

**COMPARISON OF IMAGE SPACE
SUBDIVISION ALGORITHMS FOR
PARALLEL VOLUME RENDERING**

A THESIS

**SUBMITTED TO THE DEPARTMENT OF COMPUTER
ENGINEERING AND INFORMATION SCIENCE
AND THE INSTITUTE OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE**

By

Egemen Tanin

July, 1995

Thesis

T

385

.T36

1995

COMPARISON OF IMAGE SPACE SUBDIVISION ALGORITHMS FOR PARALLEL VOLUME RENDERING

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER
ENGINEERING AND INFORMATION SCIENCE
AND THE INSTITUTE OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE


By
Egemen Tanin
July, 1995

Egemen Tanin
tarafından onaylanmıştır


1
385
.T36
1985

B031601

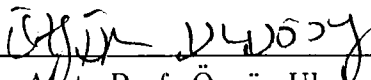
I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.


Assoc. Prof. Cevdet Aykanat (Advisor)

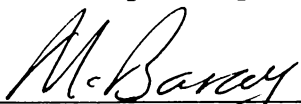
I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.


Prof. Bülent Özgüç (Co-advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.


Asst. Prof. Özgür Ulusoy

Approved for the Institute of Engineering and Science:


Prof. Mehmet Baray
Director of the Institute

ABSTRACT

COMPARISON OF IMAGE SPACE SUBDIVISION ALGORITHMS FOR PARALLEL VOLUME RENDERING

Egemen Tanin

M.S. in Computer Engineering and Information Science

Advisor: Assoc. Prof. Cevdet Aykanat

July, 1995

In many scientific applications, results are presented as *unstructured volumetric data sets*. *Direct Volume Rendering* (DVR) is a powerful way of visualizing these volumetric data sets. However, it involves intensive computations. In addition, most of the volumetric data sets also require huge memories. Hence, DVR is a good candidate for parallelization on *distributed memory multicomputers*. Also most of the engineering simulations are done on multicomputers. Therefore, visualization of these results on the same architectures where simulations are done avoids the overhead of transporting large amount of data. In order to visualize unstructured volumetric data sets, the underlying algorithms should resolve the *point location* and the *view sort* problems of the 3D grid points. In this thesis, these problems are solved by using the well-known *Scanline Z-Buffer* algorithm. Three image space subdivision algorithms, namely *horizontal*, *rectangular*, and *recursive* subdivisions, are utilized to distribute the computations evenly among the processors in the rendering phase. The main parallel algorithm uses *Raycasting* approach of DVR to visualize the data sets, which is also an image space method. Therefore, the divisions are made in order to obtain a set of sub-images. *Static task decomposition* is used where each processor is assigned to a single sub-image. The *load balance* among the processors is achieved by defining the overall work load within a sub-image by using the milestone operations done in the Scanline Z-Buffer algorithm. The algorithms are developed in a way that they can handle any kind of polygonal, volumetric, and etc. data set where the underlying architecture is also kept

flexible in many aspects for the sake of generality and portability. The experimental performance evaluation of the horizontal, rectangular, and recursive subdivision algorithms on an *IBM-SP2* system are presented and discussed in a comparative way.

Keywords: Direct volume rendering, computer graphics, parallel algorithms, distributed memory multicomputers.

ÖZET

EKRAN UZAYINDA BÖLME YÖNTEMLERİNİN PARALEL HACİM GÖRÜNTÜLEME AMACIYLA KARŞILAŞTIRMALI İNCELENMESİ

Egemen Tanin

Bilgisayar ve Enformatik Mühendisliği, Yüksek Lisans

Danışman: Doç. Dr. Cevdet Aykanat

Temmuz, 1995

Birçok mühendislik uygulamalarında, elde edilen sonuçlar *yapısal olmayan hacimsel veri kümeleri* olarak saklanmaktadır. *Doğrudan Hacim Görüntüleme* (DHG) yöntemleri bu amaçla kullanılan en etkin görüntüleme tekniklerinden biridir. Ancak bu yöntemler oldukça yoğun işlemler sonucunda istenilen görüntüyü elde edebilmekte ve dolayısıyla animasyon ve benzeri uygulamalar için oldukça yavaş yöntemler olarak kabul edilmektedirler. Ayrıca hacimsel veriler çok büyük bilgisayar bellekleri kullanılarak saklanabilmekte ve bu açıdanda görüntüleme işlemi oldukça zorlaştırmaktadırlar. Bunlara ek olarak uygulamaların çoğunluğu *çok işlemcili dağıtık hafızalı* bilgisayarlarda yapılmaktadır. Dolayısıyla büyük veri kümelerinin görüntüleme amaçlı bilgisayarlara taşınması büyük sorunlar doğurabilmektedir. İşte bütün bu nedenlerden dolayıdır ki *Paralel-DHG* (P-DHG) önemli bir araştırma konusu olmuştur. Fakat DHG yöntemlerinin yapısal olmayan hacimlerde uygulanması *nokta yeri tespiti* ve *bakış açısı sıralaması* adı verilen iki problemin çözümünü gerektirmektedir. Bu tezde standart *poligon boyama* yöntemleri kullanılarak bu problemlere çözüm aranmış ve paralelleştirmek amacıyla üç yöntem ileri sürülmüştür. Önerilen ana paralel algoritma *Işın Düşürme* yönteminin kullanılması yoluyla görüntüleme yapılması esasını kullanmakta ve ileri sürülen bu üç yöntem gibi ekran uzayını baz olarak almaktadır. Dolayısıyla iş bölümü ekran uzayının daha küçük ekran parçacıklarına bölünmesi ile gerçekleştirilmektedir. İş dağılımı her bir paralel işlemci başına tek bir ekran

bölümü düşecek şekilde *statik* dağılım yapılarak gerçekleştirilmiştir. Kullanılan poligon boyama yönteminin ana bölümleri göz önüne alınarak bir ekran bölümündeki iş hesaplanmış ve mümkün olduğunca işlemcilere eşit *iş dağılımı* yapılmaya çalışılmıştır. Geliştirilen programlar her türlü sistem ve veri kümesi kullanabilecek şekilde genelleştirilmiş ve bu şekilde gerçekleştirilmişlerdir. *Yatay, dikdörtgenel, ve özyineli* adı verilen bölme yöntemlerinin *IBM-SP2* sisteminde karşılaştırmalı incelenmesi yapılmış ve bu tezle birlikte sunulmuştur.

Anahtar Sözcükler: Doğrudan hacim görüntüleme, bilgisayar grafikleri, paralel algoritmalar, dağıtık hafızalı çok işlemcili bilgisayarlar.

ACKNOWLEDGEMENTS

I am very grateful to my advisor Assoc. Prof. Cevdet Aykanat for his guidance, suggestions, and encouragement throughout the development of this thesis. I would like to thank Prof. Bülent Özgüç for reading and commenting on the thesis. I would also like to thank Asst. Prof. Özgür Ulusoy for reading and commenting on this thesis. Finally, thanks to Tahsin M. Kurç for his invaluable effort in the development of the thesis.

Bu alıřmamı,
herřeyimi borlu olduėum *anneme, babama,*
ve
kardeřime
adıyorum.

Contents

1	Introduction	1
1.1	Overview of the Visualization Process	1
1.2	Related Work	3
1.3	Motivation and Overview of the Thesis	4
2	Sequential Direct Volume Rendering	6
2.1	Direct Volume Rendering in General	6
2.2	The Proposed Sequential Approach	15
3	Parallel Direct Volume Rendering	22
3.1	Our Parallel Implementations in General	22
3.2	Subdivision Heuristics	24
3.2.1	Horizontal Heuristic	25
3.2.2	Rectangular Heuristic	29
3.2.3	Recursive Heuristic	31
3.3	Load Balancing Metrics	33
3.4	Refining the Algorithms	35

<i>CONTENTS</i>	x
4 Results	38
4.1 Introduction to the Experimental Results	38
4.2 Experimental Results	44
5 Conclusion	50
A Tables of the Sample Runs	53

List of Figures

2.1	Data types used in volume rendering.	7
2.2	Flow chart of the sampling process.	8
2.3	Volume rendering overview.	9
2.4	Overview of the volume rendering for unstructured data sets (2D illustration).	11
2.5	The general DVR algorithm.	12
2.6	The interpolation process for different dimensions.	13
2.7	Flow chart of our sequential algorithm.	17
2.8	Overview of our sequential process.	18
2.9	Data from University of North Carolina Chapel Hill. A Computer Tomography image of a cadaver head.	20
2.10	Data from NASA-Ames Research Center. An airflow analysis workshop on a blunt fin rising from a plate.	21
3.1	Fundamental Parallel Algorithm.	23
3.2	The pseudo-code for the triangle exchange operation.	25
3.3	Parallel algorithm for horizontal division scheme.	26
3.4	An example of horizontal division for eight processors. The regions are separated by dotted lines.	27

3.5	Pseudo-code for FIND_DIVISION procedure. The input parameter R represents the number of horizontal regions and WLA represents the work load array.	29
3.6	Parallel algorithm for rectangular division scheme.	31
3.7	An example of rectangular division for eight processors organized into four clusters and two processors in each cluster. Dotted lines represent the boundaries of each region.	32
3.8	Parallel algorithm for the recursive subdivision scheme.	33
3.9	An example of recursive subdivision for eight processors. Dotted lines represent the boundaries of each region.	34
4.1	Input data BLUNT in hexahedral grid form.	39
4.2	Input data POST in hexahedral grid form.	40
4.3	Input data DELTA in hexahedral grid form.	41
4.4	Interconnection network of the IBM-SP2 Architecture.	43
4.5	Speedup graph of the horizontal, rectangular, and recursive division schemes where bounding box approximation is used and obtained by using the rendering times.	45
4.6	Speedup graph of the horizontal and recursive division schemes where rasterization algorithm is used and obtained by using the rendering times.	46
4.7	Speedup graph of the horizontal division scheme where bounding box approximation is used and obtained by using the overall execution times.	46
4.8	Efficiency graph of the horizontal, rectangular, and recursive division schemes where bounding box approximation is used and obtained by using the rendering times.	47
4.9	Efficiency graph of the horizontal and recursive division schemes where rasterization algorithm is used and obtained by using the rendering times.	47

4.10	Efficiency graph of the horizontal division scheme where bounding box approximation is used and obtained by using the overall execution times.	48
4.11	Load-balance graph of the horizontal, rectangular, and recursive division schemes where bounding box approximation is used and obtained by using the rendering times.	48
4.12	Load-balance graph of the horizontal and recursive division schemes where rasterization algorithm is used and obtained by using the rendering times.	49
4.13	Load-balance graph of the horizontal division scheme where bounding box approximation is used and obtained by using the overall execution times.	49

List of Tables

A.1	<i>Horizontal</i> division results using <i>bounding box</i> with <i>blunt</i> data. . .	55
A.2	<i>Horizontal</i> division results using <i>bounding box</i> with <i>post</i> data. . .	56
A.3	<i>Horizontal</i> division results using <i>bounding box</i> with <i>delta</i> data. . .	57
A.4	<i>Rectangular</i> division results using <i>bounding box</i> with <i>blunt</i> data.	58
A.5	<i>Rectangular</i> division results using <i>bounding box</i> with <i>post</i> data. . .	59
A.6	<i>Rectangular</i> division results using <i>bounding box</i> with <i>delta</i> data.	60
A.7	<i>Recursive</i> division results using <i>bounding box</i> with <i>blunt</i> data. . .	61
A.8	<i>Recursive</i> division results using <i>bounding box</i> with <i>post</i> data. . .	62
A.9	<i>Recursive</i> division results using <i>bounding box</i> with <i>delta</i> data. . .	63
A.10	<i>Horizontal</i> division results using <i>rasterization</i> with <i>blunt</i> data. . .	64
A.11	<i>Recursive</i> division results using <i>rasterization</i> with <i>blunt</i> data. . .	65

Chapter 1

Introduction

In this chapter, a brief introduction to the concepts of Scientific Visualization, Volume Rendering, Raycasting, Parallel Volume Rendering, and Distributed Memory Multicomputers will be given. In the first section, a general introduction will be presented. In the second section, a short overview of the related work will be introduced with some citations to some base papers for this thesis. In the third section we will give the motivation for this research, very briefly, along with the general organization of this document.

1.1 Overview of the Visualization Process

Scientific Visualization [20] is a developing field of research. As scientists try to develop more complex structures and models they feel the need to visualize their results on more understandable domains. At this point Scientific Visualization algorithms are utilized for detailed interpretation purposes of these complex data sets. *Volume Rendering* or *Volume Visualization* [7, 13], as a branch of Scientific Visualization, is a powerful approach to visualize *3 Dimensional* (3D) scientific data. It uses either standard computer graphics techniques like surface rendering [10, 18] or direct rendering techniques like ray shooting [17, 27] to visualize 3D scientific data. *Direct Volume Rendering* (DVR) [7, 13] is a technique that creates an image from the three-dimensional volumetric data set without generating an intermediate geometrical representation.

Three dimensional volumetric data sets are usually given as a set of data

points defined on 3D real Cartesian space where each data point represents a scalar, vectorial, etc. value about an entity like brain, plane, etc. These data points are called the sampling points as they give some sample results about the entity to be visualized, like heat, density, etc. The sample points of the volume data may be distributed in two major ways depending on the application. In the first way, the sample points are distributed over a *structured grid* with equal or variable spacing along each axis where an implicit interconnectivity between the grid points can be found. That is to say they can be given as a 3D computational virtual array where each neighbor in this array represents the same neighbor-ship relation in the 3D Cartesian space. This kind of distribution is common to medical imaging such as CT (Computer Tomography), MR (Magnetic Resonance), etc. In the second way, the samples are on an *unstructured grid* where connectivity is given explicitly. That is to say they are given as a 1D array in computational space and need an explicit information to be defined as a 3D array in real Cartesian space. The neighbor-ship relations among the data points can only be driven by using this explicit connectivity information. This type is common in Computational Fluid Dynamics (CFD) and Finite Volume Analysis (FVA) [20].

Usually, volume data is represented by 3D *voxels* which constitute the atomic pieces of the overall data structure in the context of domain mapping. These atomic pieces are usually assumed as cubes (e.g. in CT) or tetrahedrals (e.g. in FVA) where the corner points are the data (sample) points that represent the entity value at that 3D point. Note that some authors use cell versus voxel or sample versus voxel interchangeably but we will use them as they are defined in this chapter to avoid confusions and inconsistencies.

DVR is a desirable technique to visualize these kind of data sets because of the amount of information about the volume contents that can be presented in one image. DVR techniques have their drawbacks, however. DVR operates on volume data representation that requires a large amount of memory. DVR is also very slow since it requires massive computations for each image. So interactive speed rates are very hard to achieve.

An important approach to solve the speed and memory problems of volume rendering is to employ *parallel processing*. Furthermore, CFD and FVA simulations are usually run on parallel architectures because of simulation time and memory constraints. Hence, scientists want to visualize the simulation results on the same parallel architectures to avoid the migration of large amount of

volume data produced as a result of the simulations. Although parallel volume rendering of structured grids has been accomplished to a great degree, domain mapping problem for unstructured grids is a crucial problem to be solved. This thesis investigates the parallelization of rendering of volume data defined on unstructured grids.

Volume rendering algorithms can be classified in two main groups. First one is the *image-space* approach and second one is the *object-space* approach. *Raycasting* which is the basis of this research is an image-space approach and mainly uses ray shooting from each pixel of the image plane and sampling along its way [15, 17, 22, 25]. *Splatting* is an example of the object-space approach where each sub-element of data is projected onto the image plane with some order [23, 25, 28, 30]. In fact both of these approaches use the same underlying paradigms where only the information retrieval steps of the algorithms differ in terms of execution space (image versus object). Hence, from now on we will use DVR instead of Raycasting DVR without losing anything from accuracy and this thesis deals with the parallelization of the Raycasting type DVR algorithms [3].

The DVR algorithms use heavy computations and hence require image quality versus speed type optimizations to be advanced in sequential processing, while parallel algorithms maintain the image quality in gaining computational speed. The algorithms presented with the volume rendering paradigm are too compute intensive to be used for real-time or animation applications. Also the data sets used in this process are so huge that they can only be represented with many megabytes of storage space. Furthermore, scientists usually want to see the results of their simulations on the environment that the simulations are made (which are typically very powerful parallel architectures), because of the large data sizes that can be extremely painful to be ported to other environment. Especially time-varying datasets need to be visualized, in real-time, on a parallel work station where the simulations take place.

1.2 Related Work

Most of the previous work on sequential volume rendering dealt with structured grids where computations are carried on regularly distributed three dimensional computational structures [15, 17, 22, 25, 28, 30]. Some initial work on other

types of grids have also recently been introduced for sequential computing [8, 9, 14, 23, 26, 29, 31]. In addition to these works, there exists recent parallel research mainly concentrated on structured grids [4, 6, 11, 16, 19].

Very recently some parallel methods have been developed for unstructured grids [1, 2] which form a basis for this research. On the other hand these recent approaches are done on shared memory architectures which does not solve many of the problems of volume rendering on distributed memory architectures. Also although they maintain their scalability with the processor size they gradually deviate from the load balance parameters as the number of divisions increase.

The usage of polygon rendering algorithms in volumetric domain is introduced by [1, 2] and parallel methods for fast rendering algorithms are tried to be searched in polygonal spaces. The reason for this is to solve the problem of *fast sampling point location determination process* and *view sorting*, that is used to find the order of polygons to be traversed. These approaches use scattered task distribution and hence can only easily be used in shared memory architectures. Otherwise huge communication costs will be introduced in distributed memory multicomputers along with the great overhead of data duplications.

In addition to these works, some other parallel work has been carried out for only restricted topologies like curvilinear grids [5]. Therefore this type of approaches loose much from the data type restrictions and hence can not be used for general rendering purposes.

1.3 Motivation and Overview of the Thesis

This research presents tools for visualizing volumetric datasets on parallel architectures where huge memories, high computational capability, and power lies. This approach introduces algorithms for MIMD type distributed memory parallel architectures and gives some results on an *IBM-SP2* architecture. Although initially the algorithms developed are designed for volume rendering purposes, then, this parallel volume rendering approach is extended in such a way that any kind of data set, structured, unstructured, polygonal, or even hybrid (computational grids) can be visualized. It makes use of standard polygon rendering algorithms like *Scanline Z-Buffer Polygon Rendering* algorithm and tries to distribute the process of volume rendering evenly to all computational

nodes. The algorithms are mainly developed for real-time animation purposes where data flow over the network can be controlled. That is, the incremental movement of a view point will gradually change the local data within a processing element and hence prevent communicating nodes from congestion and from any other communication problems. The current results are promising and show us that some future work can hopefully give us a better solution. Three algorithms are developed that make use of image space subdivision that can lead to probable future algorithms for further improvements.

This thesis investigates and compares the parallelization of image-space based DVR algorithms on message-passing distributed-memory architectures, *multicomputers*. Multicomputers are very promising architectures for massive parallelism because of the nice scalability, fast access by processors to their local memories, and simultaneous local communication capability. Parallelization on such architectures necessitates the distribution of both computation and data to the processors with local memories in such a way that computational tasks can be run in parallel, balancing the computational loads of processors as much as possible. Communication between processors to exchange partial results must also be considered as a crucial part of the parallel algorithm and must match the constraints imposed by the interconnection and communication structure of the architecture by reducing the extra parallelization overhead of the rendering algorithm.

At the first step, a general overview of sequential volume rendering algorithms will be presented in the first section of the second chapter. Then our sequential approach will be given as a second section in the same chapter. After this, as a third chapter, we will analyze the parallel algorithm in general, deal with the three division heuristics, define the load balancing metrics, and finally refine our algorithms for implementation purposes, in this given order. As a fourth chapter the experimental results will be given. The first section of the fourth chapter will give the general overview of the data sets used and the architectures selected. Then as a second section the results will be presented in table and graph formats. Finally as our last chapter conclusions about the results and this research will be presented.

Chapter 2

Sequential Direct Volume Rendering

In this chapter, a general overview of the DVR approaches will be given along with our approach to DVR. In the first section, a general introduction to sequential DVR can be found. Then in the second section our approach (used in the implementations) to sequential DVR will be given.

2.1 Direct Volume Rendering in General

Data type of the input data sets is an important property of the visualization process. The presentation of the whole type set will be very helpful in deeper understanding of the overall process. There are two main types of data, *structured* and *unstructured* data types. These two main types are divided into several other subgroups.

