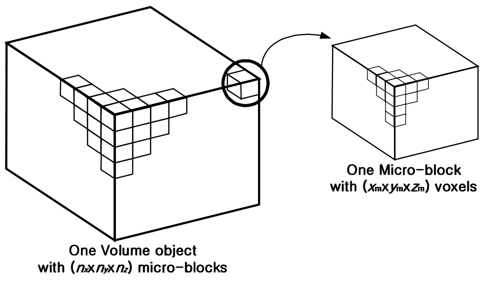

One Micro-block
with (x_mxy_mxz_m) voxels
One Volume object
with (n_xxn_yxn_z) micro-blocks

Figure 1. Division of Volume Object

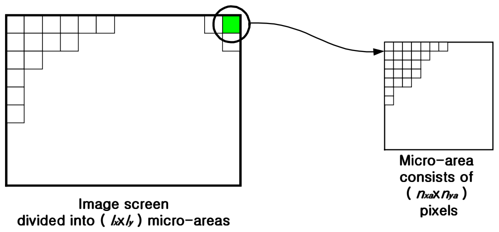


Figure 2. Division of Image Screen

Figures 1 and 2 show the division of a volume object into $n_x \times n_y \times n_z$ micro-blocks and the division of the screen into $l_x \times l_y$ micro-areas. Each micro-block has 8 vertices around their boundaries, and there are $(n_x+1) \times (n_y+1) \times (n_z+1)$ vertices in the volume with $n_x \times n_y \times n_z$ micro-blocks. Coordinate information about volume and camera enables the calculation of the projected positions of vertices on the screen. Eight projected vertices in a micro-block indicate the area to which the micro-block has to be projected. It is very difficult to calculate the exact projection area, but an approximation can be obtained by adopting the minimum and maximum x- and y-values on the screen. This is the rectangular area from (min_x, min_y) to (max_x, max_y), where min_x, min_y are the minimum x-, y-values, and max_x, max_y are the maximum x-, y-values corresponding to the 8 projected points. Figure 3 illustrates the idea.

The projection areas are recorded in a table, the key item being the "micro-block". The table indicates the areas corresponding to each micro-block. Since this "**Micro-block Projection Table**" (**MPT**) records every micro-block's rendering range, it is also possible to infer which micro-blocks are required to render which micro-area.

The "**Micro-Area Table**" (**MAT**) incorporates this information using micro-area as the primary key (Figure 4).

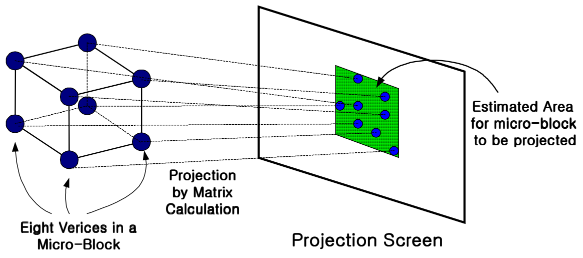
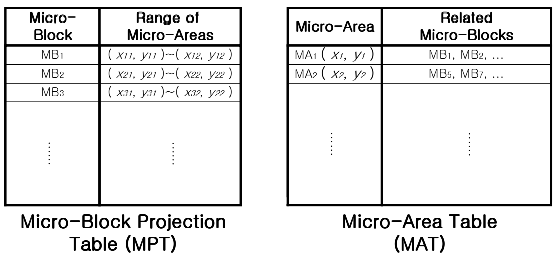


Figure 3. Estimation of Projection areas of a micro-block

The rendering system can extract just the required micro-blocks exactly to render a micro-area by referring to the MAT. MPT and MAT can be constructed with the spatial information about volume, camera and screen (position, direction, resolution, size, etc). The nodes in the parallel computer can build the MPT and MAT by receiving only this information. This is clearly quite insignificant in comparison with the size of the volume data.

Algorithm 1 describes the process of constructing the MPT & MAT.



