Virtual Animation of Tomographic Volumes on Multiprocessors

J.R. Cozar
R. Asenjo
E.L. Zapata

April 1997
Technical Report No: UMA-DAC-97/09

Published in:

VII Nat'l Symp. on Pattern Recognition and Image Analysis
Bellaterra (Barcelona), April 1997, pp. 329-334

University of Malaga
Department of Computer Architecture
C. Tecnologico • PO Box 4114 • E-29080 Malaga • Spain

Virtual Animation of Tomographic Volumes on Multiprocessors

J.R. Cózar, R. Asenjo, E.L. Zapata
Dept. of Computer Architecture,
Complejo Politécnico
Universidad de Málaga
Apdo. 4114
E-29080 Málaga, Spain
e-mail: {julian, asenjo, ezapata}@ac.uma.es

Abstract

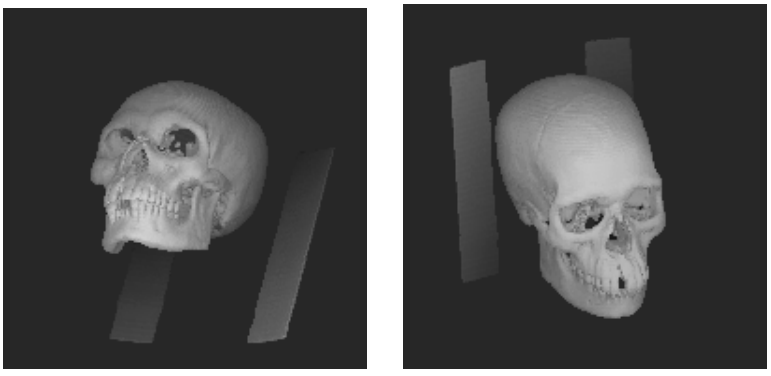
Three-dimensional Representation of the human body organs from data obtained by Computerized Tomography (CT) is a technique used in radiologic diagnosis. In this paper we present a technique for the generation of these views using massively parallel computers through the renderization of surfaces defined by *voxel faces*. The algorithms developed are generic in the sense that they impose no restrictions to the problem size and they are independent of the number of processors. The efficiency of our parallel algorithms has been tested on a multiprocessor system based on the i860 processor, and a comparative study of the results obtained with the Blocks, Cyclic, and Multiple Recursive Decomposition data distributions is shown.

Key Words : Rendering, Parallel Processing, MRD, 3DVIEWNIX.

1 Introduction

The pictures obtained via CT [1] are a set of slices depicting intensity distributions in a given region of the human body along a set of parallel planes. Using segmentation techniques, we can extract those 3D objects of interest. Visualization of these objects is improved if we apply 3D illumination, rotation and opacity variation techniques.

The aim of this work is the generation of sequences of 3D objects' views from data provided by CT, see Fig.1. Generation of these views requires a large amount of memory and computational power, so, if the results are to be obtained in any reasonable period of time, use of parallelism is mandatory.



(a) (b)

Figure 1: Example of an Human Skull Views

We have developed a parallel algorithm for generating the sequence of 3D views on message passing parallel architectures with distributed memory. In order to find out which data distribution is best suited for this problem the Blocks, Cyclic and Multiple Recursive Decomposition data distributions have been implemented, and a comparative study of the results is presented.

As a starting point, sequential codes were extracted from *3DVIENIX* [2], a software package that provides a variety of sophisticated methods for visualizing, manipulating, and analyzing image data. Furthermore, we use its environment to implement the user's interface.

Section 2 of this paper is devoted to the study of the sequential algorithm, giving a brief explanation of the rendering process. In section 3, the parallelization of this algorithm is presented. Finally, in section 4, an evaluation of the efficiency or our algorithms is done, and comparative results for the several data distributions tried are shown.

2 The Sequential Algorithm

2.1 The Rendering Process

This section briefly overviews the rendering process. *Rendering* is a jargon word that has come to mean *"The collection of operations necessary to project a view of an object or a scene onto a view surface "*. The object is lit and its interaction with a light source is calculated to produce a shaded version of the scene.

The best way to break down the overall rendering process is to consider an object description through a number of coordinate spaces. In each space, operations are carried out. The different coordinate spaces facilitate certain process and specifications.