These different data types [13], used in DVR, are given thoroughly in Fig. 2.1. Structured data types can be *Cartesian*, *Regular*, *Rectilinear*, or *Curvilinear*. All of these subgroups share the property of having implicit interconnectivity information. The first three types are self explanatory but the Curvilinear one shows a different property than the others. It may seem curved in 3D real Cartesian space but in fact it can easily be represented as a 3D computational grid (which is regular in this case) in computational space, this is obviously because of the implicit interconnectivity information property. Unstructured data types can be *Standard*, *Irregular*, *Hybrid*, etc. (e.g., B-Spline Curves and are beyond the scope of this thesis). These subgroups share the

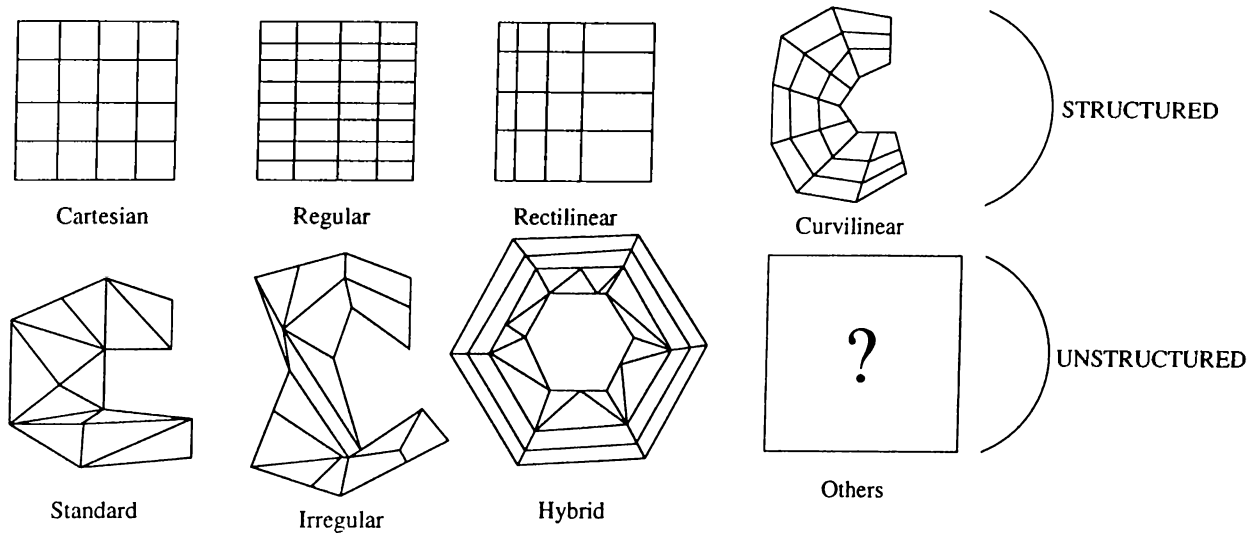


Figure 2.1. Data types used in volume rendering.

property of having explicit interconnectivity. The Hybrid type can be any mixture of the above. The Standard type uses tetrahedrals as its atomic particles. Irregular type uses polyhedrals as its atomic particles. This thesis deals with all of them except the very last type the curved types which is named as *Others*. In addition to these, we will try to render all other polygonal data sets with our basic algorithm for the sake of generality. This will bring us a great flexibility over the other approaches used in DVR media.

A basic Raycasting DVR algorithm shoots rays into the 3D data set, from each pixel of the image plane, and tries to extract information from the input data. While each ray intersects the data and the atomic particles (voxels) of the data it gathers information about the surrounding data grid points (sample points). So at the end, this gathered information somehow can be reflected as a pixel color, forming the overall image.

When the ray passes through the data, we take *samples* from the *sample points* (data points) of the data set which is called the *(re-)sampling process*. So by means of this we form a second 3D grid where the connectivity is now defined by the rays shot from the screen and the intersections with the voxels. Unfortunately these intersections are (again) called *sample points*. In fact this paradoxical definition hides the main fact, analogy, in the process. When scientists try to obtain information from their research subjects they use the same process of sampling, like Computer Tomography, they send e.g., ultra-violet rays into their data set (the brain) and take samples from the data set forming the input data for the DVR process.

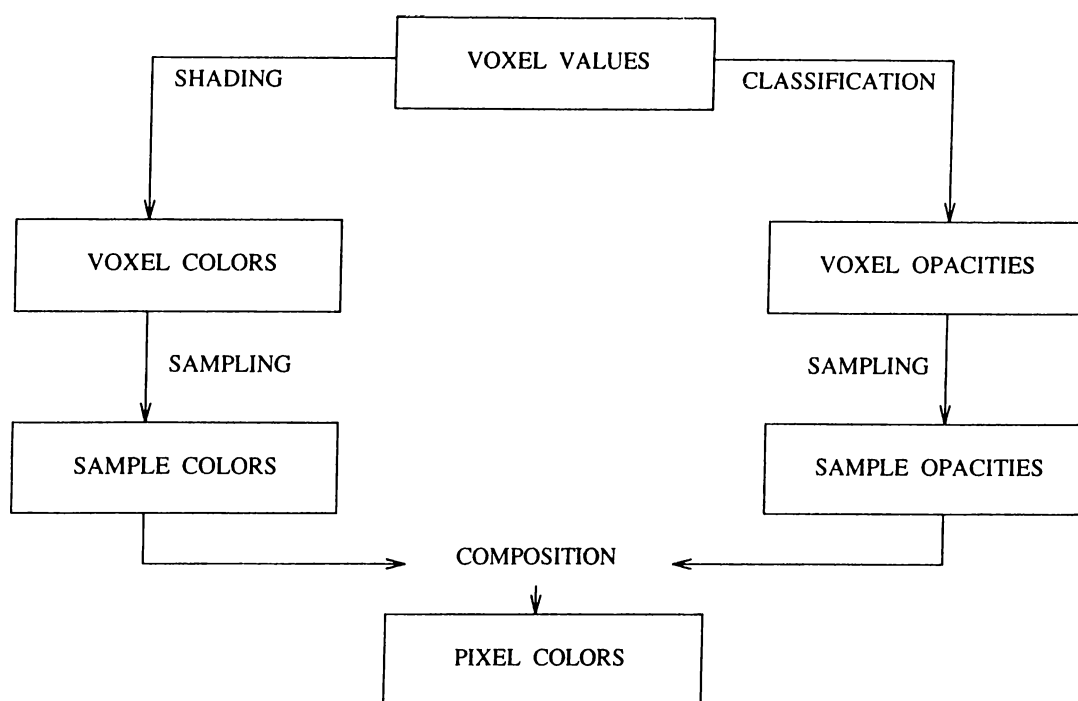
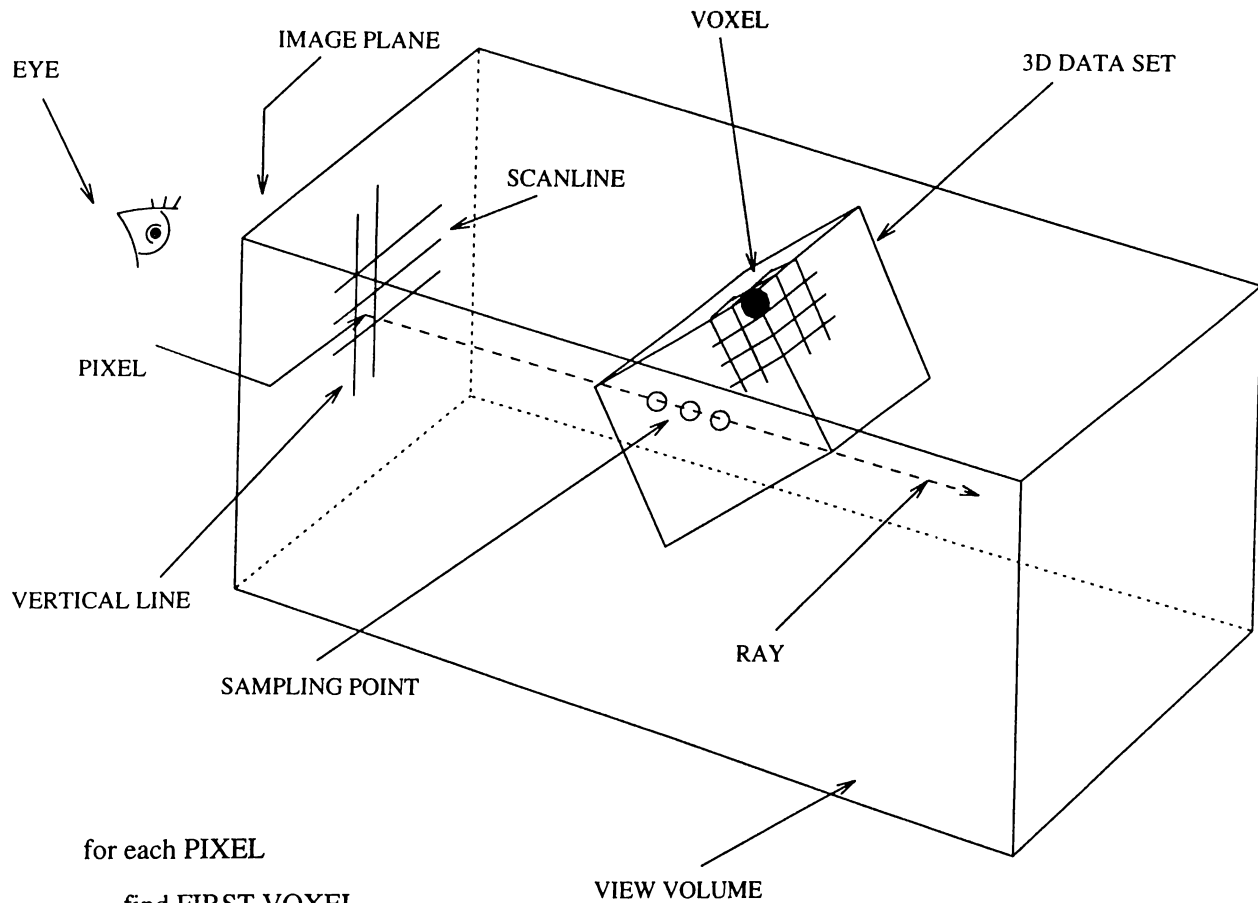


Figure 2.2. Flow chart of the sampling process.

In general, whatever the underlying data type is, the *brute force DVR sampling* algorithm works as follows as described in Fig. 2.2. From the voxel intersections we somehow get the *colors* and the *opacity* values of the intersection points (sample points) and then by means of a *composition* process we try to find the pixel color that is to be displayed. The opacity of a sample point defines the continuity of the ray, e.g., whether we hit to an opaque or transparent object or not. As the data is not defined with faces and polygons, but with only corner data points, the opacity combined with sample color plays a crucial role in the algorithm. So now we are ready to give the general Raycasting DVR algorithm for structured grids as it is shown in Fig. 2.3. As it can be seen from this figure the overall process is very similar to the Raytracing [27] algorithm which is well defined in the domain of computer graphics, except that the Raycasting approach uses grid points instead of the polygons.

So the question of how *composition* of simultaneous intersection (sampling) point values (color and opacities) remains still undefined. We take simultaneous samples from the data and try to explain the overall result as a simple pixel color so we should define a mathematical formula to sum up this array of intersection values (in fact two arrays one for opacity and one for color). This is explained by a set of formulas as in Eq. 2.1 [17].



```

for each PIXEL
  find FIRST VOXEL
  while in DATA do
    among the NEIGHBOR VOXELS
      find NEXT VOXEL to VISIT
    SAMPLE
    COMPOSE
  end while
  plot PIXEL
end for
    
```

Just Type Casting in Structured Data Sets

Equi-distant Samplings are Done in Structured Data Sets

Figure 2.3. Volume rendering overview.

$$\begin{aligned}
C_{out} &= (C_{in} \times O_{in}) + (C_s \times O_s \times (1 - O_{in})) \\
O_{out} &= O_{in} + (O_s \times (1 - O_{in})) \\
C_{out} &= \text{Ray Color after sampling in that voxel} \\
O_{out} &= \text{Ray Opacity after sampling in that voxel} \\
C_{in} &= \text{Ray Color before sampling in that voxel} \\
O_{in} &= \text{Ray Opacity before sampling in that voxel} \\
C_s &= \text{Sampling point Color} \\
O_s &= \text{Sampling point Opacity}
\end{aligned}$$

The set of formulas defining the Composition process. (2.1)

Initially the color value is set to background and opacity is set to the transparency value of that data set and this composition is done from the first to the last (or visa versa) intersection of the ray with data (by preserving the intersection order). Note that opacity changes between zero and one (floating point data) values where one means opaque intersection (e.g., skull) and zero means transparent intersection (e.g., glass). The opacity value of the data can be varied in order to find the area of interest in the data set. For example, a doctor can set the opacity of a skull to zero to make the skull transparent to the eye and see the brain inside it. This property in fact brings high-level of flexibility to the DVR algorithms in the visualization domain. Also color values should be selected carefully to keep a good diversity along the ray so that brain e.g., can be seen as gray and skull e.g., can be seen as white. The choice of these values is beyond the scope of this thesis that also needs further explanation to be understood thoroughly. But at least we can close one gap in this explanation, how do we obtain the color or the opacity of a data point from its data value (e.g., heat). This is another topic of research that is called the *mapping function* problem. There are various algorithms for this problem but (again) they are beyond the scope of this thesis. We will just state that a user defined function can be used, e.g., to obtain bright red from a high value of heat defined on a point or dark blue from a low value of heat defined on a grid point. After the whole data set is traversed the final ray color is equal to the pixel color itself. Then, this process is repeated for all of the pixels in the image.

Whatever the data type is, the underlying algorithm is similar. The only

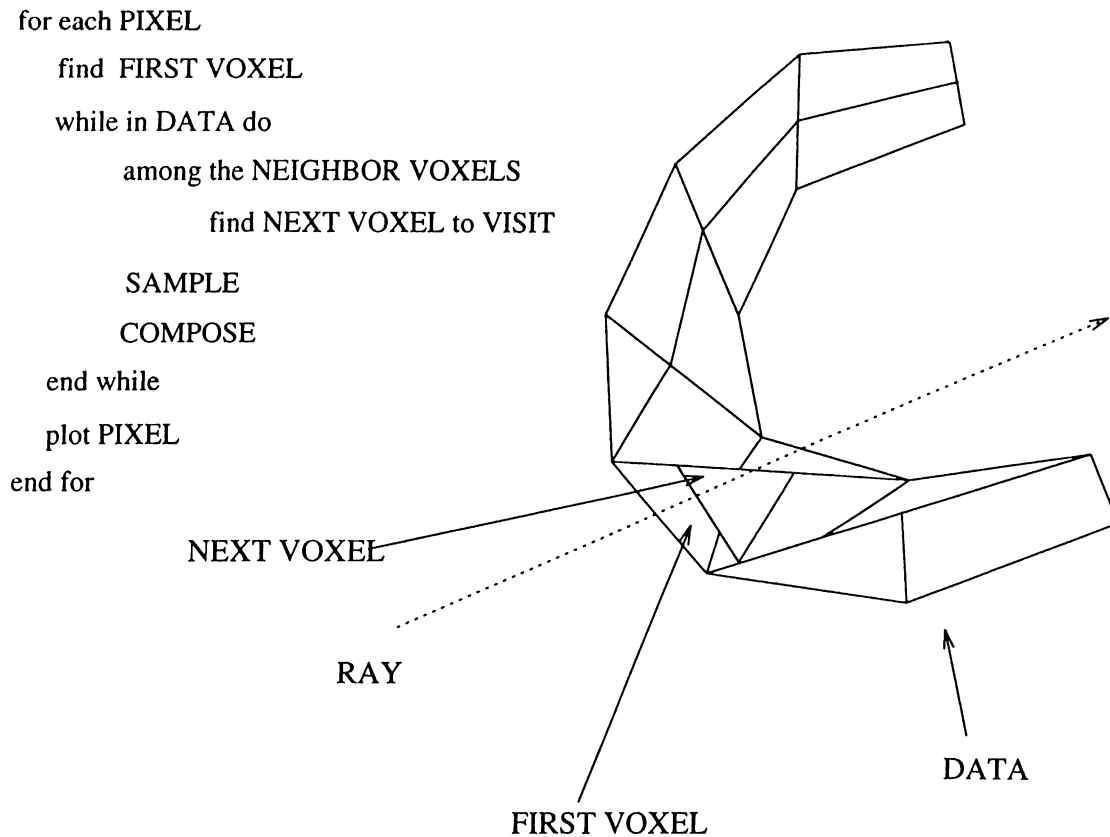


Figure 2.4. Overview of the volume rendering for unstructured data sets (2D illustration).

difference is in the traversal step. A similar algorithm for unstructured data types is given in Fig. 2.4.

Note that finding the next intersected voxel in unstructured data sets is more complex. The connectivity of the data is not implicit and an exhaustive search is need to be done. The next sampling point and the preceding sampling points should be found after various floating point operations. This problem is formally called the *point location problem*. Similarly the operation should be done with respect to depth which is a non-commutative but an associative process, this is called the *view sort problem*. These two problems are solved easily in structured domains as the real 3D space and the virtual computational space (where the floating operations done) overlaps (so the overall algorithm is easily stated for the structured domains as it is shown in Fig. 2.3). The only operation done in structured domains is type casting from floating 3D space to 3D array of voxels.

```
for each PIXEL
  find FIRST VOXEL
  while in DATA do
    among the NEIGHBOR VOXELS
      find NEXT VOXEL to VISIT
    SAMPLE
    COMPOSE
  end while
  plot PIXEL
end for
```

Figure 2.5. The general DVR algorithm.

However the overall algorithm is similar in both domains and can be simplified to one main algorithm. If we generalize the algorithm for all data types a general Raycasting DVR looks like Fig. 2.5.

Finally we should know how the samples are taken from the data set. It is defined just as an array of inverse distance interpolations. Some other approaches like tri-linear, constant, etc. can be used but all of these approaches use the same analogy used in inverse distance one. That is to say if a tetrahedral (simple four sided) is used as our input atomic data type (note that all type of data can be tetrahedralized) we just take four corner points of the tetrahedral and find the contributions of the corner values with respect to their distances (inversely) to the sampling point. So if we take a sample on one of the corners we will have zero effect from all the other corner points but the corner point that we are on itself. Or if we take samples in the mid-point of a voxel we will have equal contributions from the corner points. So the formulas for inverse distance interpolation between two corner points and a sampling point are given in Eq. 2.2, that is to say just for 1D. The same equations, can be found easily, in a similar way, for 2D and 3D cases as it is shown in Eq. 2.3 and in Eq. 2.4, respectively. These equations for 1D, 2D, and 3D interpolations are explained in Fig. 2.6 in more detail as A, B, and C, respectively.

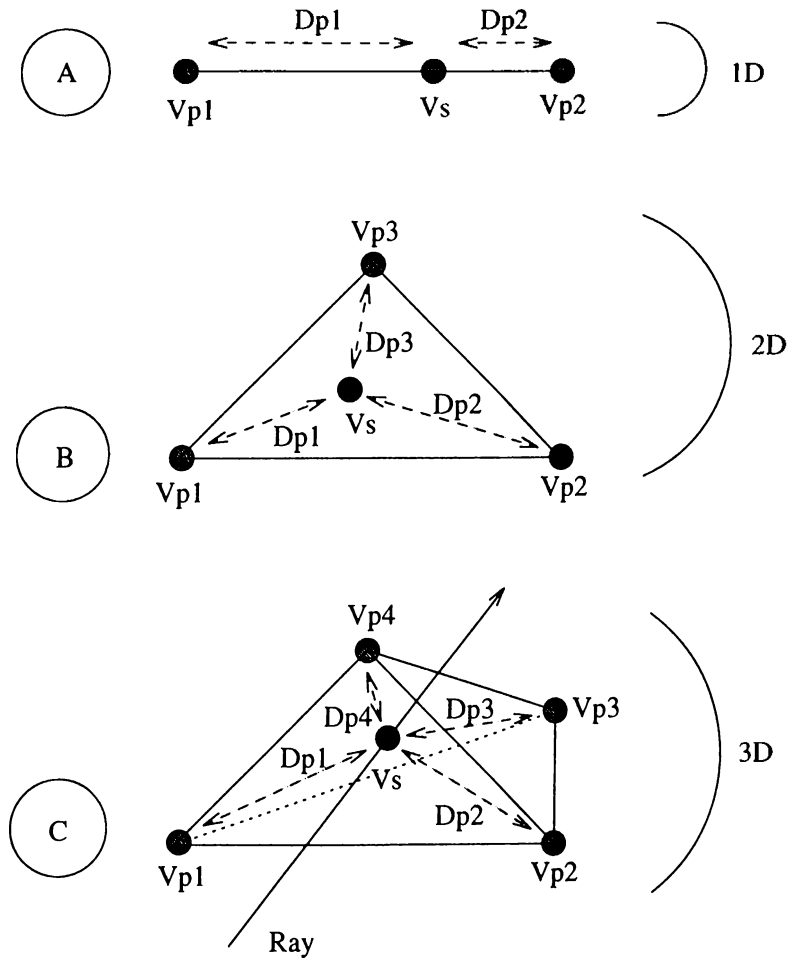


Figure 2.6. The interpolation process for different dimensions.

$$V_s = ((D_{p1} \times V_{p2}) + (D_{p2} \times V_{p1})) / (D_{p1} + D_{p2})$$

V_s = Sampled Value

V_{p1} = Value at corner 1

V_{p2} = Value at corner 2

D_{p1} = Distance of corner 1 to the sampling point

D_{p2} = Distance of corner 2 to the sampling point

The inverse distance Interpolation process for 1D. (2.2)

$$V_s = ((D_{p1} \times D_{p2} \times V_{p3}) + \\ (D_{p1} \times D_{p3} \times V_{p2}) + \\ (D_{p2} \times D_{p3} \times V_{p1}))/ \\ ((D_{p1} \times D_{p2}) + \\ (D_{p1} \times D_{p3}) + \\ (D_{p2} \times D_{p3}))$$

V_{p1} = Value at corner 1

V_{p2} = Value at corner 2

V_{p3} = Value at corner 3

D_{p1} = Distance of corner 1 to the sampling point

D_{p2} = Distance of corner 2 to the sampling point

D_{p3} = Distance of corner 3 to the sampling point

The inverse distance Interpolation process for 2D. (2.3)

$$V_s = ((D_{p1} \times D_{p2} \times D_{p3} \times V_{p4}) + \\ (D_{p1} \times D_{p2} \times D_{p4} \times V_{p3}) + \\ (D_{p1} \times D_{p3} \times D_{p4} \times V_{p2}) + \\ (D_{p2} \times D_{p3} \times D_{p4} \times V_{p1}))/ \\ ((D_{p1} \times D_{p2} \times D_{p3}) + \\ (D_{p1} \times D_{p2} \times D_{p4}) + \\ (D_{p1} \times D_{p3} \times D_{p4}) + \\ (D_{p2} \times D_{p3} \times D_{p4}))$$

V_{p1} = Value at corner 1

V_{p2} = Value at corner 2

V_{p3} = Value at corner 3

V_{p4} = Value at corner 4

D_{p1} = Distance of corner 1 to the sampling point

D_{p2} = Distance of corner 2 to the sampling point

D_{p3} = Distance of corner 3 to the sampling point

D_{p4} = Distance of corner 4 to the sampling point

The inverse distance Interpolation process for 3D. (2.4)

The corner points of a voxel in general represents a density, heat, or any other scalar value. Vectorial or multi-dimensional corner values are beyond the scope of this thesis where thoroughly different approaches are used.