Micro-Block	Range of Micro-Areas
MB1	(x_{11}, y_{11})~(x_{12}, y_{12})
MB2	(x_{21}, y_{21})~(x_{22}, y_{22})
MB3	(x_{31}, y_{31})~(x_{32}, y_{22})
⋮	⋮

Micro-Block Projection Table (MPT)

Micro-Area	Related Micro-Blocks
MA1 (x_1, y_1)	MB1, MB2, ...
MA2 (x_2, y_2)	MB5, MB7, ...
⋮	⋮

Micro-Area Table (MAT)

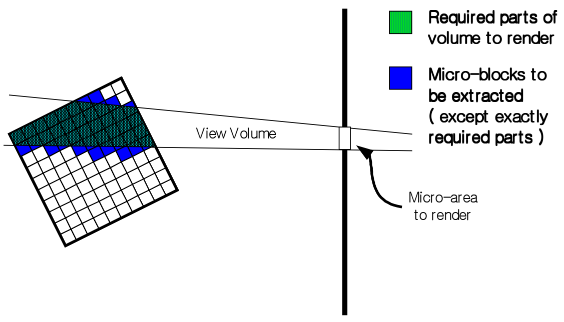
Figure 4. Examples of the form of MPT & MAT

Figure 5 shows the extraction of micro-blocks when a micro-area is specified. Because of atomicity, some micro-blocks would be required even if all the contents in the micro-block are not used for rendering the micro-area.

```
Divide the volume into $n_x \times n_y \times n_z$ micro-blocks
Divide the screen into $l_x \times l_y$ micro-areas
Calculate the position of $(n_x+1)\times(n_y+1)\times(n_z+1)$ vertices in world
Project vertices to the screen and memorize projected positions
for ( 0, 0, 0 ≤ x, y, z < $n_x$, $n_,$, $n_z$) do
    MPT( x, y, z ) ← corresponding range of micro-area
                     range of ( x, y, z ) ~ ( x+1, y+1, z+1 ) vertex
endfor

for ( 0, 0, 0 ≤ x, y, z < $n_x$, $n_,$, $n_z$) do
    start_x, start_y, end_x, end_y ← MPT( x, y, z )
    for ( start_x, start_y ≤ i, j ≤ end_x, end_y ) do
        add ( MAT( i, j ), MB( x, y, z ) )
    endfor
endfor
```

Algorithm 1. Construction of MPT & MAT



🟩 Required parts of volume to render
🟦 Micro-blocks to be extracted (except exactly required parts)
View Volume
Micro-area to render

Figure 5. Required parts and extracted micro-blocks to render micro-area

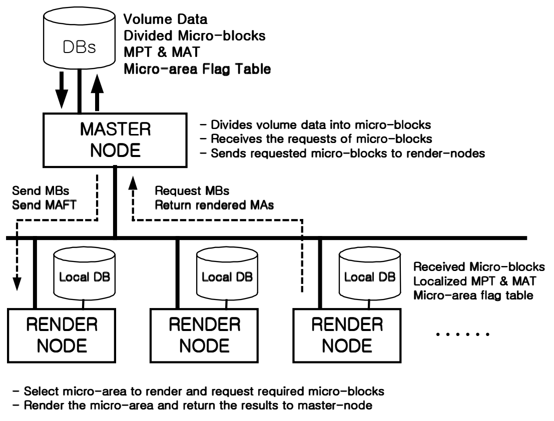
The image in a MA can be rendered correctly only with the MBs indicated in the MAT. Any of the extant volume rendering methods – ray casting, splatting, etc. – can be employed as the basic renderer for each MB. The rendering module is therefore, independent of the screen-division technique proposed in this paper.

3. PARALLELIZATION USING MPT & MAT

MPT and MAT describe the required micro-blocks in a volume object to render a micro-area on the screen. The architecture of the renderer and the corresponding algorithm that employs the two tables will be addressed in this section.

3.1 Architecture of parallel renderer

Figure 6 shows the block diagram of the parallel rendering system. The nodes are classified into two types. The "**master-node**" plays the role of storing and distributing the volume data. It divides the volume object into a number of micro-blocks and stores them with their indices. Before the start of the rendering process, the master-node sends the information about the volume object and micro-blocks (spatial information, and divided numbers) to all the nodes. Gathering the rendered micro-areas from the nodes form part of the master-node's job. The "**render-node**" determines the micro-area to be rendered, and requests the master-node for the corresponding micro-blocks.



DBs
Volume Data
Divided Micro-blocks
MPT & MAT
Micro-area Flag Table
MASTER NODE
- Divides volume data into micro-blocks
- Receives the requests of micro-blocks
- Sends requested micro-blocks to render-nodes
Send MBs
Send MAFT
Request MBs
Return rendered MAs
Local DB
Local DB
Local DB
Received Micro-blocks
Localized MPT & MAT
Micro-area flag table
RENDER NODE
RENDER NODE
RENDER NODE
......
- Select micro-area to render and request required micro-blocks
- Render the micro-area and return the results to master-node

Figure 6. Block Diagram of Parallel Renderer

Render-node receives the information on volume object, micro-blocks, micro-areas, and camera before the start of the rendering process and builds the MPT and the MAT according to the received information. After a render-node determines a micro-area, it refers to the MAT and requests the master-node for the appropriate micro-blocks. The master-node receives the request and sends the requested micro-blocks to the render-node. Rendering of micro-area proceeds in the render-node after the transfer of micro-blocks and the rendered image of micro-area is returned to the master-node.