To build up a scene, objects specified in a *modeling coordinate system* (Fig.2(a)) may have three-dimensional transformations applied to them. This embeds the object in a *word coordinate space*, a space common to all objects in a scene (Fig.2(b)). In this space we also establish the position of the light source, or sources, that illuminate the scene and a view reference.

The object is transformated into the *view* or *eye coordinate space* (Fig.2(c)) and it is in this space that the various specifications concerning the view are made.

In the eye space two operations are performed: the *culling* (Fig.2(d)) and the *hidden surface removal*, Fig.2(e). The *culling* operation only considers polygons in their entirety for each object while the *hidden surface removal* resolve the overlapping object problem.

The final two operations carried out on the polygons are *shading* and *rasterization* (Figs.2(f) and (g)) usually in the *three-dimensional screen space*. *Shading* compares the orientation of each polygon with the light source direction and assigns a shade to the interior of the polygon. *Rasterization* works out which value to allot to each *pixel* in the projection.

3 Parallelization of the Algorithm

Recently, numerous parallel algorithms have been developed for rendering. The most used strategy in the parallel rendering algorithms is the *divide and conquer* paradigm. The subdivisions can be made either in the object domain or in the image domain. The method we use is hybrid because we divide the data domain (during the rendering) as well as the image domain (during the composition).

The key idea of the algorithm we present here is very simple: it divides the data in smaller sub-volumes and distributes it among processors, renders them locally, and finally combines the resulting images. The memory requirements are modest since each processor contains only a subset of the global data. In this section we present the more frequently adopted alternatives for each stage in the parallel rendering process, justifying the ones choosed by us.