In general samples are taken in the middle of the voxel for unstructured grids. But for structured ones equi-distant samplings can be taken. In addition to these double sampling, multiple ray shooting and other heuristics can be used to refine the images obtained. But all of these approaches use the same underlying paradigms. So if we use mid-point interpolation scheme we find the entrance and exit points of the ray to the voxel, find its mid-point, and do the sampling at that point. If equi-distant one is used samples are taken at regular distances whatever the intersections with voxels of the data are. The crucial point here is to identify the voxel that we are in, in order to find the right corner points for interpolation. If e.g. densities are interpolated then we should convert these sampling values to color and opacities (on the fly) by using some mapping function. But if color and opacity values are interpolated then a preprocessing is needed to find the color and the opacity of each grid point. The explanation for this process can be derived from Fig. 2.2. There are various different approaches used at this point by the authors in this field but they only effect the image quality with out effecting the underlying rendering algorithm. The processing time versus memory trade off always makes one method preferable to other in sequential type of optimizations. So now we are ready to introduce our approach with this amount of initial knowledge in the proceeding sections.

2.2 The Proposed Sequential Approach

The high quality of the images produced by the Raycasting approach makes it a desirable choice for DVR. Although many projection algorithms run faster than the Raycasting algorithm they show us various difficulties especially in the parallelization step because of the nature of the algorithms. For example in the projection type algorithms (object-space approaches) the projections should be done in either back to front or front to back order to preserve the

data integrity in the final image. Also object space divisions of these algorithms might need intensive pixel merging operations as a post processing. But nevertheless some ideas can be borrowed and inserted into Raycasting from the projection methods.

In this approach a ray is shot from each pixel and traversed throughout the whole volume to determine the list of voxel intersections. Each voxel intersection means an entry/exit point of the ray with the voxel. For each voxel intersection, a sampling is computed at the midpoint of the ray between its entry and exit points by interpolating the scalar values at the grid points of the intersected voxel. The voxel intersections should be determined in a predetermined order (front-to-back in our case) for the composition of the sampled color and the opacity values. Ray shooting, sampling, and finally composition steps require the detection of the position of the sampling point in the whole data and finding the next (therefore previous) voxels (sub-volumes) to be intersected with the ray for composition, in the well-known Raycasting algorithm [17]. These two operations, in addition to the heavy computations introduced by the samplings and compositions, bring tremendous amount of computation to the process of Raycasting. Therefore finding the consecutive intersections and the locations of the sampling points should be done efficiently, which we refer as efficient point location operation and view sorting.

In this work, we adopt the basic ideas in the standard polygon rendering algorithms to resolve the point location and the view sorting problems. This idea which is introduced previously by [1] if cleverly used can be very useful in the parallelization of the overall process. For example, well-known parallel polygon rendering [24] approaches can be applied to the volume rendering domain. Therefore, the overall algorithm needs polygons to be rendered. This is easily done by just converting the data set into a polygonal form. That is to say connecting data points in a way that we will have a set of polygons in the final data set. This operation once completed can be saved and used forever. For example any kind of tetrahedral or hexahedral sub-volume can easily be converted to a set of triangles where any three points define a single triangle which is obviously planar, more advanced algorithms also exist for *triangulating* a given volumetric data set [23]. Hence, from now on we will mainly assume triangles as our inputs for the sake of simplicity. Moreover, this give us the power of dealing with any kind of data sets whatever the underlying type is.

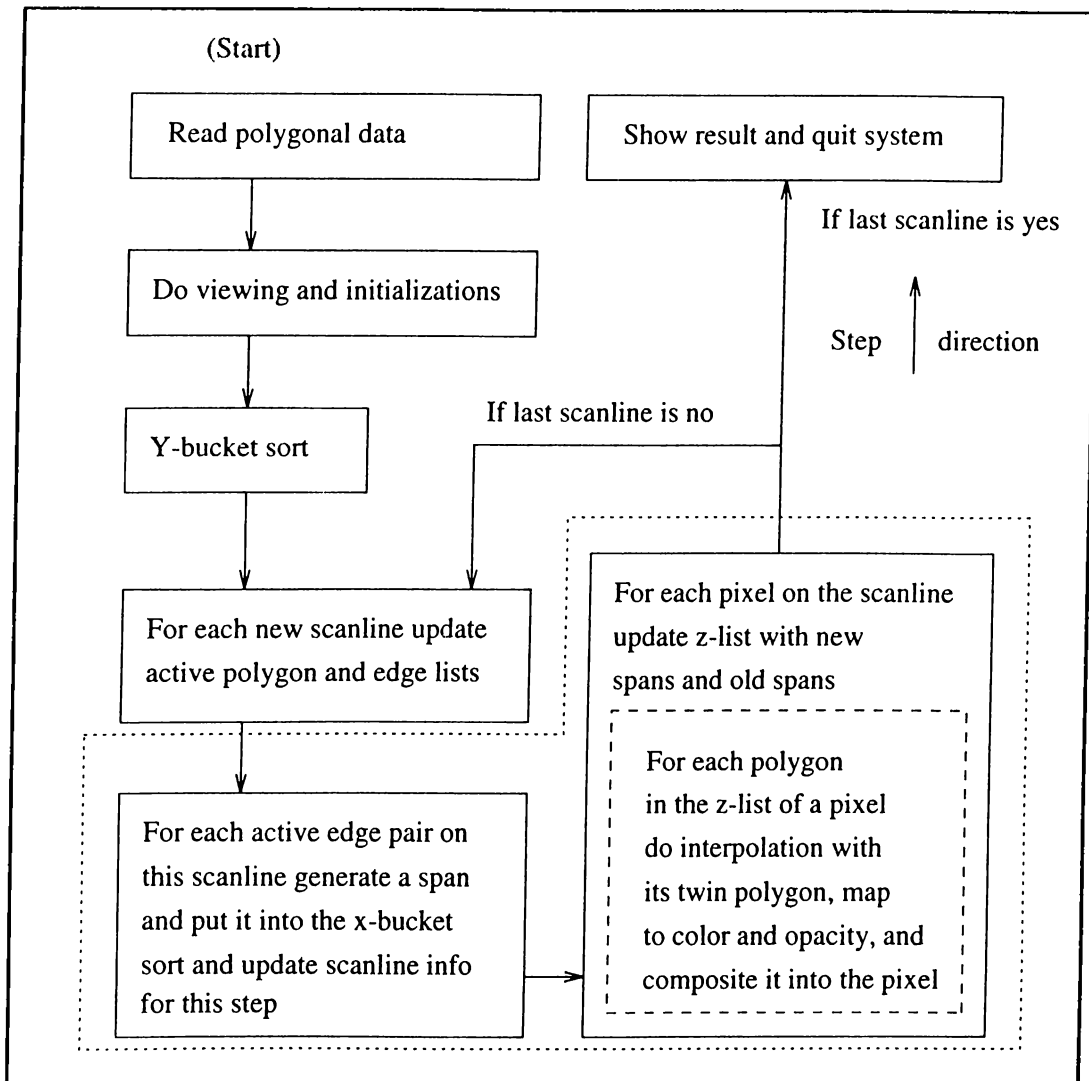


Figure 2.7. Flow chart of our sequential algorithm.

So we can define our algorithm, as a sequential process, *Scanline-Based V-Buffer* algorithm. The proposed algorithm is similar to the standard Scanline Z-Buffer algorithm in the rasterization phase. It differs in the rendering phase as follows. In our algorithm, each pixel keeps a linked list of polygons for compositing and finding a final pixel color value. The flowchart for the proposed algorithm is given in Fig. 2.7.

As seen in Fig. 2.8, through out the algorithm we move from scanline to scanline and from pixel to pixel incrementally by updating a single linked list of polygons, and therefore saving from storage space. When a new pixel is entered the current linked list is updated in an incremental fashion and some deletion and insertions to the linked list of active polygons occur. As we know

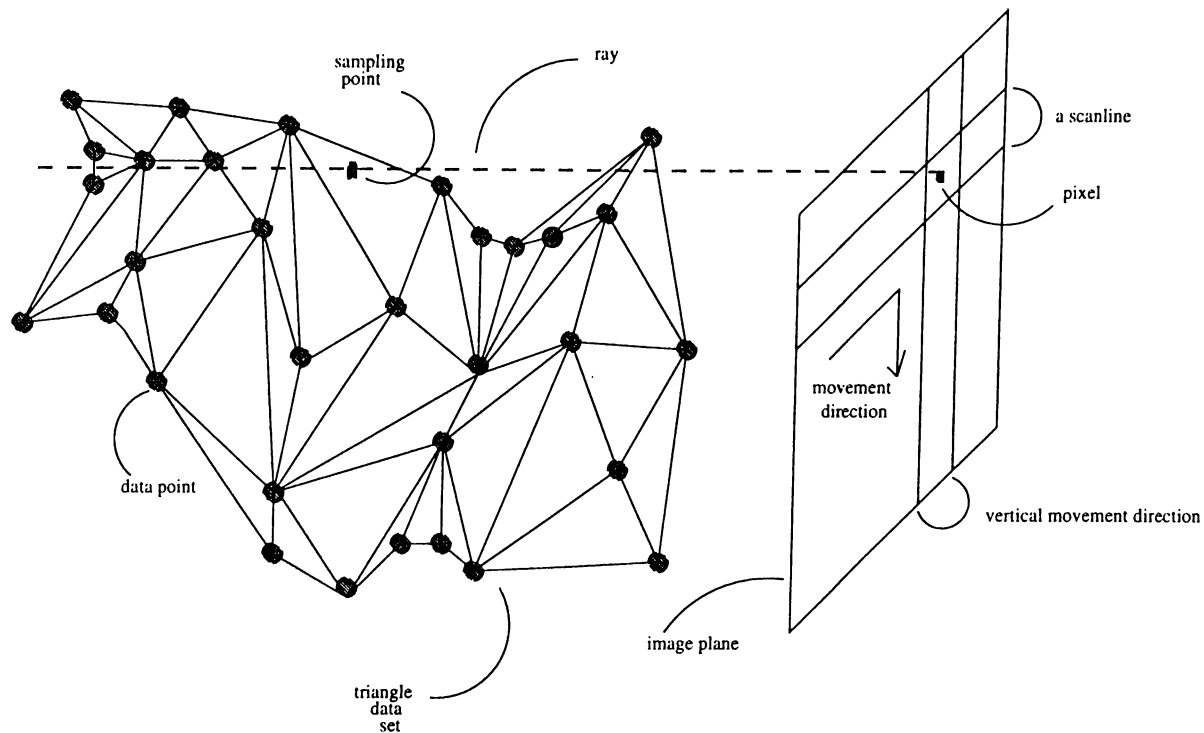


Figure 2.8. Overview of our sequential process.

the location of each intersection of the active polygon with the ray and as the list is built in an incremental fashion we can say that we just have an array of sorted intersections with a three dimensional line and a set of planes. So for a single pixel, after the intersections are found we can go through the list and take samples between each pair of triangles and composite it into the pixel color. At this point, color and opacity calculations are just done as defined in [8]. Here the corner points of the triangles give us the necessary information, e.g. speed, and these values are used in interpolations.

Here, and hereafter, we assume that the data sets used are described as a set of distinct triangles which are converted from a set of tetrahedrals or any other data set type. So we can see that many of our problems like sorting and point location are solved by this approach. Here, the two consecutive triangles in the list define a set of points that can range from 3 to 6. If the two triangles share an edge this number is obviously 4, else if two (apart) external faces of the whole data set are used this is 6 and the sampling is prohibited. If this number is 5, only a point is shared by the two triangles. So as a result, except external faces, two consecutive triangles can be used for the determination of the sampling value. This is just done by taking the end points of the triangle, finding the midpoint of the two intersections and taking the inverse distance

interpolation of the end point data values (at this point tri-linear interpolation can also be used). Finally via any mapping function the sampling point color and opacity can be derived from this interpolated data value [15]. But note that all intersection tests and interpolations use the coherency introduced by the Scanline Z-Buffer algorithm.

In fact many other polygon rendering algorithms can be used for this purpose where this one seems to be the best because of its storage and coherence advantages. This algorithm although needs repetitive renderings for different viewing parameters, can be very useful, if we consider the fact that after each rendering operation many transfer (mapping) functions can be tried on the same view of the data set, or different properties can be viewed for the same viewing values of the data. Also some time-varying data sets can be animated using the same view of the data but by just interpolating different data values. In addition to these all of the remaining methods (as it is the case in projection methods) have the same problem of view parameter dependency. Apart from these view point changes, if done incrementally, as it is the case in many animations, will be less effective on the position of the data with respect to eye.

So we are ready to see two outputs of two sample runs of two volume renderers in the following two pages to increase our knowledge about the outcome of a volume rendering program. The first one is a CT (Computer Tomography) output shown in Fig. 2.9. The second one is a CFD (Computational Fluid Dynamics) output displayed in Fig. 2.10. So now, we are ready to go on with the parallel algorithms keeping in mind these two images for the parallel versions. It would be useful to replace generic volumes in the following chapters with these images to increase understandability.

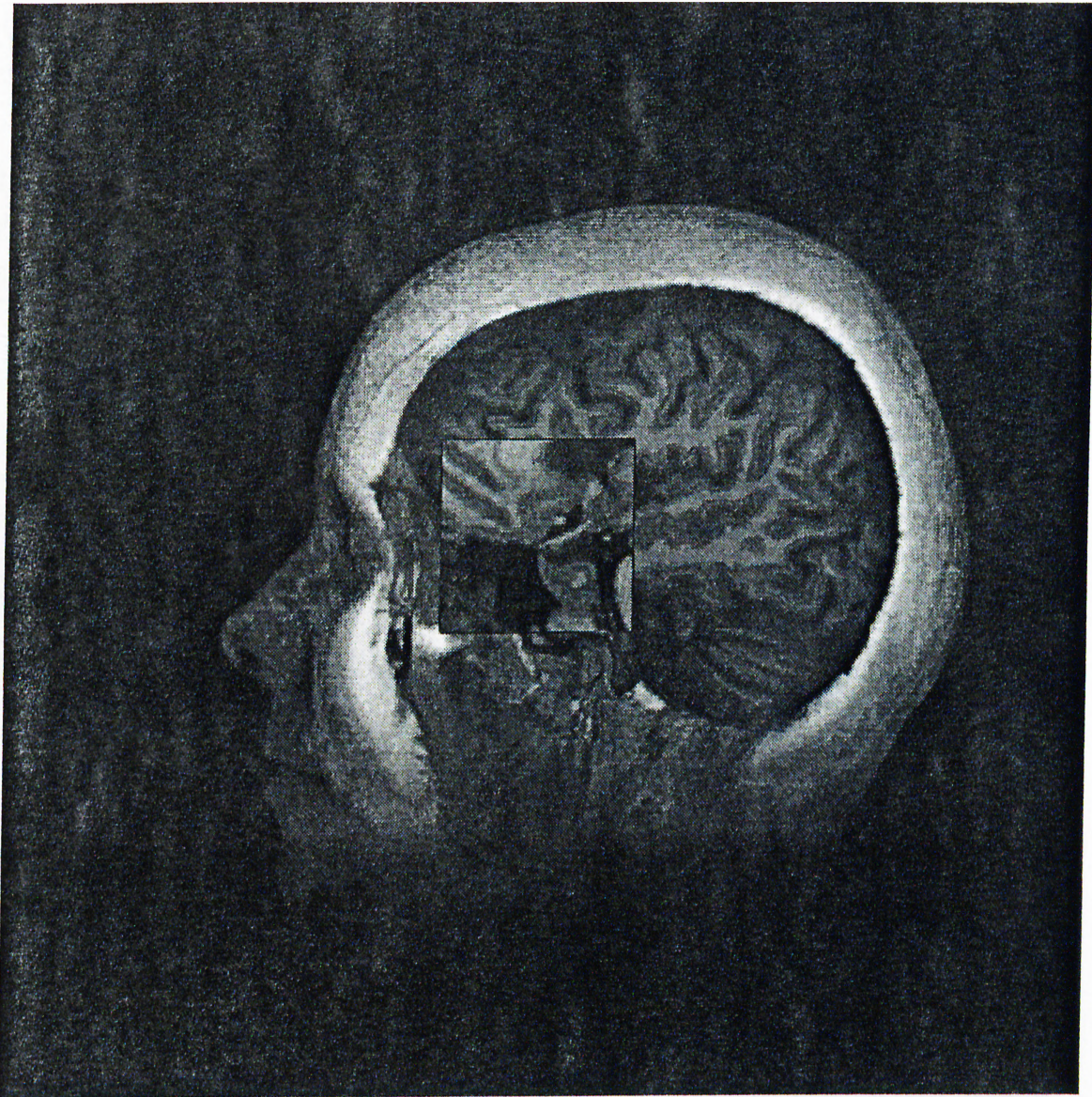


Figure 2.9. Data from University of North Carolina Chapel Hill. A Computer Tomography image of a cadaver head.

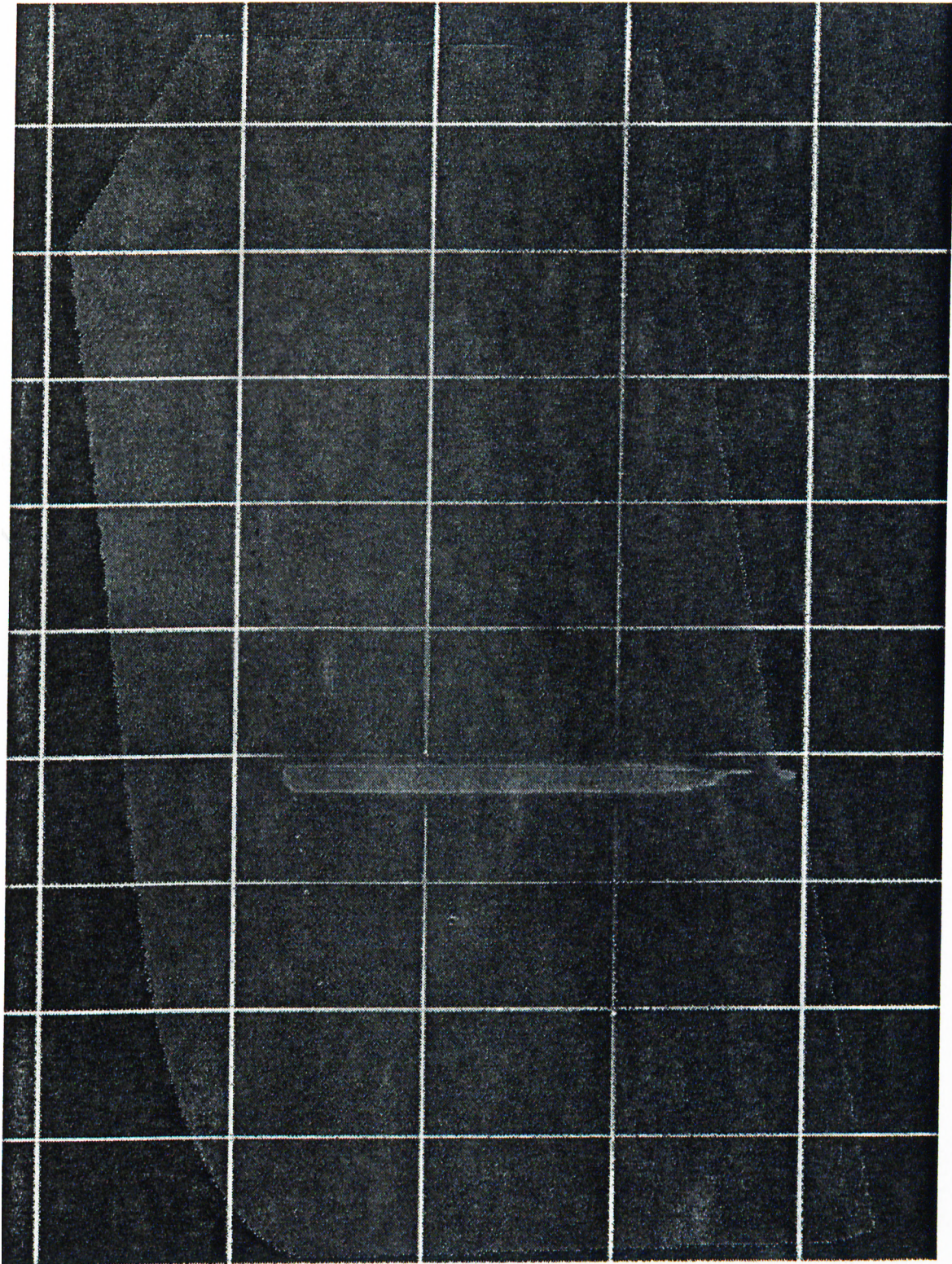


Figure 2.10. Data from NASA-Ames Research Center. An airflow analysis workshop on a blunt fin rising from a plate.

Chapter 3

Parallel Direct Volume Rendering

In this chapter, our parallel approaches to DVR will be given. In the first section our parallel algorithm, in general, will be introduced. In the second section three different subdivision heuristics will be given. Then, load balancing metrics will be analyzed in detail in the third section. Finally, in the last section, some refinements about the algorithms developed will be introduced.