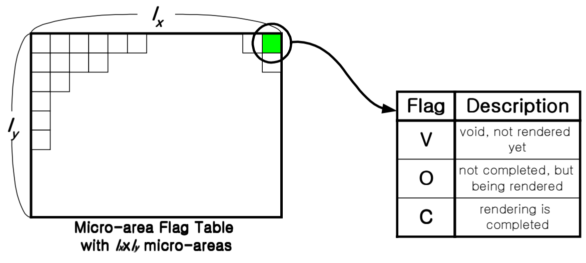
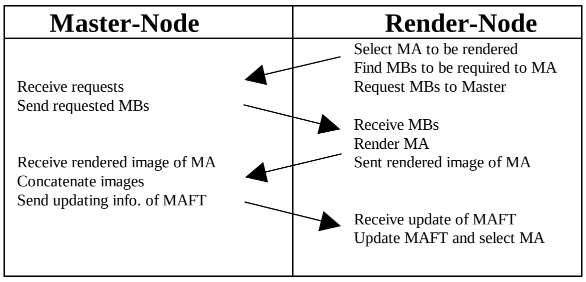


Figure 7. Micro-area Flag Table (**MAFT**)

Every node has MPT, MAT, and the flag table indicating which micro-area is completed and which is not. It is called a "**micro-area flag table**". There are three flags, viz., **V**, **O**, and **C**. Flag "V" means that the micro-area is not rendered yet, "O" indicates that the micro-area is being rendered by the render-node, and a micro-area with flag "C" indicates that it has already been rendered. All the flags in the micro-area flag table are initialized to "V". When the master-node receives a request from a render-node, it modifies the flag in the micro-area flag table to "O" and sends the contents of the table with the requested micro-blocks. Flag "C" is entered into the micro-area flag table after the rendered micro-area is received from the render-node. The master-node directs the render-nodes to update the micro-area flag table when the render-nodes return the completed micro-areas to it. This updating may be done in two ways: (a) the entire micro-area flag table may be sent to the render-nodes, and (b) only the updating information may be sent. The first seems to require more communication, and the second requires the replication of micro-area flag tables of all the render-nodes at the master-node. Render-nodes modify micro-area flag table locally and select a micro-area and request the required micro-blocs again until there is no "V" flag in the micro-area flag table.



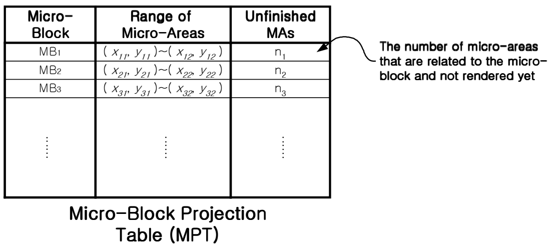
Master-Node	Render-Node
	Select MA to be rendered Find MBs to be required to MA Request MBs to Master
Receive requests Send requested MBs	
	Receive MBs Render MA Sent rendered image of MA
Receive rendered image of MA Concatenate images Send updating info. of MAFT	
	Receive update of MAFT Update MAFT and select MA

Algorithm 2. Execution between master-node and Render nodes

3.2 Selection policy of micro-area in render-node

Render-nodes select micro-areas on their own before starting the rendering process. Some selection policy is clearly necessary in this regard. While selecting the next micro-area to be rendered, the following points should be considered:

- Redundant transfer of micro-blocks has to be avoided;
- The micro-area that can use the micro-blocks already received and that need not request the master-node for excessive micro-blocks should be selected.
- Furthermore, overlapping selection between render-nodes has to be prevented.