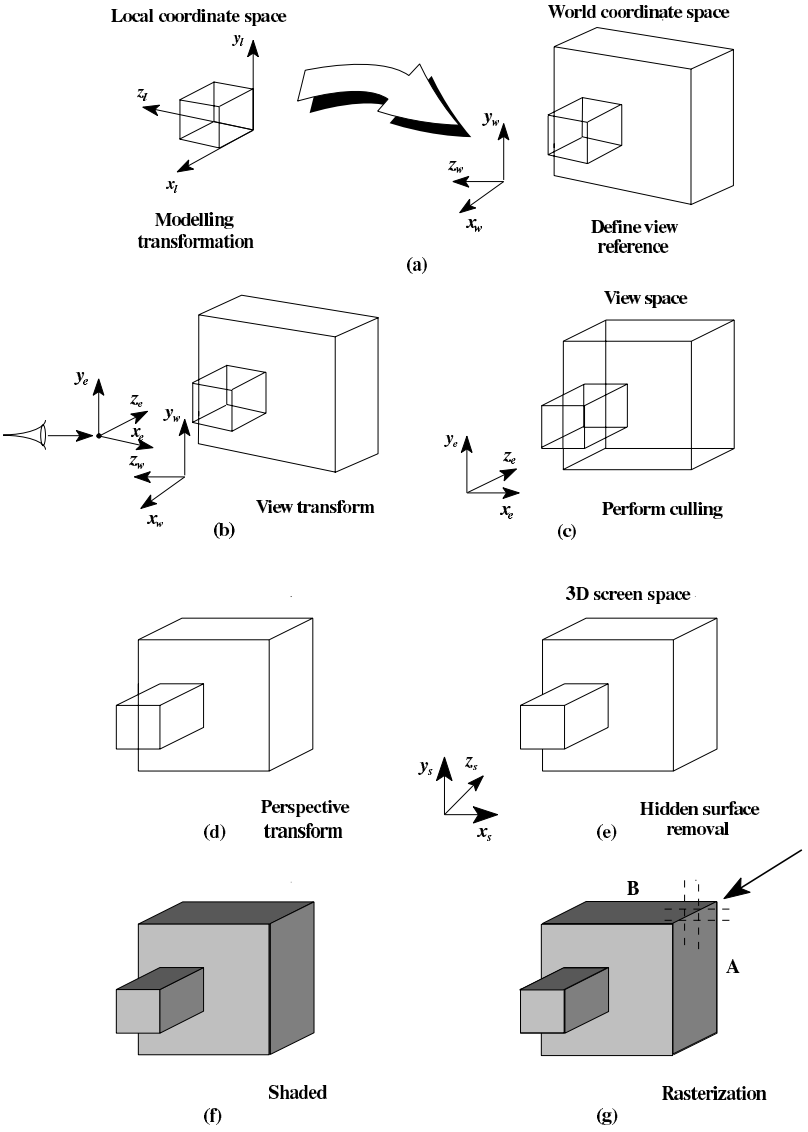


Figure 2: *Stages in the Rendering Process*

3.1 Data Partitioning and Load Balancing

There are lots of way to carry out this. For example, Neumann [3] compared the three basic data distributions derived from splitting the volume with one, two or three perpendicular planes respectively. We partitioned only one dimension (*slices*) since using two unnecessarily complicate the algorithms without a great improvement. The storage of the sparse volume is carried out through a compressed data structure, so using three dimensions is unfeasible.

3.2 Local Rendering of the Volume

Each processor renders its sub-volume locally, i.e., no communications are required during sub-volume processing. Among the many rendering algorithms [4], we use a *voxel projection* algorithm because of the initial data we have. In our parallel rendering algorithm we adopted the paradigm *host/node*. The *host*, besides preparing and distributing the work among *nodes* and collecting the results, controls the graphic user interface. The *nodes* carry out the renderization process itself. The way they create each frame is the following:

1. Receive data from Host
2. For each photogram in the sequence
  2.1 For each opaque object
    2.1.1 Process the object (to rotate & to project)
    2.1.2 Compose the opaque buffers in Node 0
    2.1.3 Any translucent object ? $\Longrightarrow$ Broadcast opaque buffers
  2.2 For each translucent object
    2.2.1 Process the object (to rotate & to project)
    2.2.2 Compose the translucent buffers in Node 0
    2.2.3 Am I Node 0 ? $\Longrightarrow$ send the photogram to Host

Opaque objects must be processed first because they can occlude the translucent objects.

3.3 Compounding the Image

The last stage of the algorithm is the composition of all the partial views computed by the nodes in a final image. The most used approaches are:

- *Direct send*: Neumann [3] use this approximation. Each computed *pixel* is directly sent to the processor that has been assigned the image plane portion where the pixel is located.
- *Binary Composition*: in each stage the processors are paired to form a new subimage according to a binary tree structure. The final image is obtained after $\log n$ stages. One problem with this method is that the processors become inactive during the composition process.
- *Binary swap composition* [5]: we can exploit even more the parallelism if we generalize the foregoing method. In each step of the reduction phase, the two processors involved in the composition operation divide the image plane in two parts and each processor is responsible of one of them.

We used the *binary composition* because to take advantage of the *binary swap composition*, the composition procedure must be complex.

The sparsity of the image data can be exploited even more if the composition is carried out only on the portion of the image that is not background. For this purpose, each processor must keep a rectangle, aligned with the screen, that delimits this area in the image.

4 Evaluation

4.1 Complexity

A generic formula that give us the complexity per photogram is:

$$O\left[opaq \cdot \frac{N^3}{n} + trans \cdot \frac{N^3}{n} + (a+b+c) \cdot \log n\right]$$

where the two first terms correspond to computations and the last one to communications. In this formula, *opaq* and *trans* are the number of opaque and translucent objects respectively. N^3 represents the three-dimensionality of the objects. *a, b* and *c* are communication time constants and are non null when we have opaque, translucent and both objects respectively. Finally, *n* is the number of processor elements (PEs).

- The *computational* terms mainly depends on the number of objects, if they are opaque and/or translucent and its magnification, how many elements are used in the definition of the surfaces, viewer's position, and number of PEs.

Node	Blocks (40563) %	Cyclic (4211) %	MDR(1D) (2337) %
0	13,5	25,2	25
1	34,7	24,9	24,6
2	28,8	23,2	24,7
3	23	25	25,7

Table 1: *Comparison of the balancing obtained with each distribution*

- The communications depends in a very sophisticated way on the class of the objects we have, its shape and viewer's position (determining the area of the rectangle that delimitates the projection), frame size and PEs number (logarithmically).

The expression for the complexity when $n = 1$ corresponds with the sequential case, since the parallel algorithm is a generalization of the first one.

4.2 Load Balancing

In Table 1 we show a comparison between the balancing obtained with the different data distributions we've used. The given values are for a surface that represents the human skull (defined by 518.894 *voxel faces*), when we use four nodes. For each node the percentage of elements it receives for each distribution is indicated. To clear the goodness of a distribution, the standard deviation in the number of received elements per node is also included (in brackets).