3.1 Our Parallel Implementations in General

The parallel implementation of volume rendering on MIMD distributed memory parallel machines requires the partitioning and mapping of data and computations to the processors of the parallel architecture. The partitioning and mapping should be done in a way to achieve maximum processor utilization. Computational *load balancing* is a crucial issue in parallel processing to achieve maximum processor utilization. The computational work load should be distributed as evenly as possible among the processors. Three load balancing heuristics are described and discussed to achieve better distribution of load among the processors in this chapter.

The parallel algorithm presented in this section is an image-space parallel algorithm. In image-space parallelism, the image plane is partitioned among the processors. After this partitioning step, each processor runs a sequential volume rendering algorithm to generate the image for its local image plane sections. Each processor needs the volume data which is covered by the view

-
1. Receive a portion of global volume data and perform viewing transformations.
 2. Partition the image-space and find the assigned image partition.
 3. Exchange some of the local volume data according to the image-space partition among the processors.
 4. Perform volume rendering on the local image partition using the new local volume data.
-

Figure 3.1. Fundamental Parallel Algorithm.

volume of the local image plane sections. Therefore, the volume data is partitioned and distributed to the processors according to partitioning of the image plane as well. The main steps of the parallel algorithm is given in Fig. 3.1.

At the first step of the algorithm, the global volume data is partitioned and distributed among the processors. Each processor receives an equal amount of volume data and performs viewing transformations. Note that the volume data for unstructured volumes is composed of tetrahedral voxels and each tetrahedral voxel data is made up of a four triangles. Also note that all the data types presented in the preceding sections can be tetrahedralized and every tetrahedral data set can be triangularized. So we can have a set of triangles at the end of these conversions which can be saved and used forever instead of the original data, after it has been created. There are various algorithms for this process which can be used as an *early-pre-processing* for the main algorithm [23]. Hence, we can generalize our algorithm to all data sets that are given as a set of polygons. Besides, no connectivity information is needed as the underlying sequential algorithm is the Scanline Z-Buffer algorithm which makes the parallelization operation much easier than expected as it just needs a set of unconnected polygons to be rendered. These assumptions and generalizations makes the algorithm a much more flexible one among its rivals. Therefore, each processor receives a distinct set of T/P triangles, where T is the number of triangles and P is the number of processors.

At step 2, the image space is partitioned into P rectangular regions and each region is assigned to a processor. The partitioning of the image space should be done adequately in order to achieve an even distribution of load among the processors. So we can say that our parallelization approach is a

static one which maps every generated subtask to a single processor and each processor takes only a single subtask, all of which forms the overall job to be done. Hence, as it can also be seen from the fundamental algorithm only a preprocessing step for job distribution is needed (introduced as an overhead to the rendering process with the data exchange operation) but not a post-one as the partition does not divide the 3rd dimension of the volumetric data set. Three strategies to achieve this goal is presented and discussed in the following sections. The triangle set received at the first step is utilized to perform an adaptive division of image plane.

After the partitioning and assignment of the image plane, each processor needs triangles which fall into the view volume of the local image plane partition. The local triangle data set may contain triangles that belong to image plane partitions assigned to other processors. Similarly, some of the triangles that are covered by the view volume of local image plane section may reside in the local memories of other processors. Therefore, at step 3, some of the triangles – hence, volume data – should be exchanged among the processors. Each processor finds the image plane region a triangle belongs to by performing projection and clipping operations and sends the triangle to the corresponding processor and receives triangles that fall into its local image plane region. The pseudo-code for this exchange operation is given in Fig. 3.2. Note that, the triangles at the boundaries of image plane partitions will be shared by two or more processors. Hence, such triangles may be transmitted more than once and will be duplicated after each exchange operation.

At step 4, each processor runs the sequential volume rendering algorithm for its local image plane section using new local volume data (triangle data). Also no post-processing is needed in this type of parallelizations as the final sub-images can directly be concatenated to obtain the overall final image.

3.2 Subdivision Heuristics

In this section, three different subdivision heuristics will be introduced. In the first subsection, the horizontal, in the second subsection the rectangular, and in the third subsection the recursive heuristics will be given. The exact definition of the division metrics will be delayed until the next section for the sake of abstraction and then after the development of these heuristics a

```
EXCHANGE_DATA
  for each processor p do
    for each local triangle t do
      Project and clip triangle to the image plane assigned to p
      if triangle is in the image plane of p then
        Put triangle into send array
      endfor
    Transmit send array to processor p
    Receive triangle information from some other processor
    Store the received triangle information to the local array
  endfor
END_EXCHANGE_DATA
```

Figure 3.2. The pseudo-code for the triangle exchange operation.

compact formalization of the metrics will be given as a separate main section. Similar methods for parallel Raytracing is used in [12].

3.2.1 Horizontal Heuristic

We have stated that there are three different heuristics for the division scheme. All of these three heuristics divide the image plane into smaller rectangular regions first of which is named as the *horizontal* subdivision heuristic. In this scheme, the image plane is divided into P horizontal bands which are composed of consecutive scanlines on the image plane. In this way, intra-scanline coherence is preserved to some extent while disturbing the inter-scanline coherency. The amount of work load in each region is given by the sum of the work load at each scanline in that region. Hence, the atomic process for this type of subdivision scheme is a single scanline which can not be divided into smaller tasks. So the division alternatives for a given image plane is the number of scanlines for that plane which obviously restricts the division flexibility. However, this scheme suffers from unscalability since the number of atomic tasks is limited by the number of scanlines in the image plane. The algorithm for horizontal division scheme is given in Fig. 3.3. An example of horizontal division scheme for eight processors is given in Fig. 3.4.

In steps 1-3 of the algorithm, each processor calculates the local work load

-
1. Take a local triangle.
 2. Project the bounding box of the triangle onto the image plane and find the y-span of the projected bounding box.
 3. Update the work load at each scanline covered by the y-span of the projected bounding box by updating the corresponding entries of *SWLA*.
 4. Repeat steps 1-3 for all local triangles.
 5. Perform a global sum operation on *SWLA* to find the global work load at each scanline.
 6. Perform a prefix sum on *SWLA*. After the prefix sum operation $SWLA[s]$ gives the work load of region between scanlines 1 and s , including s .
 7. Last entry of the *SWLA* gives the total work load on the image plane. Divide this value to the number of processors to find the average amount of work for each processor. Set $AvrgLd = SWLA[N]/P$, where N is the number of scanlines and P is the number of processors.
 8. Call `FIND_DIVISION($P, SWLA$)` procedure (given in Fig. 3.5).
-

Figure 3.3. Parallel algorithm for horizontal division scheme.

at each scanline using local triangle information, and stores the values in an array, called *Scanline Work Load Array (SWLA)*, of size equal to the number of scanlines on the image plane. Each entry of this array corresponds to a scanline on the image plane. The work load metrics used for each scanline are the total number of triangles intersected by the scanline, the total length of *x-spans* generated for each triangle on this scanline, and the total number of *x-spans* for that scanline (each constitutes an atomic job that has relative work load proportions with respect to each other in the sequential rendering process, e.g., initialization of a triangle might be two times expensive than a single span). Each of these metrics and the overall load balancing metric logic will be discussed in the following sections in more detail. The y-span of each triangle gives the number of scanlines covered by the triangle. Hence, the number of triangles at each scanline can easily be calculated using y-span of each triangle. Similarly the number of *x-spans* for a scanline can be found easily. However, the length of the *x-span* at each individual scanline requires rasterizing edges of the triangle. This computational overhead can be decreased

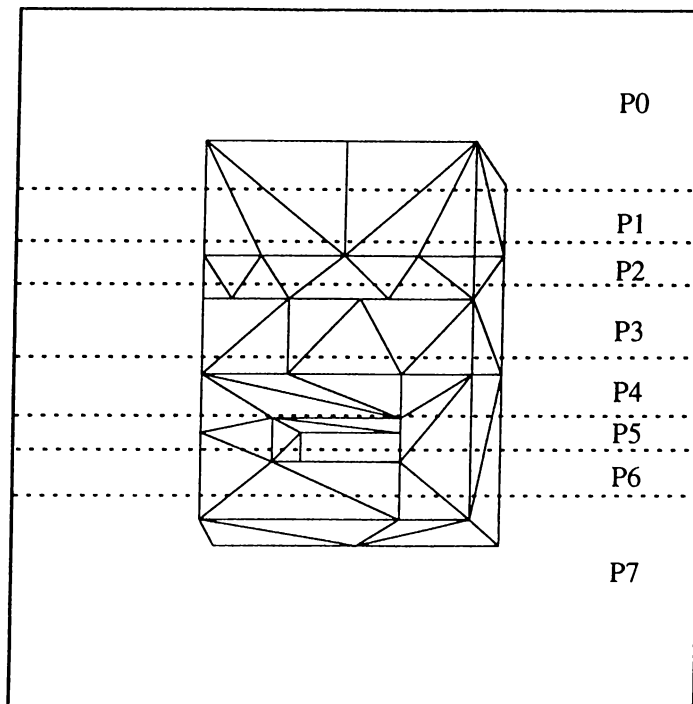


Figure 3.4. An example of horizontal division for eight processors. The regions are separated by dotted lines.

by using the bounding box of the triangle instead of triangle itself. The x -span length at each scanline is approximated by the x -span of the bounding box. Since the bounding box of a triangle is a rectangle, the x -span lengths will be the same at each scanline. Therefore, the computational overhead of rasterizing edges of the triangle is avoided. The area of the bounding box is a close approximation to the actual projection area of small triangles. Hence, the x -span of the bounding box is a good approximation to the actual x -span lengths of such triangles. However, for large triangles, the actual areas of two triangles that have the same bounding box area and the same y -span length may be substantially different. Hence, for such triangles, bounding box approximation may introduce large errors. These errors can be decreased by calculating the actual area of triangles whose area is larger than a predefined threshold value. Then, the actual area calculated is used to create a new bounding box of area equal to actual area of the triangle and of the same y -span. The x -span of new bounding box is calculated as $x\text{-span} = \text{Area} / y\text{-span}$, and this new x -span length is used to update work load at each scanline.

After step 4, entries of $SWLA$ contain the local work load at each scanline. Since the global work load is required to obtain the partitioning, a global sum operation is performed at step 5. This global sum operation can be done in

$\log_2(P)$ steps using the communication structure of the IBM-SP2. At the end of this step, each processor has global work load distribution in y-dimension of the image plane.

At step 6, a prefix sum is performed on the *SWLA* so that $SWLA[s] - SWLA[l]$ gives the work load of horizontal band bounded by scanlines $l+1$ and s . Note that, $SWLA[N]$ gives the total work load on the image plane, where N is the total number of scanlines. Average work load (*AvgLd*) is calculated by dividing this value to the number of processors at step 7.

At step 8, the partitioning of the image plane is performed in procedure *FIND_DIVISION*. The pseudo-code algorithm for this procedure is given in Fig. 3.5. This procedure finds the partitioning of image plane using the work load values stored in *WLA*. Two arrays of size R is allocated, where R is the number of regions to be generated. Note that, the value of R is equal to the number of processors in horizontal division scheme. The array called *region_start* stores the starting scanline (lower boundary) of the image plane region and the array called *region_end* stores the ending scanline (upper boundary) of the same region. The algorithm starts from first scanline setting $s = 1$ and $e = 1$ forming a region of one scanline, where s represents the lower boundary of a region, and e represents the upper boundary of a region. Then, this region is expanded by incrementing e until $WLA[e]$ is larger than average load. The upper boundary of the region is determined by comparing the work load values at entries $e - 1$ and e to the average load and choosing the one closest to the average load as the upper boundary of the horizontal band. After this region is formed, algorithm repeats the same process for a new region until image plane is divided into R horizontal sections. Note that, the lower boundary of the new region is set to $s = e + 1$.

So we have found the necessary regions for sequential rendering where after the data exchange operation each processor runs a sequential rendering on its local data set. This division scheme is refined gradually for the sake of efficiency and good load balance purposes where these refinements along with the load balancing metrics will be more precisely defined in the following sections.

```

FIND_DIVISION(R,WLA)
  s = 1
  e = 1
  p = 1
  k = AvgLd
  for p = 1 to R do
    while AvgLd < WLA[e] do
      e = e + 1
    endwhile
    if ((AvgLd - WLA[e - 1]) < (WLA[e] - AvgLd)) then
      region_start[p] = s
      region_end[p] = e - 1
      s = e
    else
      region_start[p] = s
      region_end[p] = e
      e = e + 1
      s = e
    endif
    AvgLd = k + WLA[s - 1]
  endfor
END_FIND_DIVISION

```

Figure 3.5. Pseudo-code for FIND_DIVISION procedure. The input parameter R represents the number of horizontal regions and WLA represents the work load array.

3.2.2 Rectangular Heuristic

In the horizontal division scheme, the image plane is divided into horizontal bands of consecutive scanlines by using the work load distribution in y-dimension of the image plane. Therefore, horizontal division scheme partitions the image plane in one dimension only, namely y-dimension. Due to this restriction, the scalability of horizontal division scheme is limited by the number of scanlines. In addition, if there are large differences in the work loads of scanlines, the load imbalance between regions may still be large. These disadvantages can be avoided to some extent by partitioning the image plane in both dimensions into rectangular regions.

In rectangular division scheme, processors are organized into a two dimensional $M \times K$ mesh, thus forming M clusters of K processors in each cluster. Then, the image plane is divided into M horizontal bands. After partitioning image plane into M regions, each processors calculates the work load distribution in x-dimension of each region. Then, each region is divided into K vertical bands of consecutive vertical scanlines in x-dimension. The algorithm for rectangular division scheme is given in Fig. 3.6. Each processor keeps a *scanline work load array* (*SWLA*) similar to the one in horizontal division scheme to find M horizontal bands. In addition, each processor allocates M *x-dimension work load arrays* (*XWLA*) to find K vertical divisions in each horizontal section of the image plane. Each $XWLA[v]$, for $v = 1, \dots, M$, is of size equal to the number of vertical scanlines in the x-dimension of the image plane. Each entry $XWLA[v][j]$, for $v = 1, \dots, M$ and $j = 1, \dots, x - dimension$, corresponds to a vertical scanline j in region belonging to cluster v . An example of rectangular division scheme for eight processors is illustrated in Fig. 3.7.

In this scheme, after M horizontal partitions are found, each processor treats each horizontal region as a new image plane rotated 90 degrees. Hence, the number of scanlines in each new image plane is equal to the number of vertical scanlines in x-dimension of the global image plane. Each processor projects the bounding boxes of local triangles to find the work load distribution in each horizontal band. If a bounding boxes spans two or more horizontal regions it is divided into segments and work load distribution of each region is updated according to the corresponding segment. After this step, a global sum operation is performed to obtain the global work load distribution in x-dimension in each region. Afterwards, each processor finds vertical partitioning in the horizontal region of its cluster using `FIND_DIVISION` procedure with input parameters $R = K$ and $WLA = XWLA[my_cluster]$.

Each processor needs the vertical division information in other clusters so that it can find the rectangular region that the projection of a local triangle overlaps. Therefore, at the last step, a global concatenate operation, on the vertical divisions in each cluster, is performed so that each processor has the information about vertical divisions in other clusters. This global operation can be done in $\log_2(M)$ steps.

-
1. Partition image plane into M regions using horizontal division scheme.
 2. Calculate the work load distribution at each horizontal region using local triangle information.
 - (a) Project bounding boxes of each triangle.
 - (b) Partition the bounding box of the triangle if it covers two or more horizontal regions.
 - (c) Update the corresponding entries of $XWLA$ for each region covered by the bounding box. Use the partitions of the bounding box data to update entries as in the previous scheme.
 3. Perform global sum operation on the $XWLA$ for each region.
 4. Perform prefix sum operation on the $XWLA[my_cluster]$, where $my_cluster$ is the cluster that processor belongs.
 5. Last entry of the $XWLA[my_cluster]$ gives the total work load in the horizontal region of cluster $my_cluster$. Divide this value to the number of processors in $my_cluster$ to find the average amount of work load. Set $AvgLd = XWLA[my_cluster][L]/P$, where L is the number of vertical scanlines in x-dimension.
 6. Call $FIND_DIVISION(K, XWLA[my_cluster])$ procedure (given in Fig. 3.5) with the number of regions equal to number of processors in each cluster (K).
 7. Perform global concatenate operation to obtain the vertical partitions in other clusters.
-

Figure 3.6. Parallel algorithm for rectangular division scheme.

3.2.3 Recursive Heuristic

Recursive approach is the superset of the two initial algorithms and introduces a very general subdivision heuristic to the area. It divides the image plane into smaller rectangular regions gradually and uses the similar ideas presented in the previous two sections. The main algorithm for this heuristic can be found in Fig. 3.8 and an example division can be found in Fig. 3.9.

As it can be seen from the algorithm, initially we set the local image plane to our screen and then continue to divide it into subregions recursively into two, at each loop of the process. When we reach to one processor per plane limit we

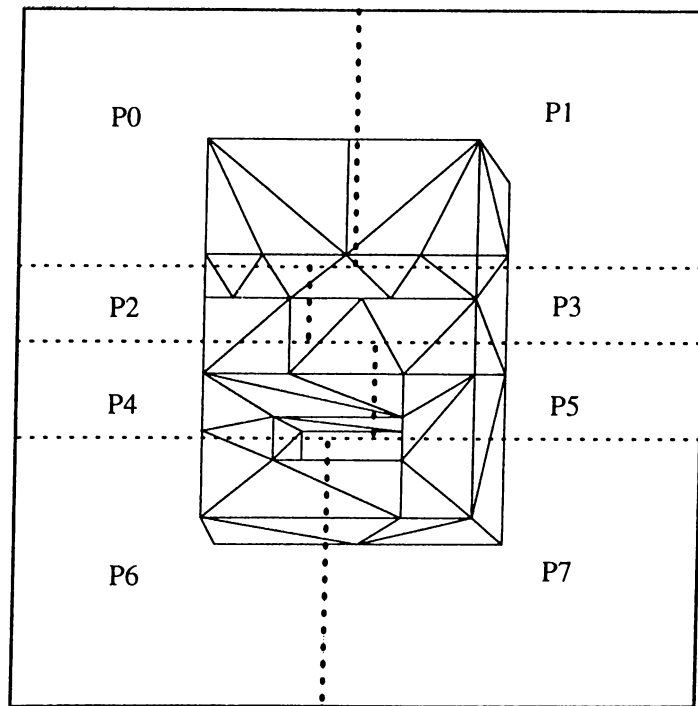


Figure 3.7. An example of rectangular division for eight processors organized into four clusters and two processors in each cluster. Dotted lines represent the boundaries of each region.

terminate and run the sequential rendering algorithm with our local data set. Note that, at each pass, the local image plane is divided into two, vertically or horizontally, which ever gives the best load balance. Also the local data within a region is found gradually as the data migrates from processor to processor at each pass of the algorithm. As one half of the processors are dealing with one region the other half deals with the other region. Hence, the total number of passes is equal to $\log_2(P)$ where P is the total number of processors.

Note that in rectangular and recursive subdivision schemes intra-scanline coherency is disturbed along with the inter-scanline one which causes extra processing time, to be overcome, generated because of the loss of active list information at the end of each region. This extra work versus the two dimensional scalable division flexibility should be compared for better understanding of the heuristics. The disturbance of coherencies causes job approximations within a region drastically deviated while increasing the scalability of the algorithm. The outcomes of these problems and advantages versus disadvantages between the three heuristics will be discussed in the following chapters along with the results obtained from the sample runs of the programs developed using these algorithms.

-
1. Find the overall load in the *local image plane*.
 2. Divide this load into two *horizontally*.
 3. Divide this load into two *vertically*.
 4. Compare these two division and choose the *best* one.
 5. *Exchange* local data with respect to this new division.
 6. Set half of the processors to one region and other half to the other region.
 7. Repeat this until one region per processor is reached.
-

Figure 3.8. Parallel algorithm for the recursive subdivision scheme.

Inter-scanline coherency can be disturbed by dividing a region horizontally. This causes the active polygon information introduced in one scanline to be lost and hence recreated. While intra-scanline coherency can be disturbed by dividing a region vertically which causes the active polygon, edge, x-bucket, and finally z-list information (formed incrementally by insertions which causes an overall insertion sort if repeated) in one pixel to be lost and hence recreated. This can be clearly seen from the sequential algorithm because the Scanline Z-Buffer algorithm is used as the underlying paradigm which obviously takes a single scanline as its atomic process. Hence, makes the intra-scanline coherency as a crucial paradigm to be preserved through out the algorithm.