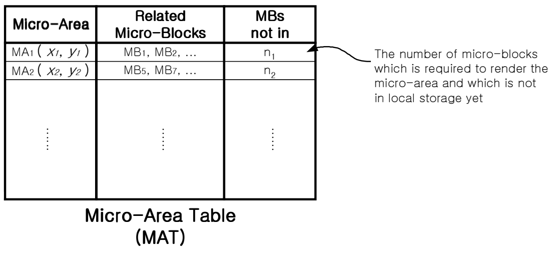


Micro-Block	Range of Micro-Areas	Unfinished MAs
MB1	$(x_{11}, y_{11})\sim(x_{12}, y_{12})$	n_1
MB2	$(x_{21}, y_{21})\sim(x_{22}, y_{22})$	n_2
MB3	$(x_{31}, y_{31})\sim(x_{32}, y_{32})$	n_3
⋮	⋮	⋮

The number of micro-areas that are related to the micro-block and not rendered yet
Micro-Block Projection Table (MPT)

Figure 8. MPT in each render-nodes

Figure 8 and 9 show the MPT and MAT of each render-node. Additional items are inserted into the MPT and MAT to cope with these issues. "**MBs not in**" in MAT refers to the number of micro-blocks that have not been received by the master-node as yet. At the time of initialization (of render-nodes), the values of all the 'MBs not in' are set equal to the number of required micro-blocks (in MAT). When the master-node sends the requested micro-blocks to a render-node, the latter refers to the MPT and finds the micro-areas corresponding to the received micro-blocks. The values of 'MBs not in' in MAT are modified at the reception of micro-blocks, because those micro-blocks are not 'not in local' in the render-node any more.



Micro-Area	Related Micro-Blocks	MBs not in
MA1 (x_1, y_1)	MB1, MB2, ...	n_1
MA2 (x_2, y_2)	MB5, MB7, ...	n_2
⋮	⋮	⋮

The number of micro-blocks which is required to render the micro-area and which is not in local storage yet
Micro-Area Table (MAT)

Figure 9. MAT in each render-nodes

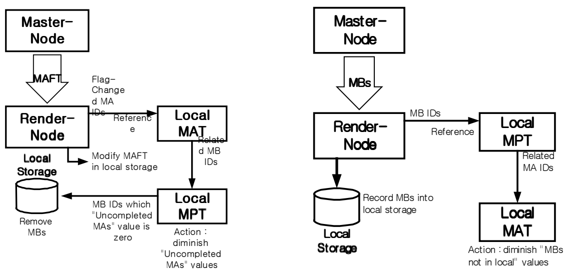


Figure 10. Modification of MPT and MAT when receiving MBs and MAFT

When the occupied micro-area is rendered, it is also required to remove the micro-blocks from local memory if the micro-blocks are not required any more. The item "**Unfinished MAs**" in MPT is used for the purpose. As with the "MBs not in", the value of "Unfinished MAs" in MPT is the number of micro-areas in MPT at the time of initialization. Render-node modifies the value of 'Unfinished MAs' in MPT when it receives MAFT from the master-node. A micro-block whose "Related MAs" in MPT becomes zero is removed from local memory of the render-node because it will not be required any more.

The next micro-area selection is determined by referring to MAT. The value of "MBs not in" is the number of micro-blocks to be imported from the master-node. The micro-area that has the smallest "MBs not in" value in MAT is selected as the next micro-area; this requires the transfer of the smallest number of micro-blocks for rendering. Micro-blocks in the render-node and the newly received ones are used for rendering the occupied micro-areas: unnecessary micro-blocks are trashed.
Algorithm 3 implements these policies.

```
MBs received :
  ; Updating MAT
  for  mb_id in received MBs   do
      ma_ids← MPT(mb_id)
      for  ma_id in ma_ids   do
          diminish ( MAT(ma_id).MBs_not_in )
      endfor
  end for

MAFT update received :
  ; Updating MPT
  for  ma_id in MAs updated to C or O   do
      mb_ids←MAT(ma_id)
      for  mb_id in mb_ids  do
          diminish ( MPT(mb_id).Unfinished_MAs )
          if ( MPT(mb_id).Unfinished_MAs = 0 )  then
              remove MPT(mb_id)
          endif
      endfor
  end for

Selection of next MA to be rendered
  ; after receiving MAFT updating info.
  min_val ← ∞, min_id ←-1;
  for ma_id in all MAs which MAFT(ma_id) is void
      if  min_val > MAT(ma_id).MBs_not_in  then
          min_val = MAT(ma_id).MBs_not_in
          min_id = ma_id
      endif
  endfor

  Select ma_id to next MA
```

Algorithm 3. Basic Policies for render-nodes for selection of micro-areas

4. CONSIDERATIONS

Each render-node executes independently with the same policy of selection, but the selected micro-areas have to be distinct. The factor that enables distinct selections lies in the first selected micro-areas in each render-node. The first selection in render-nodes should be influenced by factors such as spatial position of micro-areas on the screen. Also, the selection policy after that should be influenced a little by spatial factors to reduce coincident selections.
Selection policy must consider another problem: conflicting (selected) micro-areas. This is under investigation.