In the table we can see how the MRD distribution is the best in balancing the use of the memory. The cyclic, however, is the best balancing the computational load. The surfaces are defined by three classes of voxel faces and not always all kinds come into computation. If we take alternative *slices*, the probability of their elements belonging to different classes is higher. So, the different kinds of voxel faces are more uniformity distributed among processors.

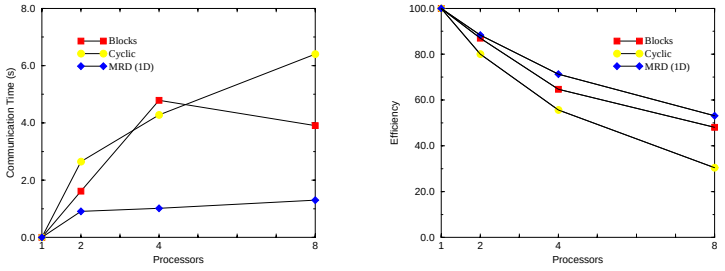
4.3 Comparison between Times for the Different Distributions

The times we present here were measured on the parallel supercomputer Paramid $16 \times i860$ [6]. It is composed by 16 nodes or TTM200 boards, all of them consisting in an Intel i860-XP processor for the computation purposes and an Inmos T805 transputer for the computations and/or communications ones.

In Fig.3 two graphics that summarize the results obtained for the different distributions are shown. The time is for the generation of a sequence of frames of the human skull. Since the time for the creation of a photogram is not constant, we've taken its average value. The size of the frame is 256×256 pixels. The number of processors we've used is a power of two -1, 2, 4, and 8- so we can obtain the maximum efficiency in the communication phases.

4.3.1 Communications

The general trend in the communication time is to grow with the number of processors (in a logarithmic way) and with the size of the frame. The peak that appears in the blocks distribution is due to load unbalancing. In the consecutive distributions (blocks and MRD) the area of the rectangle that contains a portion of image not being background is smaller, and a lesser number of messages are sent since elements of the volume that are placed near will be close in the projection.



(a) Communication Time (b) Efficiencies

Figure 3: Performance of our Parallel Algorithm

4.3.2 Efficiency

The global efficiency of the algorithm decreases when we add new processors (increasing the volume of communications) and with the load unbalancing. So, the best efficiency is achieved with the MRD distribution (less communication costs and unbalancing). By the other hand, consecutive distributions (blocks and MRD) results in a more efficient use of the cache since they hold a higher spatial locality when the elements are projected.

Finally we note that the greater number of elements is used in the definition of an object surface, the better efficiency is achieved.

5 Conclusions

In this work we have studied the behavior of the blocks, cyclic and MRD distributions in a global sense for this particular type of problems. The best results were obtained using the MRD distribution and the worst with the cyclic distribution. The blocks partition is between then, closer to the MRD than to the cyclic. In fact, the MRD distribution is a generalization of the blocks distribution that gets optimal load balancing when we work with sparse data structures.

References

[1] G.T. Herman. *Image Reconstruction from Projections. The Fundamentals of Computerized Tomography*. Academic Press, 1980. ls

[2] J. Udupa, R. Goncalves, K.I. Narendula, D. Odhner, S. Samarasekera y S. Sharma. *3DVIEWNIX: An Open, Transportable Software System for the Visualization and Analysis of Multidimensional, Multimodality, Multiparametric Images*. SPIE Proccedings, Vol.1897, 1991, pp. 47-58.

[3] U. Neumann. *Parallel Volume-Rendering Algorithm Performance on Mesh Conected Multicomputers*. Proc. Parallel Rendering Symp., ACM, New York, 1993, pp. 253-259.

[4] A. Watt. *3D Computers Graphics*. Ed. Addison-Wesley, 2nd edition, 1994.

[5] K. Ma, J.S. Painter, C.D. Hansen y M.F. Krogh. *Parallel Volume Rendering Using Binary-Swap Compositing*. Computer Graphics and Aplications, Vol.14, n.4, 1994, pp. 59-68.

[6] R. Asenjo, M. Ujaldón. *Manual Básico del Supercomputador Paralelo Paramid*. Dpto. de Arquitectura de Computadores, Universidad de Málaga, 1993.