3.3 Load Balancing Metrics

In many scientific applications, the volume data to be visualized is not regularly sampled and distributed in three dimensional space like in all of the non-Cartesian cases. Hence, the computational work load on the image space will also be irregularly distributed. In addition, different viewing locations will result in different work load distributions on the image space. Hence, a straight forward division of image plane into equal rectangular regions may result in very poor load balances among the processors due to the nature of the volume data. Therefore, an adaptive division of image plane into rectangular regions will generate better work load distributions and better processor utilizations. In this section you will find the necessary metrics to approximate the overall

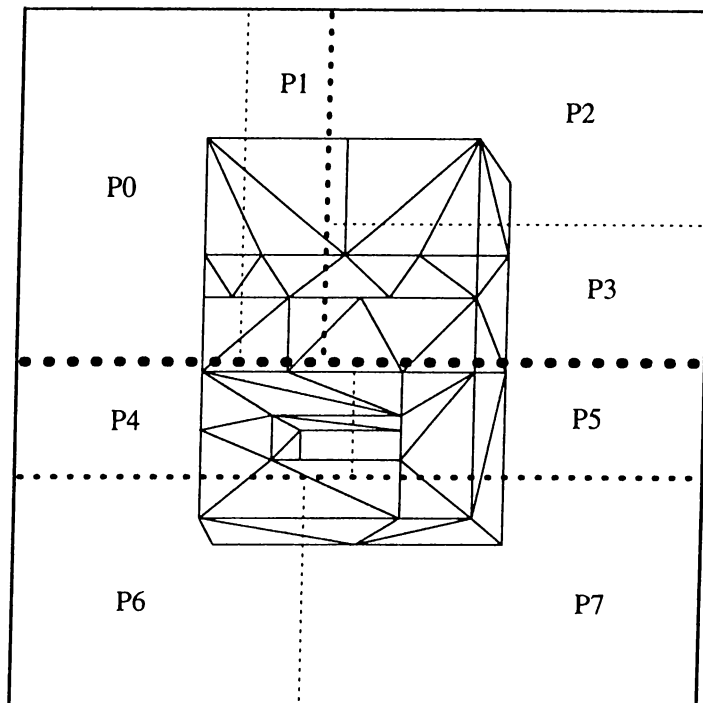


Figure 3.9. An example of recursive subdivision for eight processors. Dotted lines represent the boundaries of each region.

work load within a rectangular subregion.

There are three parameters that affect the computational work load in an image plane section. First one is the number of triangles, because the total work load due to clipping of a triangle to boundaries and insertion operations into y -bucket and active polygon lists are proportional to the number of triangles in a region. Second parameter is the number of scanlines each triangle spans. This parameter represents the computational work load associated with creation of x -spans, and insertion of these spans into x -bucket lists. The total number of pixels generated by rasterization of x -spans of a triangle is the third parameter affecting the computational load in a region. Each pixel generated adds computations required for interpolation and composition operations. By some sample runs of the systems developed the relative proportions for these metrics can be found and used in the calculation of the overall work load in a rectangular subregion for the future use of the same implementations. So the overall work will be defined as in Eq. 3.1. Using these formulas the divisions for an image plane can be approximated where each subregion introduces nearly same amount of work to its related processor. So better these formulas are better our division schemes are. The necessary thing to do here is to approximate the work load within a region as close as possible to its real value so that

a better load balance can be reached.

$$W = (a \times T) + (b \times S) + (c \times P) + E$$

$$A = W - E$$

W = Total real Work load

A = Total Approximated work load

E = Error introduced by approximations

T = Total number of Triangles

S = Total number of Spans

P = Total number of Pixels

a = Task done per triangle

b = Task done per span

c = Task done per pixel

The set of formulas defining the load metric. (3.1)

3.4 Refining the Algorithms

The heuristics and algorithms presented in this chapter can be refined by using the following refinements. First of all the load metric can be refined by introducing extra fields like *clipped polygons* and *clipped spans*. This is necessary as we are disturbing the coherencies in the sequential algorithms and adding extra processing time to subtasks by introducing new clipped structures into the data sets. So the new formulas will look like in Eq. 3.2. Note that further fields to this new set of formulas can be added if the sequential algorithm is analyzed in more detail. But this can increase the total division time while defining a better approximation for the real work done for a subregion. In the implementations of the algorithms only this many fields are used.

$$W = (a \times T) + (b \times S) + (c \times P) + (d \times CT) + (e \times CS) + E$$

$$A = W - E$$

W = Total real Work load

A = Total Approximated work load
 E = Error introduced by approximations
 T = Total number of Triangles
 S = Total number of Spans
 P = Total number of Pixels
 CT = Total number of Clipped Triangles
 CS = Total number of Clipped Spans
 a = Task done per triangle
 b = Task done per span
 c = Task done per pixel
 d = Task done per clipped triangle
 e = Task done per clipped span

The set of formulas defining the new refined load metric. (3.2)

Also a simple *rasterization* process can be used instead of the bounding box approximation in the algorithms which solves the problems coming from the approximations of the bounding box. The well-known *Z-Buffer* [21] algorithm is used for this simple rasterization process and each pixel covered by each triangle is found by using this algorithm. This especially reduces the errors coming from the vertical divisions as the vertical divisions create new spans and hence new edges to be rendered for the main algorithm. The distribution of this extra work introduced can only be done by finding the exact span values for a triangle which is the informal definition of rasterization. Another approach might be to make some initial runs for the data set introduced and then doing some re-load balancing scheme for further visualizations. This can only be done if multiple runs from a single view is needed, e.g. time-varying data sets.

Finally some minor refinements on the division heuristics can be made like, dividing the region by using a binary search to speed up the division process, re-calculating the work load for each region after each division is done in order to reduce the approximation errors, using multiple arrays for work load contributions of each item (like triangles) for better approximations, and using recursion in `FIND_DIVISION` procedure in order to avoid cumulative errors introduced by the linear division, all of which do not effect the overall run and

results of the algorithm as much as the previous refinements. So we are now ready to see some results about the sample runs of the programs developed for this thesis and then we can discuss these results using the algorithms and key points given in this chapter.

Chapter 4

Results

In this chapter, the results of the implementations done will be presented along with some auxiliary tools used (like data sets, architectures, etc.). In the first section of this chapter, you can find the definitions of the data sets and the architectures used. Then, the experimental results themselves will be given as another separate section. Various graphs are used to present the results obtained at the end of this chapter and some tables are given as Appendix. A.

4.1 Introduction to the Experimental Results

In this thesis, three basic data sets are used. These are namely the *blunt*, the *post*, and the *delta* data sets. These data sets are taken from the NASA-Ames Research Center obtained for the sake of visualization of the CFD (Computational Fluid Dynamics) simulation results. Both structured and unstructured versions of these sets are available at NASA-Ames Research Center. We have obtained the curvilinear form of these data sets and hence there exists an implicit connectivity in each of them defined by a 3D array structure. Therefore, the data sets were originally given as 3D hexahedral arrays where each grid point defines the density, heat, etc. values of the object to be visualized.

After obtaining these data sets we have converted them into unstructured standard tetrahedral (connectivity is explicit) 1D computational arrays. And then we have extracted the distinct triangles of the tetrahedrals from these data sets. Hence, as a final data set we have had a set of triangles for each data set where no connectivity is defined but each corner point points to a data

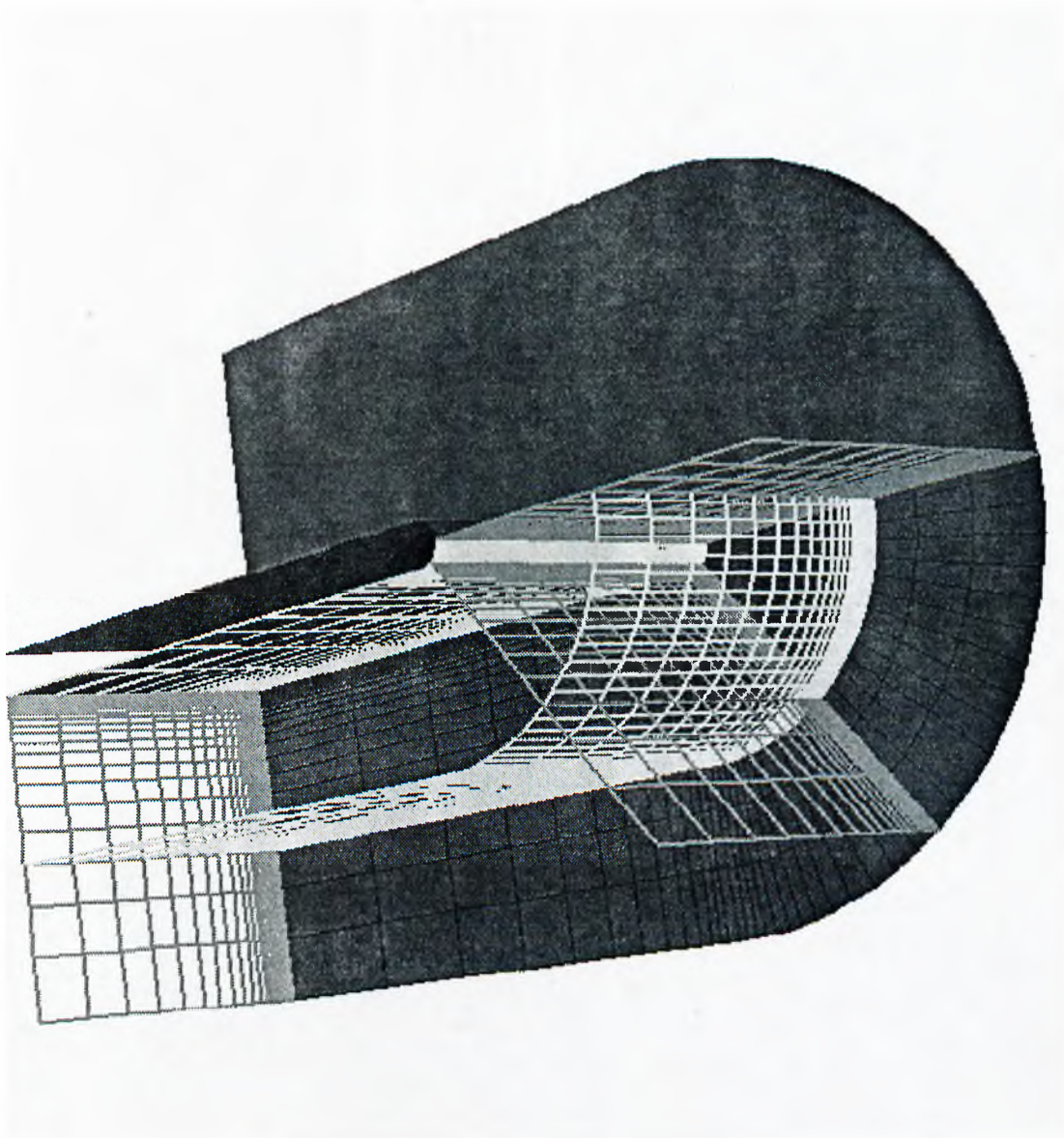


Figure 4.1. Input data BLUNT in hexahedral grid form.

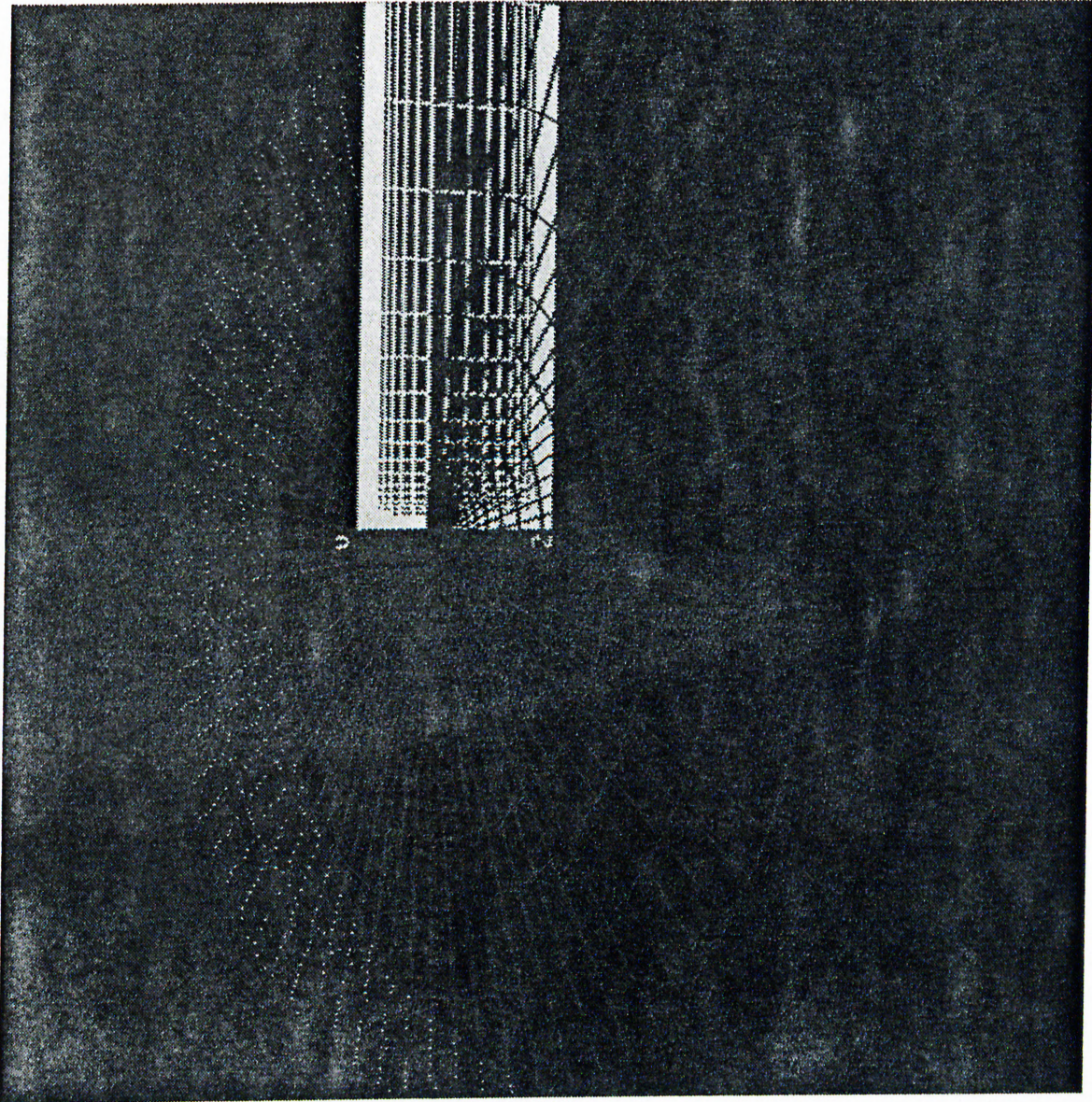


Figure 4.2. Input data PQST in hexahedral grid form.

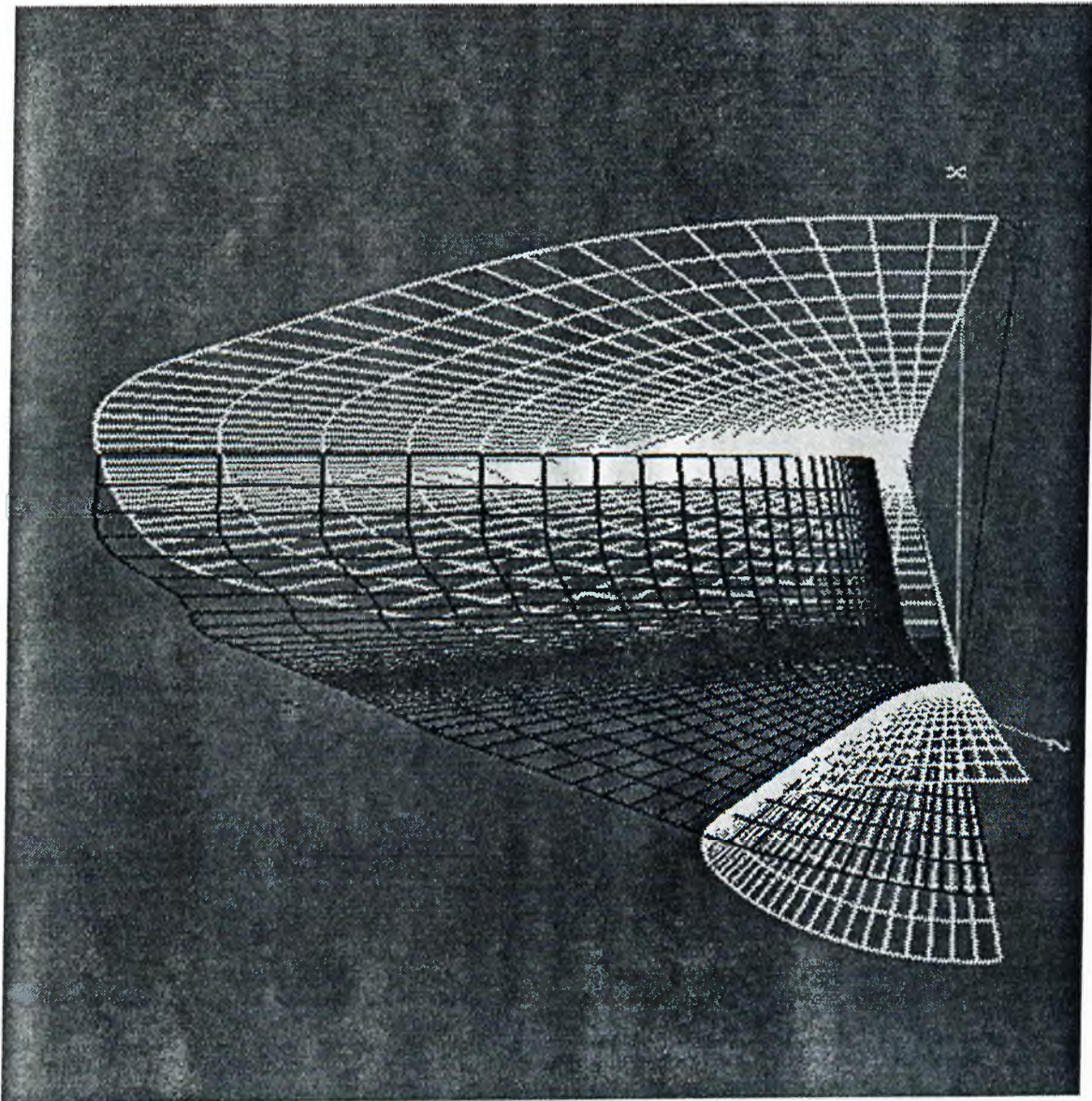


Figure 4.3. Input data DELTA in hexahedral grid form.

value to be visualized. So we now have a standard polygonal data set where the polygon corners are given in an indexed manner, pointers to data values for visualization.

Many runs of the programs developed are done using these data sets and the results are given in the following sections. As a final chapter the conclusions on the algorithms developed are given with respect to these runs. Now as a brief introduction for these data sets we will give some information about some of their interesting features.

The first data set Fig. 4.1 is named as *blunt* data set and defines the airflow over a flat plate with a blunt fin rising from the plate. Free stream flow is parallel to the plate and to the flat part of the fin, entirely in the x-component direction. The flow is assumed to be symmetrical about a plane through the center of the fin, so only one half of the real geometry is present in the data set and used in computations. The dimensions of this data set defined in its curvilinear format is $40 \times 32 \times 32$ grid points.

The second data set Fig. 4.2 is named as *post* data set and defines the liquid oxygen flow across a flat plate with a cylindrical post rising perpendicular to the plate (and therefore the flow). The simulation is modeling a flow internal to a rocket engine. A space shuttle launch vehicle engine has a region in which many such posts obstruct flow of liquid oxygen to promote better mixing. Since the fluid is incompressible, pressure is constant, and visualizations of these values are very well-defined and hence boring. Note that the areas of interest are the grid points closer to the post where much smaller and high quantized grids are defined. The dimensions of this data set defined in its curvilinear format is $38 \times 76 \times 38$ grid points.

The last data set Fig. 4.3 is named as *delta* data set and defines the flow past over a very simplified geometry representing a delta wing aircraft, at a moderately high angle of attack. Features of interest are vortices and vortex breakdowns. The grid is particularly twisted and scaled, and therefore makes a good test of certain features and capabilities of visualization systems used. The dimensions of this data set defined in its curvilinear format is $56 \times 54 \times 70$ grid points.

Implementations done for benchmarking results are made on an IBM-SP2 parallel computer with eight equivalent computational nodes and two powerful server nodes. This is a distributed memory MIMD architecture where the

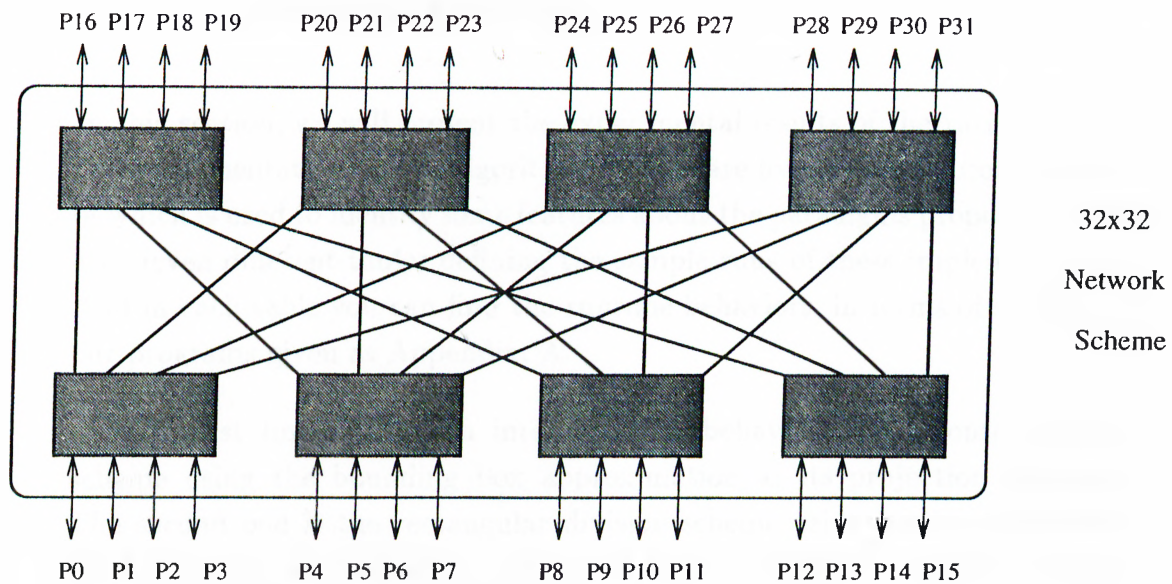


Figure 4.4. Interconnection network of the IBM-SP2 Architecture.

thin (parallel) nodes have two means of interprocessor communication. The eight nodes have a *High Performance Interconnection Network* (HPIN) used for high bandwidth, real-time communications, and also an *Ethernet* (E) connection for regular PVM purposes or independent usage purposes. Each node is a 64Mbyte main memory RISC/6000 processor with direct connections to the HPIN switches. *Wide* (server) nodes are powerful 512Mbyte memory general purpose nodes having the same architecture with thin nodes except their parallel usages. These wide nodes are not directly used in this work but only the thin nodes are used in the benchmarking results.