5. PROGRESS IN IMPLEMENTATION

The parallel volume rendering system that adopts the suggested method is being implemented on a platform of network-connected cluster of work-stations. The core routines that play a role in dividing the volume into MBs, building MPT and MAT, and rendering thereafter have already been developed. Figure 10 shows a sample image that has been rendered by this system.

The rendering software is written in C for portability across UNIX based computers.

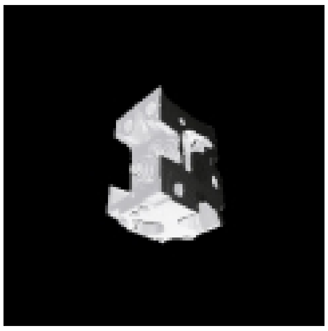


Figure 10. Example image rendered by core routines

The other components (to be developed) are: job-division module, communication modules, and load balancing control modules. MPI-based API functions will be employed for clustering. Data and control instructions will be formatted as messages in the system. MA-selection module in every render node will play a role in job-division and load balancing. They are presently being implemented along with the communication module. Fully integrated rendering system is expected to be in place soon.

6. CONCLUSION

This paper has suggested a screen-division based volume rendering algorithm that reduces communic-ation overheads as much as possible. It incurs a lower communication cost than other existing parallel volume rendering algorithms. A volume rendering system is being developed presently with a view to adopt this method to run on multi-computer based cluster machines. The system communicates via TCP/IP protocol using PVM [4] or MPI. The suggested technique transfers only the required volume data to the other nodes by dividing the volume object into micro-blocks and by dividing image screen into micro-areas.

The main advantage of the proposed method lies in the fact that each node transfers only the volume data and parts of rendered image. For example, the well-known parallel volume renderer suggested by Greg Johnson [11] requires the exchange of pre-composited images among rendering nodes in order to finalize the overall image. The method elaborated in this paper completes all the rendering processes of one MA in the local render node, and only the rendered MAs (as final results) are sent to the master-node. This policy reduces the communication overheads by a significant measure.

Massive volume data characterises some of the applications such as MRI imaging. Parallel volume rendering techniques are expected to offer feasible solutions to such problems. A technique with a reasonable communications overhead will be clearly a winner. It is hoped that the suggested algorithm will be among the winners.

REFERENCES

[1] A. Kaufman, R. Bakalash, D. Cohen and R.Yagel. A survey of architectures for volume rendering. IEEE Engineering In Medicine And Biology, pages 18-23, Dec. 1990

[2] Christopher Giertsen, Johnny Peterson, Parallel Volume Rendering on a Network of Workstations, November, 1993, IEEE Computer Graphics and Applications, pp. 16-13

[3] Michael E. Palmer, Stephen Taylor, Brian Totty, Exploiting Deep Parallel Memory Hierarchies for Ray Casting Volume Rendering, October, 1997, Proceedings of the IEEE symposium on Parallel Computing, pp. 15-22

[4] Clemens H. Cap, Volker Strumpen, Efficient Parallel Computing in Distributed Workstation Environments, Parallel Computing, 19(1993), pp. 1221-1234

[5] Craig M. Wittenbrink, Survey of Parallel Volume Rendering Algorithms, Parallel and Distributed Processing Techniques and Applications, July, 1998, pp. 1329-1336

[6] T. T. Elvins, A survey of algorithms for volume visualization, Computer Graphics, 26(3), pp. 194-201

[7] A. Van Gelder and K. Kim, Direct Volume Rendering with Shading via 3D Textures, ACM/IEEE Symposium on Volume Visualization, October 1996, pp. 23-30

[8] Craig M. Wittenbrink, M. Harrington, A Scalable MIMD Volume Rendering Algorithm, Eighth International Parallel Processing Symposium, April, 1994, pp. 916-920

[9] Corrie B., Mackerras, Data Coherences in Volume Rendering Algorithm", Proceddings of 1993 Parallel Rendering Symposium, San Jose, CA, 1993, pp. 23-26

[10] Schroder. P, and Krueger, Data Parallel Volume Rendering Algorithm for Interactiove Visualization, The Visual Computer, 9, 1993, pp. 405-416

[11] Greg Johnson, Volume Rendering of Large Datasets on the Cray T3D, 1996 Spring Proceedings of Cray User Group, 1996, pp 155-159