The interconnection networking Fig. 4.4 scheme is very helpful in the data swap steps of the algorithms used, if and only if it is used cleverly in the algorithms developed. That is to say the programmer of these switches should check the communicating routines of his/her programs carefully in order to prevent switch congestions. In our implementations congestions are prevented by using a communication scheme where each processor communicates with another by defining and then using an empty link and hence avoiding the switch congestions. The routines represented for this scheme can easily be changed by just changing the formula defining the two communicating nodes at a time. Hence, the programs developed are quite portable to other multicomputer architectures. Now, we think that we are ready to see some experimental results along with some deeper comments and explanations on the runs of the programs used in the following sections.

4.2 Experimental Results

In this section, we will present the experimental results of the various runs of the implementations of our algorithms. There are five main test programs each of which is used to identify some features about the paradigms proposed. There are eleven different tables defining the sample runs of these implementations. And in each table you can find the runtime behaviors, in terms of seconds, of our programs given as Appendix. A.

The first implementation interprets the behavior of horizontal division scheme using the bounding box approximation as its projection approach. The second one is the rectangular division scheme using, again, the bounding box projection paradigm. The third is the recursive one given with the similar projection approximation. Each of these implementations are run on three different data sets named as blunt, post, and delta data sets. The first nine tables are the sample runs of these three implementations on these three different data sets. The last two tables are the runs of the final two implementations on blunt data set which use horizontal and recursive division schemes (respectively) along with the rasterization heuristic. These eleven tables given in Appendix. A can be used for further deductions about the sample runs of the implementations. The overall behavior of the algorithms and the general deductions about the paradigms proposed are represented as a set of graphs as follows.

Nine graphs which show the general behavior of the algorithms are given using these tables. First three are the speedup, the second three are the efficiency and the last three are the load-balance graphs. The Fig. 4.5 is the speedup graph of the rendering process of the bounding box approximations. This graph includes ideal, horizontal, rectangular, and recursive division cases. The Fig. 4.6 shows speedups of the rasterized versions of the horizontal and recursive schemes along with the ideal case, using the rendering times. The Fig. 4.7 shows the total execution time behavior of the horizontal scheme with bounding box approximation, represented as a speedup graph. The Fig. 4.8, the Fig. 4.9, and the Fig. 4.10 are the efficiency versions of the speedup graphs, respectively, showing how efficiently the processors are used. The Fig. 4.11, the Fig. 4.12, and the Fig. 4.13 are the load-balance graphs showing the load distributions of the same speedup graphs, respectively. Note that speedups are defined as the times of the single processor runs over the times of the latest

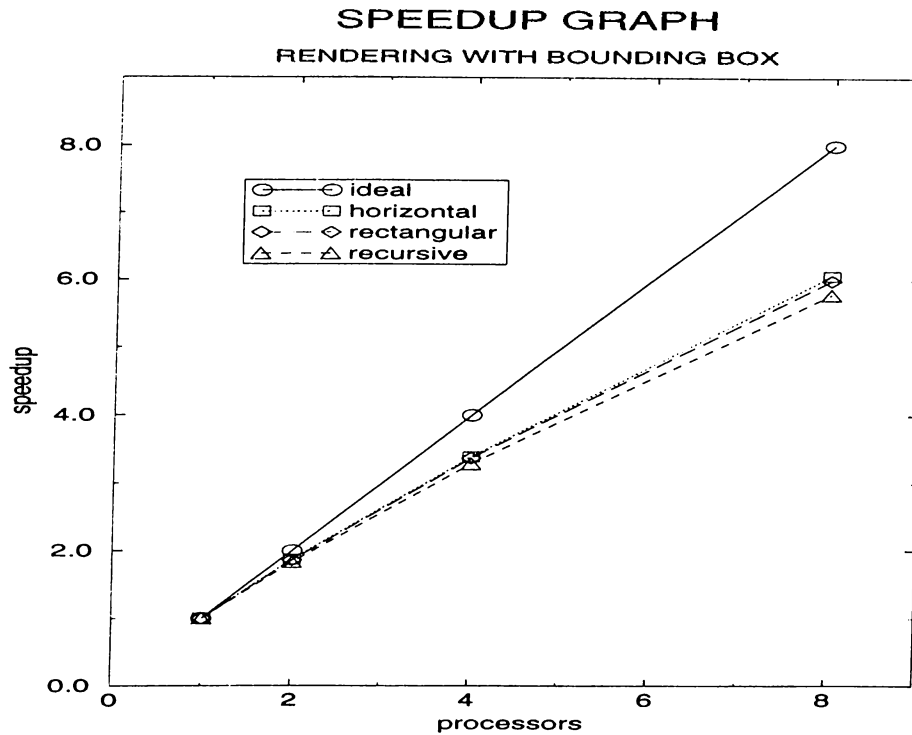


Figure 4.5. Speedup graph of the horizontal, rectangular, and recursive division schemes where bounding box approximation is used and obtained by using the rendering times.

completed sub-tasks of the related runs. The efficiency is the speedup value over the number of processors used for that run which is then converted into the percent format. And load-balance is found by first finding the load-imbalance and then subtracting this value from one hundred where load-imbalance can be found in various ways. The version used in this thesis is represented as Eq. 4.1. All of these values are found by averaging the related run-tables of the implementations used. Now we are ready for some deductions about these graphs which are given in the last chapter as a set of conclusions.

$$L_b = ((t_{max} - t_{min})/t_{max}) \times 100$$

$$L_b = \text{Load-imbalance}$$

t_{max} = execution time of the processor that completes the latest

t_{min} = execution time of the processor that completes the earliest

The formula defining the load-imbalance. (4.1)

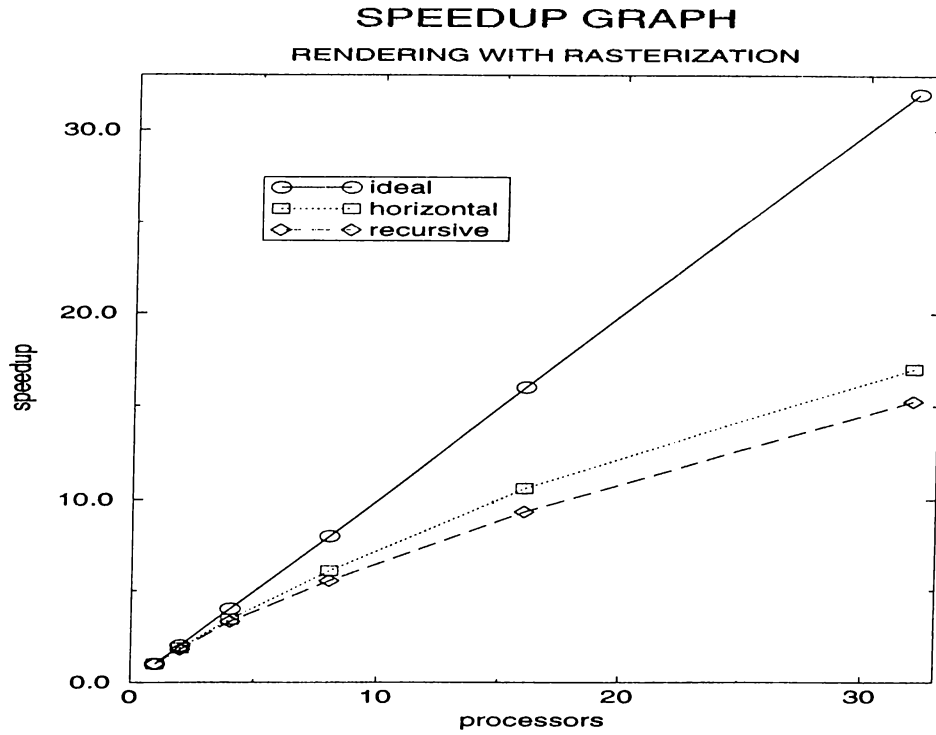


Figure 4.6. Speedup graph of the horizontal and recursive division schemes where rasterization algorithm is used and obtained by using the rendering times.

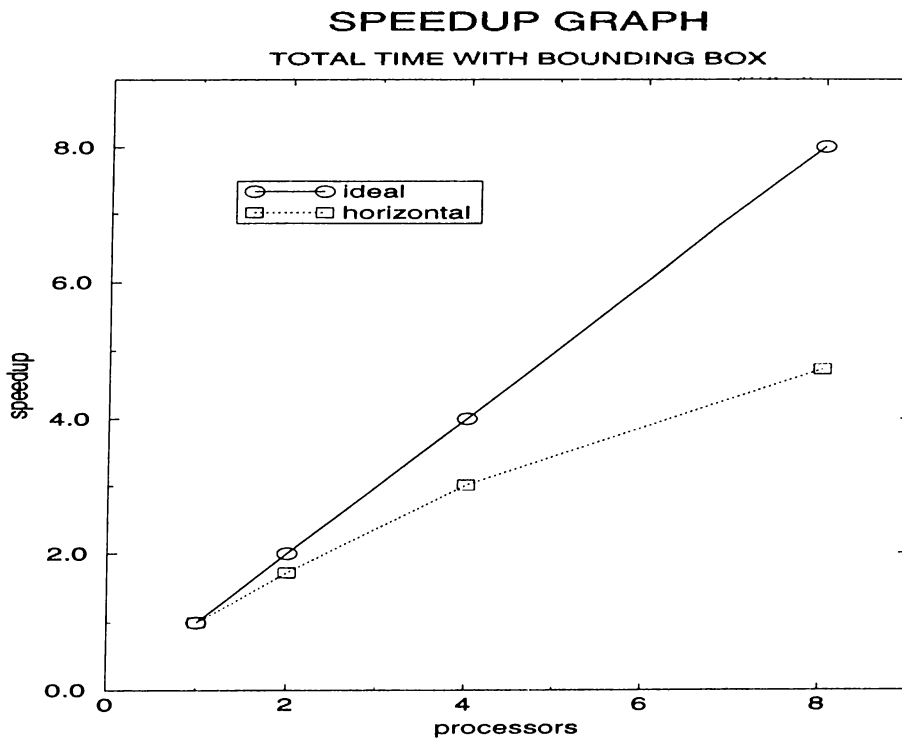


Figure 4.7. Speedup graph of the horizontal division scheme where bounding box approximation is used and obtained by using the overall execution times.

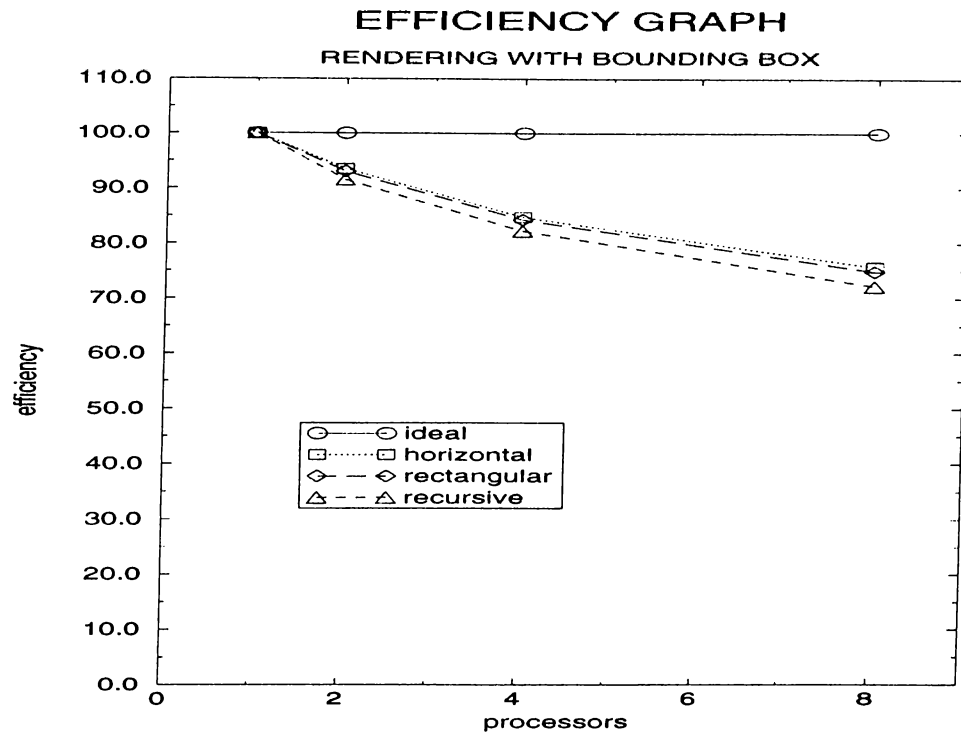


Figure 4.8. Efficiency graph of the horizontal, rectangular, and recursive division schemes where bounding box approximation is used and obtained by using the rendering times.

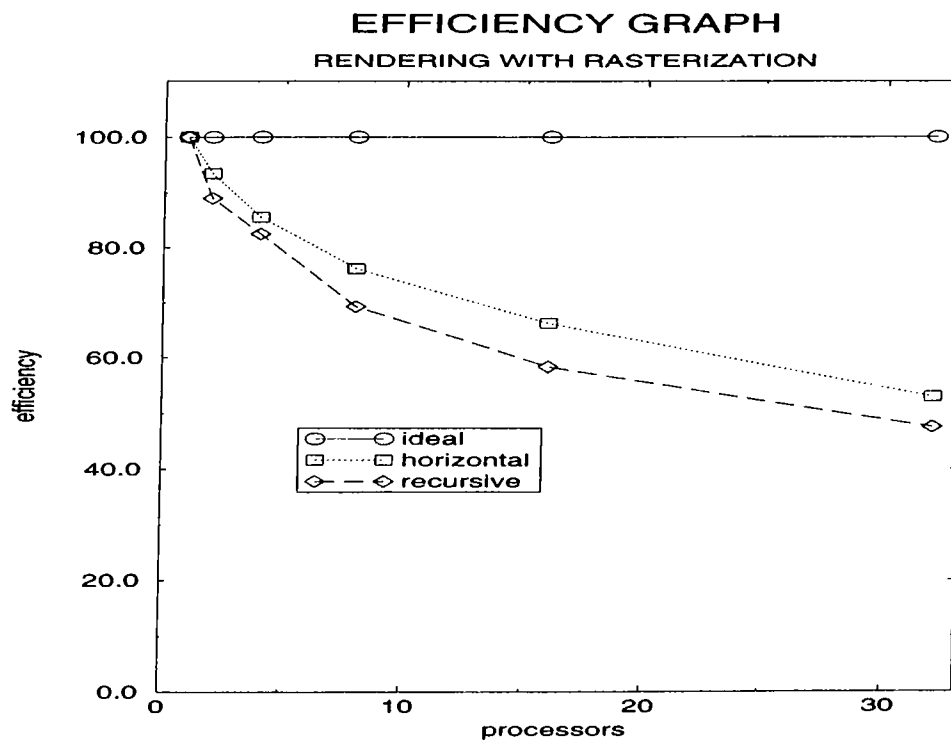


Figure 4.9. Efficiency graph of the horizontal and recursive division schemes where rasterization algorithm is used and obtained by using the rendering times.

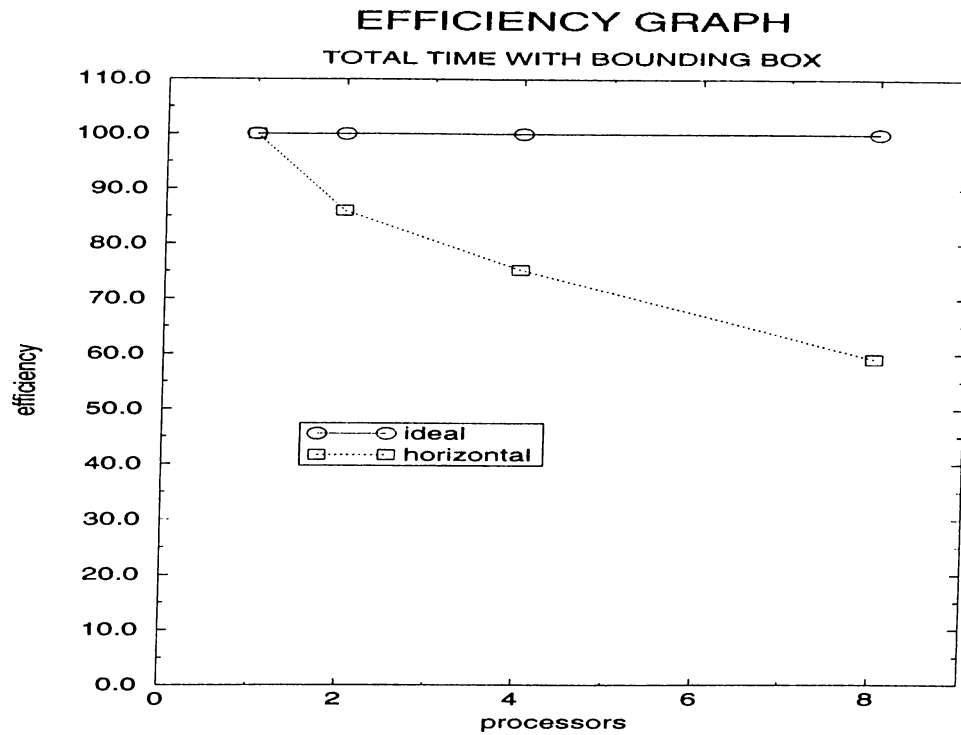


Figure 4.10. Efficiency graph of the horizontal division scheme where bounding box approximation is used and obtained by using the overall execution times.

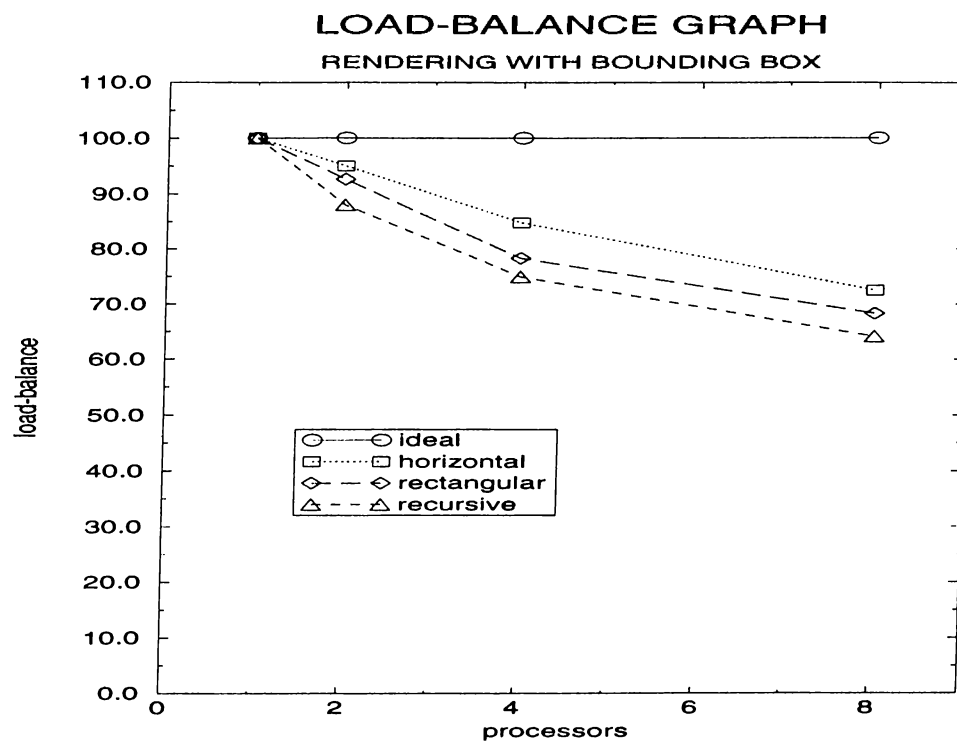


Figure 4.11. Load-balance graph of the horizontal, rectangular, and recursive division schemes where bounding box approximation is used and obtained by using the rendering times.

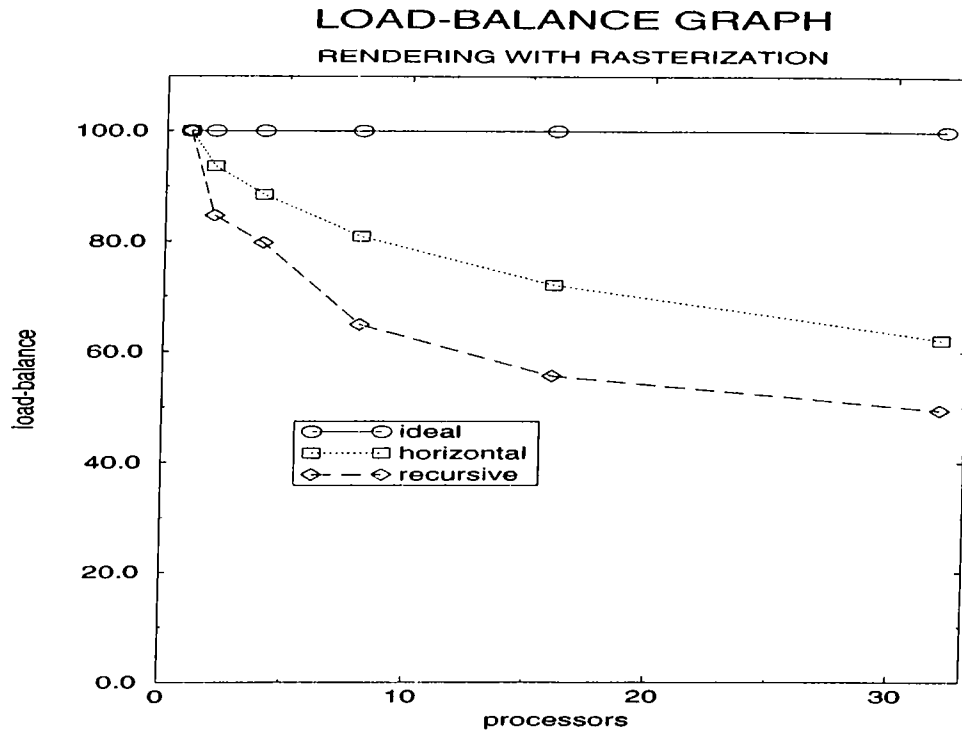


Figure 4.12. Load-balance graph of the horizontal and recursive division schemes where rasterization algorithm is used and obtained by using the rendering times.

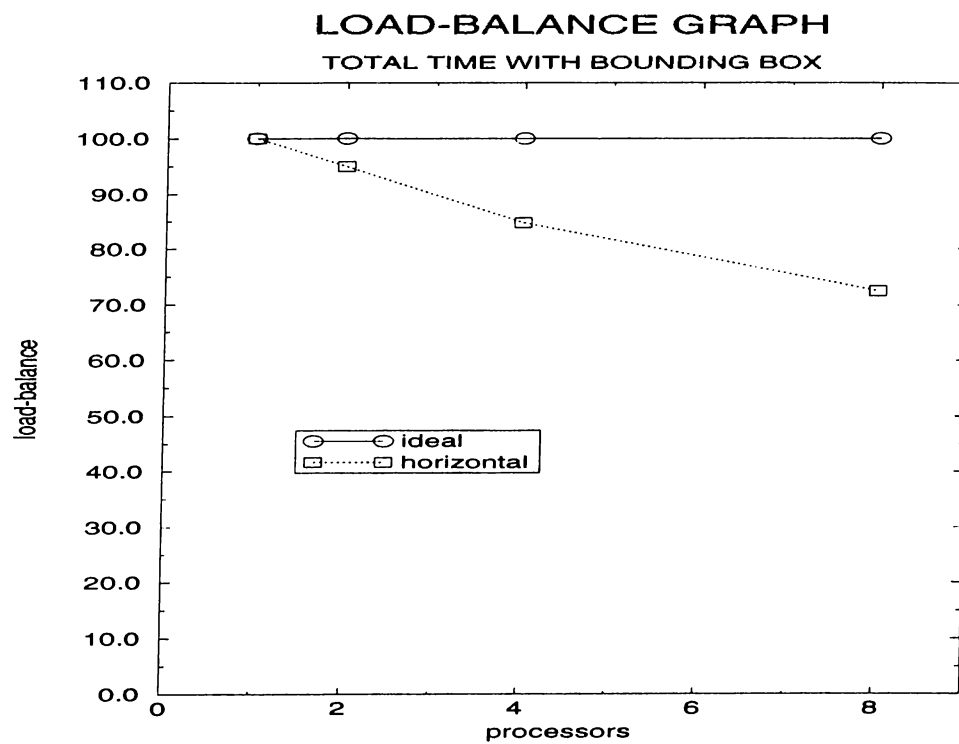


Figure 4.13. Load-balance graph of the horizontal division scheme where bounding box approximation is used and obtained by using the overall execution times.

Chapter 5

Conclusion

In this thesis, we have investigated various image space subdivision algorithms for parallel direct volume rendering of volumetric data sets on distributed memory multicomputers using the ray shooting paradigm. We have used a standard polygon rendering algorithm as our underlying sequential schema and obtained results on three different data sets using an IBM-SP2 architecture. Now, we are ready to give our conclusions, discuss in general our results, and assign some future solutions to our problems. We hope that this comparative work would be a good starting point for various other works on this topic.

We have seen that our subdivision heuristics do not give very good load balancing and speedup results with respect to [1, 2]. In fact these two papers study the shared memory approach of the same parallel algorithm and hence can not directly be compared with this work. But as there is no other publications done specifically in this field we can only compare our results with our initial expectations, theoretical possible results, and with [1, 2]. Although our test data sets are of restricted type we think that most of the results for other data types will be the same. Although the algorithms developed in this thesis are generalized for all of the data types known to us, this generalization, which uses the standard polygon rendering algorithms as their base, causes some problems in the parallelization steps of the algorithms because of their low scalability on volumetric data sets. Therefore the assumption of ease at parallel polygon rendering and load balancing was not correct.

The reasons for these results are the low scalability of the image space subdivisions which can be solved by using further division schemes on the 3rd dimension that can introduce some overheads like pixel merging, difficulties in

defining a good load balancing metric in the static decompositions which can be solved by using a dynamic strategy where probably there are lots of sub-tasks to be completed while the number of processors used is much smaller than this number of tasks (increases duplications and communications) or using a re-balancing scheme in the rendering process that can increase the completion time while introducing a better load balance to the runs of the algorithms, lack of information about the geometry to be divided, causing scalability to be disturbed which can be solved by using the number of sampling points with in a region (e.g. avoiding many small triangles from going to a single processor and other normal or big ones, relatively, to another processor), and the difficulty of defining a good metric for a region because of the extra tasks introduced especially at vertical divisions which can be solved by taking more well defined ratios for atomic means of task decomposition (like spans) by taking exhaustive sample runs of the algorithms for ratio determination (currently nearly impossible because of lack of data types and computer hardware usage).

Also we have observed that although the first heuristic is a very naive one it beats the others in most of the runs that we have done in terms of load balances and speedups. The reason for this is that the vertical divisions introduce much more expensive extra tasks to do after the division, than the horizontal ones, that are also very difficult to estimate and distribute (because of the memory allocation problems of huge data groups and extra sorting used in the composition steps). But note that for very large number of processors or where the image height is less than or nearly equal to the number of processors that we are using, we can use the 2nd and especially the 3rd heuristics because of their higher scalability (compared to 1D horizontal one) and their flexibility introduced in the division schemes although it can introduce huge extra overheads. For example, you can not divide a 32×32 image to 64 processors horizontally adequately because of the definition of an atomic task in horizontal scheme which is a single scanline (therefore causing some processors to stay idle while others are working hard to finish their assigned subtasks). But on the other hand the recursive scheme has $32 \times 32 = 1024$ different division possibilities for the same image which can beat the overheads introduced and even give better results than the horizontal one. This is true especially if the data to be visualized is longer in its width than its height and the number of processors is nearly equal to this height. So we can say that in general (for practical usages) horizontal scheme beats the others except some of the cases defined as above. Therefore, usage of the horizontal scheme, besides its division results, by also introducing a smaller division pre-processing overhead, smaller communication

times, and simplicity can be selected for various visualization schemes.

In addition to these we can say that we can use rasterization type optimizations in the division schemes which introduces a huge overhead while giving a slightly better load balance or speedup (in terms of the rendering process but not the overall time). These type of divisions can be used especially with time-varying data sets where more than one visualization of the same view is possible. Also load re-balancing after each run can be done as a future work. Besides animation type visualizations can be very adequate for the algorithms developed with in this thesis because of their incremental view point changes, etc. (e.g., controlling the data flow after each run). Some extensions to this methodology could be made but we advise that the underlying schemata should be changed (for example to shared memory architectures or etc.) for more innovative results and solutions. Finally, we can say that this work can be named as a good initial work for future works on this field of research. The presentation of this thesis along with its results will hopefully be very helpful to all the other researchers in this area.

Appendix A

Tables of the Sample Runs

This appendix gives the tables of the sample runs of the implementations made. Each table includes runs, on the defined data set, of three different views of the data set, three different screen resolutions, and some different number of processors (from up to 8 to up to 32). Tables include the total, rendering, swapping, and the division times of the programs. They also give the maximum and minimum values for each time. Due to usage restrictions of the IBM-SP2 architecture that we are using some runs are limited with only one data set and some are only given up to 8 processors. But all runs give information about the 1, 2, 4, and 8 processor tests of the programs.

The Table. A.1 is the first table. It is generated using the blunt data set. There are three different views of the data which are taken randomly to obtain a general feeling about the test programs. Three different image plane sizes, namely 256×256 , 512×512 , and 1024×1024 , are used. This table uses horizontal division scheme and shows this behavior by using up to 8 processors. The Table. A.2 and the Table. A.3 use post and delta data sets respectively where the other properties are the same with the first table.

The Table. A.4 is the fourth table. It is generated using the blunt data set. There are three different views of the data which are taken randomly to obtain a general feeling about the test programs. Three different image plane sizes, namely 256×256 , 512×512 , and 1024×1024 , are used. This table uses rectangular division scheme and shows this behavior by using up to 8 processors. The Table. A.5 and the Table. A.6 use post and delta data sets respectively where the other properties are the same with the fourth table.

The Table. A.7 is the seventh table. It is generated using the blunt data set. There are three different views of the data which are taken randomly to obtain a general feeling about the test programs. Three different image plane sizes, namely 256×256 , 512×512 , and 1024×1024 , are used. This table uses recursive division scheme and shows this behavior by using up to 8 processors. The Table. A.8 and the Table. A.9 use post and delta data sets respectively where the other properties are the same with the seventh table. Note that these last three tables show division times (the time used to distribute the process into sub-processes) and the swapping times (the time used to exchange the local data of the processors) differently then the first six tables. As this version of the implementations use recursion, the division time is in fact equivalent to the time of dividing plus swapping where pure division time is equal to the one in the table minus the swapping time. But also note that the intermediate swaps are in fact crucial for the division process to be complete which can also be seen as a part of the division time except the last swap operation.

The last two tables show us the behavior of the rasterized versions of the heuristics, named as horizontal and recursive, given as Table. A.10 and Table. A.11 respectively. Note that these tables use up to 32 processors for the sake of completeness of the deductions about these runs. The division and swap (and hence the total) times given are not that much accurate, as the division process for rasterization is much longer than the bounding box approach and causes various errors for these two tables. You must have noticed that there are some fluctuating outputs in the first nine tables which are much more drastical in these last two tables. The reason for this is the non-dedicated usage of the IBM-SP2 computer which causes swaps in some sample runs. Although we have used the user time for executions, the communicating steps of the algorithms suffers from system swaps where one processor might be waiting for the other (hence causing some CPU time bursts). As the rasterized versions of the algorithms developed need more communications the times for these are faultier than the others, so a careful reader should mainly consider the rendering times for the last two tables for further deductions. Also note that the rasterized versions can (if used in dedicated modes) only be used if the input data sets are time-varying. The reason for this is these type of data sets are used for multiple visualizations of the same view of an input data set and hence causing the division overheads negligible for about e.g. twenty runs of the same view. Also you must have noticed that the 16 and 32 processor runs are simulations by 8 processors as the original computer have only 8 real processors. So the communications are made by the ethernet protocol (but

RESULTS IN SECONDS			BLUNT DATA SET							
EXECUTION TIMES			TOTAL		RENDERING		SWAPPING		DIVIDING	
VIEW	RESOLUTION	P	MIN	MAX	MIN	MAX	MIN	MAX	MIN	MAX
1st	256 × 256	1	40.79	40.79	40.79	40.79	0.00	0.00	0.00	0.00
1st	256 × 256	2	26.68	29.57	20.05	23.02	2.48	2.54	3.69	3.80
1st	256 × 256	4	15.16	17.59	10.22	12.54	2.77	2.81	1.94	2.10
1st	256 × 256	8	10.30	12.07	5.60	7.11	3.15	3.31	1.23	1.56
2nd	256 × 256	1	22.50	22.50	22.50	22.50	0.00	0.00	0.00	0.00
2nd	256 × 256	2	16.95	17.16	12.01	12.20	1.80	1.83	2.84	2.94
2nd	256 × 256	4	10.26	11.25	6.19	7.08	2.25	2.28	1.61	1.78
2nd	256 × 256	8	7.09	8.28	3.37	4.21	2.61	2.89	0.99	1.22
3rd	256 × 256	1	26.92	26.92	26.92	26.92	0.00	0.00	0.00	0.00
3rd	256 × 256	2	18.03	19.63	13.33	15.05	1.57	1.61	2.83	2.85
3rd	256 × 256	4	10.19	12.18	6.54	8.43	1.98	2.02	1.51	1.67
3rd	256 × 256	8	6.67	8.32	3.59	4.87	2.13	2.33	0.88	1.11
1st	512 × 512	1	115.39	115.39	115.39	115.39	0.00	0.00	0.00	0.00
1st	512 × 512	2	66.97	67.45	59.50	59.71	2.85	2.92	4.02	4.63
1st	512 × 512	4	34.97	37.71	29.46	32.11	3.05	3.10	2.17	2.78
1st	512 × 512	8	18.84	22.69	14.23	17.85	3.23	3.31	1.21	1.64
2nd	512 × 512	1	65.01	65.01	65.01	65.01	0.00	0.00	0.00	0.00
2nd	512 × 512	2	38.60	40.92	32.43	34.63	2.34	2.40	3.50	3.57
2nd	512 × 512	4	21.38	24.40	16.24	19.35	2.82	2.92	1.98	2.13
2nd	512 × 512	8	12.93	15.66	8.14	11.03	3.14	3.45	1.30	1.64
3rd	512 × 512	1	81.76	81.76	81.76	81.76	0.00	0.00	0.00	0.00
3rd	512 × 512	2	46.92	48.59	40.91	42.76	2.19	2.19	3.40	3.50
3rd	512 × 512	4	24.87	27.35	20.21	22.69	2.55	2.58	1.87	2.02
3rd	512 × 512	8	14.15	16.41	10.38	12.44	2.51	2.68	1.02	1.25
1st	1024 × 1024	1	353.49	353.49	353.49	353.49	0.00	0.00	0.00	0.00
1st	1024 × 1024	2	183.18	191.95	175.34	183.66	3.07	3.16	4.50	4.51
1st	1024 × 1024	4	92.98	102.91	87.25	96.90	3.21	3.25	2.40	2.40
1st	1024 × 1024	8	46.99	56.85	42.18	51.98	3.26	3.33	1.31	1.69
2nd	1024 × 1024	1	202.79	202.79	202.79	202.79	0.00	0.00	0.00	0.00
2nd	1024 × 1024	2	106.82	113.90	99.75	106.46	2.78	2.82	4.05	4.09
2nd	1024 × 1024	4	54.57	62.47	48.81	56.62	3.24	3.30	2.36	2.38
2nd	1024 × 1024	8	29.28	34.91	24.01	30.00	3.38	3.68	1.31	1.54
3rd	1024 × 1024	1	257.06	257.06	257.06	257.06	0.00	0.00	0.00	0.00
3rd	1024 × 1024	2	136.68	139.30	129.52	132.45	2.56	2.67	4.03	4.06
3rd	1024 × 1024	4	68.54	73.91	63.23	68.59	2.90	2.96	2.10	2.26
3rd	1024 × 1024	8	36.01	41.04	31.81	36.86	2.80	2.94	1.18	1.46

Table A.1. *Horizontal division results using bounding box with blunt data.*

not with high performance switches) causing an extra error in communication times, because of this simulation restriction. Finally note that the last two versions of the implementations are run only on the blunt data set as they took very long execution times (in dedicated mode about a month with 16 and 32 processor simulations and more than three months in non-dedicated mode) in the IBM-SP2 computer that we are using.

RESULTS IN SECONDS			POST DATA SET							
EXECUTION TIMES			TOTAL		RENDERING		SWAPPING		DIVIDING	
VIEW	RESOLUTION	P	MIN	MAX	MIN	MAX	MIN	MAX	MIN	MAX
1st	256 × 256	1	61.40	61.40	61.40	61.40	0.00	0.00	0.00	0.00
1st	256 × 256	2	42.53	43.61	31.40	32.08	3.43	3.45	6.48	8.26
1st	256 × 256	4	23.60	24.45	16.46	17.07	3.61	3.65	3.33	3.56
1st	256 × 256	8	14.26	15.98	8.18	9.81	3.99	4.02	1.99	2.22
2nd	256 × 256	1	31.01	31.01	31.01	31.01	0.00	0.00	0.00	0.00
2nd	256 × 256	2	26.74	27.21	15.85	16.89	3.20	3.21	6.17	7.70
2nd	256 × 256	4	15.99	17.23	8.38	9.35	3.93	3.95	3.48	3.63
2nd	256 × 256	8	10.55	11.70	4.49	5.39	3.81	4.04	1.94	2.18
3rd	256 × 256	1	22.53	22.53	22.53	22.53	0.00	0.00	0.00	0.00
3rd	256 × 256	2	21.74	22.78	11.61	12.45	2.86	2.88	6.01	7.88
3rd	256 × 256	4	13.60	14.99	6.07	7.47	3.76	3.78	3.52	3.71
3rd	256 × 256	8	8.86	10.84	2.93	4.60	3.76	4.02	1.86	2.12
1st	512 × 512	1	175.11	175.11	175.11	175.11	0.00	0.00	0.00	0.00
1st	512 × 512	2	101.63	109.58	88.77	88.98	4.29	11.88	7.76	8.04
1st	512 × 512	4	54.28	58.47	44.03	47.81	4.98	5.07	4.16	6.19
1st	512 × 512	8	29.28	33.40	21.79	25.15	4.98	5.15	2.35	2.97
2nd	512 × 512	1	89.68	89.68	89.68	89.68	0.00	0.00	0.00	0.00
2nd	512 × 512	2	59.11	60.40	45.15	46.86	4.86	4.87	8.02	8.39
2nd	512 × 512	4	33.67	36.74	22.94	25.59	5.80	5.86	4.69	4.85
2nd	512 × 512	8	20.03	23.16	11.44	14.22	5.62	5.95	2.55	2.74
3rd	512 × 512	1	61.91	61.91	61.91	61.91	0.00	0.00	0.00	0.00
3rd	512 × 512	2	45.44	45.60	31.13	33.11	4.25	4.28	7.67	9.42
3rd	512 × 512	4	25.72	29.65	14.93	18.76	5.57	5.62	4.84	5.03
3rd	512 × 512	8	15.09	19.34	6.93	10.69	5.52	5.82	2.48	2.72
1st	1024 × 1024	1	546.84	546.84	546.84	546.84	0.00	0.00	0.00	0.00
1st	1024 × 1024	2	294.08	294.13	274.68	276.60	6.47	6.49	10.07	12.03
1st	1024 × 1024	4	144.06	160.70	131.38	147.42	6.75	6.93	5.58	5.66
1st	1024 × 1024	8	75.67	87.83	65.35	77.21	6.66	6.93	3.00	3.62
2nd	1024 × 1024	1	261.28	261.28	261.28	261.28	0.00	0.00	0.00	0.00
2nd	1024 × 1024	2	150.15	153.10	132.78	133.69	6.47	6.48	10.06	12.02
2nd	1024 × 1024	4	77.34	89.34	63.36	75.03	7.65	7.71	5.91	6.25
2nd	1024 × 1024	8	41.57	52.37	30.70	40.67	7.30	7.70	3.28	3.56
3rd	1024 × 1024	1	184.77	184.77	184.77	184.77	0.00	0.00	0.00	0.00
3rd	1024 × 1024	2	110.80	112.69	92.46	96.46	5.94	5.96	9.58	11.50
3rd	1024 × 1024	4	57.43	67.85	43.31	53.84	7.55	7.61	5.97	6.10
3rd	1024 × 1024	8	29.51	40.61	18.65	29.32	7.42	7.70	3.19	3.42

Table A.2. Horizontal division results using bounding box with post data.

RESULTS IN SECONDS			DELTA DATA SET							
EXECUTION TIMES			TOTAL		RENDERING		SWAPPING		DIVIDING	
VIEW	RESOLUTION	P	MIN	MAX	MIN	MAX	MIN	MAX	MIN	MAX
1st	256 × 256	1	75.72	75.72	75.72	75.72	0.00	0.00	0.00	0.00
1st	256 × 256	2	60.78	61.21	34.89	42.42	6.32	13.78	10.53	11.79
1st	256 × 256	4	31.76	41.65	16.29	24.16	8.00	8.07	6.40	8.81
1st	256 × 256	8	26.60	32.41	7.77	14.37	12.19	13.28	4.59	5.45
2nd	256 × 256	1	60.89	60.89	60.89	60.89	0.00	0.00	0.00	0.00
2nd	256 × 256	2	47.84	62.59	31.04	33.19	4.61	18.81	9.98	10.92
2nd	256 × 256	4	30.34	33.08	15.70	18.03	6.75	6.84	6.44	8.09
2nd	256 × 256	8	25.48	27.82	8.55	10.29	11.20	12.16	4.32	5.20
3rd	256 × 256	1	35.52	35.52	35.52	35.52	0.00	0.00	0.00	0.00
3rd	256 × 256	2	37.92	55.10	18.36	19.27	7.69	25.57	9.16	10.98
3rd	256 × 256	4	22.06	25.70	8.54	11.27	6.75	6.79	5.23	7.20
3rd	256 × 256	8	20.01	22.49	4.36	7.15	10.33	11.33	4.30	4.96
1st	512 × 512	1	219.86	219.86	219.86	219.86	0.00	0.00	0.00	0.00
1st	512 × 512	2	150.83	186.68	93.83	127.40	7.36	76.10	13.67	16.08
1st	512 × 512	4	62.93	89.83	42.45	67.20	10.40	11.19	8.86	10.88
1st	512 × 512	8	42.00	60.55	18.81	37.91	14.66	15.89	5.04	7.20
2nd	512 × 512	1	177.38	177.38	177.38	177.38	0.00	0.00	0.00	0.00
2nd	512 × 512	2	114.46	144.34	88.86	93.32	6.19	42.50	12.15	13.06
2nd	512 × 512	4	59.76	71.04	42.06	51.35	9.16	9.25	7.96	10.00
2nd	512 × 512	8	41.54	48.41	20.15	27.94	14.05	15.20	5.21	6.08
3rd	512 × 512	1	96.82	96.82	96.82	96.82	0.00	0.00	0.00	0.00
3rd	512 × 512	2	71.19	91.34	47.20	52.01	5.36	31.73	11.55	12.49
3rd	512 × 512	4	39.17	51.12	21.50	31.08	9.22	9.32	7.80	10.04
3rd	512 × 512	8	30.38	37.30	10.45	17.93	13.11	14.35	5.41	6.07
1st	1024 × 1024	1	650.86	650.86	650.86	650.86	0.00	0.00	0.00	0.00
1st	1024 × 1024	2	399.96	414.99	272.74	380.83	10.38	108.31	18.25	19.68
1st	1024 × 1024	4	167.75	233.65	121.88	204.25	10.91	35.78	10.73	12.94
1st	1024 × 1024	8	78.98	135.72	55.29	111.87	15.86	16.81	5.62	7.03
2nd	1024 × 1024	1	532.26	532.26	532.26	532.26	0.00	0.00	0.00	0.00
2nd	1024 × 1024	2	317.37	329.49	250.79	291.90	7.83	63.71	14.40	15.22
2nd	1024 × 1024	4	138.07	174.96	116.27	150.83	11.64	11.83	9.64	11.81
2nd	1024 × 1024	8	78.38	104.66	55.06	81.57	15.51	16.68	5.57	6.81
3rd	1024 × 1024	1	285.92	285.92	285.92	285.92	0.00	0.00	0.00	0.00
3rd	1024 × 1024	2	176.17	203.28	138.05	152.88	7.10	50.87	13.44	14.35
3rd	1024 × 1024	4	84.26	113.85	62.88	89.64	11.34	11.53	9.54	11.73
3rd	1024 × 1024	8	52.97	72.94	30.38	51.17	14.51	15.83	5.99	6.66

Table A.3. Horizontal division results using bounding box with delta data.

RESULTS IN SECONDS			BLUNT DATA SET							
EXECUTION TIMES			TOTAL		RENDERING		SWAPPING		DIVIDING	
VIEW	RESOLUTION	P	MIN	MAX	MIN	MAX	MIN	MAX	MIN	MAX
1st	256 × 256	1	40.79	40.79	40.79	40.79	0.00	0.00	0.00	0.00
1st	256 × 256	2	26.65	29.56	20.06	23.02	2.47	2.54	3.65	3.79
1st	256 × 256	4	15.13	18.18	10.06	12.96	2.78	2.80	2.02	2.25
1st	256 × 256	8	9.69	12.08	5.31	7.57	3.06	3.22	1.06	1.34
2nd	256 × 256	1	22.50	22.50	22.50	22.50	0.00	0.00	0.00	0.00
2nd	256 × 256	2	16.96	17.19	12.03	12.23	1.81	1.83	2.82	2.95
2nd	256 × 256	4	9.32	11.16	5.39	7.40	2.08	2.12	1.51	1.75
2nd	256 × 256	8	6.58	8.00	2.76	4.32	2.29	2.62	0.89	1.18
3rd	256 × 256	1	26.92	26.92	26.92	26.92	0.00	0.00	0.00	0.00
3rd	256 × 256	2	17.93	19.65	13.31	15.03	1.58	1.63	2.74	2.87
3rd	256 × 256	4	9.83	13.03	6.19	9.31	1.97	2.01	1.47	1.65
3rd	256 × 256	8	6.03	8.73	2.99	5.27	2.03	2.32	0.84	1.11
1st	512 × 512	1	115.39	115.39	115.39	115.39	0.00	0.00	0.00	0.00
1st	512 × 512	2	66.85	66.87	59.34	59.56	2.80	2.91	4.11	4.21
1st	512 × 512	4	33.95	39.18	28.43	33.82	3.02	3.10	2.10	2.35
1st	512 × 512	8	18.55	24.30	13.96	19.62	3.11	3.21	1.13	1.41
2nd	512 × 512	1	65.01	65.01	65.01	65.01	0.00	0.00	0.00	0.00
2nd	512 × 512	2	38.52	40.78	32.37	34.56	2.34	2.40	3.43	3.56
2nd	512 × 512	4	18.18	25.73	13.35	20.86	2.63	2.71	1.88	2.07
2nd	512 × 512	8	11.06	16.46	6.77	12.22	2.78	3.11	1.18	1.61
3rd	512 × 512	1	81.76	81.76	81.76	81.76	0.00	0.00	0.00	0.00
3rd	512 × 512	2	46.82	48.62	40.90	42.78	2.13	2.20	3.37	3.51
3rd	512 × 512	4	20.97	31.82	16.54	27.22	2.49	2.57	1.77	1.95
3rd	512 × 512	8	11.83	18.42	8.26	14.54	2.43	2.68	0.96	1.25
1st	1024 × 1024	1	353.49	353.49	353.49	353.49	0.00	0.00	0.00	0.00
1st	1024 × 1024	2	183.37	191.85	175.50	183.57	3.07	3.16	4.51	4.51
1st	1024 × 1024	4	89.25	104.63	83.59	98.59	3.18	3.26	2.33	2.34
1st	1024 × 1024	8	44.39	64.05	39.47	59.27	3.16	3.26	1.24	1.66
2nd	1024 × 1024	1	202.79	202.79	202.79	202.79	0.00	0.00	0.00	0.00
2nd	1024 × 1024	2	106.86	113.74	99.76	106.47	2.76	2.84	3.94	4.06
2nd	1024 × 1024	4	47.60	69.85	42.14	64.12	2.99	3.10	2.26	2.51
2nd	1024 × 1024	8	26.15	39.82	21.23	35.22	3.04	3.31	1.26	1.60
3rd	1024 × 1024	1	257.06	257.06	257.06	257.06	0.00	0.00	0.00	0.00
3rd	1024 × 1024	2	136.71	139.09	129.62	132.27	2.57	2.66	3.97	4.01
3rd	1024 × 1024	4	55.12	89.98	50.07	84.73	2.86	2.94	2.01	2.21
3rd	1024 × 1024	8	29.39	47.79	25.38	43.38	2.72	2.94	1.09	1.46

Table A.4. Rectangular division results using bounding box with blunt data.

RESULTS IN SECONDS			POST DATA SET							
EXECUTION TIMES			TOTAL		RENDERING		SWAPPING		DIVIDING	
VIEW	RESOLUTION	P	MIN	MAX	MIN	MAX	MIN	MAX	MIN	MAX
1st	256 × 256	1	61.40	61.40	61.40	61.40	0.00	0.00	0.00	0.00
1st	256 × 256	2	42.44	43.61	31.37	32.01	3.42	3.45	6.46	8.29
1st	256 × 256	4	23.34	23.58	16.14	16.37	3.54	3.56	3.22	3.41
1st	256 × 256	8	14.01	14.95	8.26	8.98	3.78	3.83	1.85	2.18
2nd	256 × 256	1	31.01	31.01	31.01	31.01	0.00	0.00	0.00	0.00
2nd	256 × 256	2	26.76	27.06	15.81	16.86	3.20	3.20	6.22	7.61
2nd	256 × 256	4	16.23	16.94	8.51	9.24	3.87	3.94	3.45	3.58
2nd	256 × 256	8	10.21	11.38	4.46	5.20	3.75	4.08	1.83	2.19
3rd	256 × 256	1	22.53	22.53	22.53	22.53	0.00	0.00	0.00	0.00
3rd	256 × 256	2	21.73	22.74	11.52	12.43	2.85	2.87	6.01	7.94
3rd	256 × 256	4	13.22	13.95	6.03	6.63	3.53	3.62	3.42	3.61
3rd	256 × 256	8	8.58	10.13	3.20	4.19	3.52	3.82	1.73	2.03
1st	512 × 512	1	175.11	175.11	175.11	175.11	0.00	0.00	0.00	0.00
1st	512 × 512	2	101.93	106.55	88.77	89.08	4.28	8.81	7.97	8.13
1st	512 × 512	4	54.31	56.24	44.84	45.79	4.89	4.95	4.05	5.20
1st	512 × 512	8	29.70	32.82	22.14	24.78	4.78	4.96	2.20	3.21
2nd	512 × 512	1	89.68	89.68	89.68	89.68	0.00	0.00	0.00	0.00
2nd	512 × 512	2	59.74	60.34	45.33	46.88	4.76	4.77	8.03	8.99
2nd	512 × 512	4	32.79	37.22	22.19	26.40	5.83	5.93	4.48	4.70
2nd	512 × 512	8	19.24	23.14	11.09	14.55	5.62	5.99	2.38	2.73
3rd	512 × 512	1	61.91	61.91	61.91	61.91	0.00	0.00	0.00	0.00
3rd	512 × 512	2	45.53	45.70	31.08	33.18	4.22	4.27	7.70	9.60
3rd	512 × 512	4	25.85	27.61	15.77	17.29	5.34	5.36	4.43	4.67
3rd	512 × 512	8	15.49	18.59	7.71	10.30	5.27	5.61	2.30	2.57
1st	1024 × 1024	1	546.84	546.84	546.84	546.84	0.00	0.00	0.00	0.00
1st	1024 × 1024	2	293.99	294.53	274.65	277.05	6.47	6.47	10.07	11.94
1st	1024 × 1024	4	151.95	155.95	139.08	143.12	6.92	6.97	5.40	5.49
1st	1024 × 1024	8	76.25	87.23	66.68	77.23	6.57	6.85	2.81	3.26
2nd	1024 × 1024	1	261.28	261.28	261.28	261.28	0.00	0.00	0.00	0.00
2nd	1024 × 1024	2	149.98	152.97	132.64	133.57	6.45	6.46	10.05	12.02
2nd	1024 × 1024	4	75.24	93.19	61.44	79.23	7.69	7.75	5.68	5.84
2nd	1024 × 1024	8	38.90	54.81	28.43	44.00	7.31	7.74	2.90	3.18
3rd	1024 × 1024	1	184.77	184.77	184.77	184.77	0.00	0.00	0.00	0.00
3rd	1024 × 1024	2	110.52	112.69	92.28	96.51	5.88	5.92	9.54	11.47
3rd	1024 × 1024	4	58.53	63.24	45.16	49.85	7.20	7.26	5.75	5.76
3rd	1024 × 1024	8	33.05	39.73	21.49	29.00	6.80	8.75	2.86	3.22

Table A.5. Rectangular division results using bounding box with post data.

RESULTS IN SECONDS			DELTA DATA SET							
EXECUTION TIMES			TOTAL		RENDERING		SWAPPING		DIVIDING	
VIEW	RESOLUTION	P	MIN	MAX	MIN	MAX	MIN	MAX	MIN	MAX
1st	256 × 256	1	75.72	75.72	75.72	75.72	0.00	0.00	0.00	0.00
1st	256 × 256	2	59.70	87.08	34.81	42.33	5.15	39.80	10.56	11.76
1st	256 × 256	4	29.81	38.53	14.86	21.79	7.71	7.75	6.42	8.50
1st	256 × 256	8	25.19	30.56	7.26	13.54	11.57	12.65	4.57	5.20
2nd	256 × 256	1	60.89	60.39	60.89	60.89	0.00	0.00	0.00	0.00
2nd	256 × 256	2	51.01	52.74	31.11	33.24	7.81	8.81	9.98	10.90
2nd	256 × 256	4	28.79	31.76	15.70	16.76	6.60	6.67	6.04	8.23
2nd	256 × 256	8	23.74	25.64	7.91	9.53	10.27	11.58	4.36	4.94
3rd	256 × 256	1	35.52	35.52	35.52	35.52	0.00	0.00	0.00	0.00
3rd	256 × 256	2	35.38	38.25	18.32	18.97	4.35	9.64	9.50	11.11
3rd	256 × 256	4	21.22	25.09	8.57	10.52	6.69	6.74	5.63	7.80
3rd	256 × 256	8	19.04	21.70	3.96	5.86	10.16	11.02	4.08	4.74
1st	512 × 512	1	219.86	219.86	219.86	219.86	0.00	0.00	0.00	0.00
1st	512 × 512	2	150.45	162.29	93.64	127.42	7.00	52.92	13.67	15.08
1st	512 × 512	4	63.60	90.65	36.60	69.58	9.26	17.94	8.31	10.68
1st	512 × 512	8	38.66	57.80	17.06	36.73	14.00	15.20	4.84	6.34
2nd	512 × 512	1	177.38	177.38	177.38	177.38	0.00	0.00	0.00	0.00
2nd	512 × 512	2	114.36	147.38	88.88	93.26	6.27	45.75	12.15	13.15
2nd	512 × 512	4	62.08	66.39	45.05	48.04	8.95	9.09	7.57	8.97
2nd	512 × 512	8	41.94	46.50	21.55	27.02	12.91	14.54	5.14	5.79
3rd	512 × 512	1	96.82	96.82	96.82	96.82	0.00	0.00	0.00	0.00
3rd	512 × 512	2	71.48	102.75	47.13	52.06	5.57	43.27	11.49	12.50
3rd	512 × 512	4	37.19	48.57	20.15	29.23	9.15	9.23	7.54	9.59
3rd	512 × 512	8	28.40	35.50	9.18	16.06	12.99	14.02	5.11	5.74
1st	1024 × 1024	1	650.86	650.86	650.86	650.86	0.00	0.00	0.00	0.00
1st	1024 × 1024	2	368.90	413.27	273.29	382.10	10.12	76.43	17.19	18.33
1st	1024 × 1024	4	132.77	244.78	106.72	219.21	12.34	15.30	9.42	11.61
1st	1024 × 1024	8	74.20	141.11	51.23	117.69	15.13	16.03	5.26	7.13
2nd	1024 × 1024	1	532.26	532.26	532.26	532.26	0.00	0.00	0.00	0.00
2nd	1024 × 1024	2	316.65	324.57	249.99	290.99	7.83	59.60	14.39	15.52
2nd	1024 × 1024	4	149.54	173.00	128.02	149.24	11.55	11.65	9.61	12.01
2nd	1024 × 1024	8	82.42	99.08	60.21	77.80	14.18	15.99	5.35	6.21
3rd	1024 × 1024	1	285.92	285.92	285.92	285.92	0.00	0.00	0.00	0.00
3rd	1024 × 1024	2	175.93	198.45	138.07	152.71	6.95	46.05	13.40	14.33
3rd	1024 × 1024	4	78.91	112.69	57.82	88.98	11.34	11.51	9.27	11.42
3rd	1024 × 1024	8	48.59	69.48	27.81	48.29	14.33	15.42	5.30	6.27

Table A.6. Rectangular division results using bounding box with delta data.

RESULTS IN SECONDS			BLUNT DATA SET							
EXECUTION TIMES			TOTAL		RENDERING		SWAPPING		DIVIDING	
VIEW	RESOLUTION	P	MIN	MAX	MIN	MAX	MIN	MAX	MIN	MAX
1st	256 × 256	1	40.79	40.79	40.79	40.79	0.00	0.00	0.00	0.00
1st	256 × 256	2	27.39	28.12	21.84	22.77	2.64	2.67	5.19	5.22
1st	256 × 256	4	15.47	19.08	10.69	12.63	1.97	3.52	4.11	6.51
1st	256 × 256	8	8.55	15.29	5.18	7.25	1.45	4.44	3.29	8.07
2nd	256 × 256	1	22.50	22.50	22.50	22.50	0.00	0.00	0.00	0.00
2nd	256 × 256	2	14.55	17.12	10.54	13.04	1.79	1.80	3.86	3.93
2nd	256 × 256	4	8.13	12.39	4.90	7.44	1.20	2.44	3.20	4.82
2nd	256 × 256	8	5.37	9.31	2.71	4.29	0.94	2.49	2.64	4.96
3rd	256 × 256	1	26.92	26.92	26.92	26.92	0.00	0.00	0.00	0.00
3rd	256 × 256	2	17.38	18.86	13.30	15.02	1.65	1.71	3.79	3.89
3rd	256 × 256	4	8.91	12.22	6.15	9.53	0.97	2.19	2.65	4.28
3rd	256 × 256	8	6.28	9.17	3.27	5.04	0.83	2.23	2.33	4.67
1st	512 × 512	1	115.39	115.39	115.39	115.39	0.00	0.00	0.00	0.00
1st	512 × 512	2	65.34	65.67	59.20	59.80	2.95	2.96	5.71	5.75
1st	512 × 512	4	32.34	38.63	28.01	33.81	2.20	4.07	4.14	7.55
1st	512 × 512	8	16.26	24.04	12.88	18.05	1.58	5.07	3.35	9.65
2nd	512 × 512	1	65.01	65.01	65.01	65.01	0.00	0.00	0.00	0.00
2nd	512 × 512	2	31.36	44.42	26.53	39.37	2.35	2.35	4.72	4.76
2nd	512 × 512	4	16.48	29.58	12.68	22.84	1.38	3.53	3.63	6.55
2nd	512 × 512	8	9.25	19.55	6.46	12.01	1.11	3.59	2.75	7.65
3rd	512 × 512	1	81.76	81.76	81.76	81.76	0.00	0.00	0.00	0.00
3rd	512 × 512	2	45.73	47.06	40.82	42.31	2.22	2.22	4.61	4.69
3rd	512 × 512	4	25.69	29.37	20.24	22.98	1.33	3.20	3.35	6.21
3rd	512 × 512	8	14.19	18.10	10.22	12.26	1.16	3.03	2.38	6.29
1st	1024 × 1024	1	353.49	353.49	353.49	353.49	0.00	0.00	0.00	0.00
1st	1024 × 1024	2	183.20	191.00	176.70	184.10	3.27	3.29	6.41	6.42
1st	1024 × 1024	4	87.43	107.20	83.04	98.32	2.28	4.60	4.33	8.52
1st	1024 × 1024	8	40.21	66.55	36.67	59.58	1.51	6.17	3.52	12.11
2nd	1024 × 1024	1	202.79	202.79	202.79	202.79	0.00	0.00	0.00	0.00
2nd	1024 × 1024	2	90.17	127.70	84.25	121.50	3.05	3.08	5.72	5.85
2nd	1024 × 1024	4	44.63	74.83	40.59	66.91	1.74	4.03	3.98	7.70
2nd	1024 × 1024	8	23.66	46.55	20.67	40.42	1.30	4.71	2.93	9.13
3rd	1024 × 1024	1	257.06	257.06	257.06	257.06	0.00	0.00	0.00	0.00
3rd	1024 × 1024	2	103.50	167.10	97.96	161.20	2.73	2.74	5.47	5.50
3rd	1024 × 1024	4	54.49	90.48	50.14	83.37	1.89	3.75	4.27	7.29
3rd	1024 × 1024	8	30.06	60.61	25.14	50.45	1.57	5.07	3.76	9.89

Table A.7. Recursive division results using *bounding box* with *blunt* data.

RESULTS IN SECONDS			POST DATA SET							
EXECUTION TIMES			TOTAL		RENDERING		SWAPPING		DIVIDING	
VIEW	RESOLUTION	P	MIN	MAX	MIN	MAX	MIN	MAX	MIN	MAX
1st	256 × 256	1	61.40	61.40	61.40	61.40	0.00	0.00	0.00	0.00
1st	256 × 256	2	39.99	40.72	31.11	31.92	3.56	3.56	8.50	8.57
1st	256 × 256	4	24.02	25.21	16.01	17.10	3.60	3.84	7.82	8.09
1st	256 × 256	8	13.73	16.65	7.96	9.58	2.17	3.80	4.69	7.91
2nd	256 × 256	1	31.01	31.01	31.01	31.01	0.00	0.00	0.00	0.00
2nd	256 × 256	2	24.20	25.58	15.88	17.19	3.26	3.29	8.06	8.10
2nd	256 × 256	4	16.82	17.06	8.31	9.32	2.80	4.01	7.63	8.41
2nd	256 × 256	8	10.58	13.58	4.48	5.34	1.87	3.80	5.17	8.69
3rd	256 × 256	1	22.53	22.53	22.53	22.53	0.00	0.00	0.00	0.00
3rd	256 × 256	2	19.52	19.66	11.56	11.81	2.88	2.89	7.62	7.74
3rd	256 × 256	4	12.78	14.98	5.36	7.34	2.53	3.47	7.33	7.67
3rd	256 × 256	8	7.10	10.63	2.81	4.59	1.50	2.67	3.99	6.48
1st	512 × 512	1	175.11	175.11	175.11	175.11	0.00	0.00	0.00	0.00
1st	512 × 512	2	100.40	101.20	89.04	89.35	4.78	5.73	10.62	11.67
1st	512 × 512	4	55.43	57.02	45.29	46.10	4.64	5.28	9.83	10.69
1st	512 × 512	8	29.02	35.57	22.03	25.54	3.21	5.08	6.58	10.89
2nd	512 × 512	1	89.68	89.68	89.68	89.68	0.00	0.00	0.00	0.00
2nd	512 × 512	2	56.04	60.45	45.06	49.37	4.82	4.84	10.58	10.62
2nd	512 × 512	4	34.91	38.03	22.34	26.60	4.53	6.42	11.27	12.32
2nd	512 × 512	8	21.41	27.17	10.87	14.25	3.19	6.44	7.38	13.67
3rd	512 × 512	1	61.91	61.91	61.91	61.91	0.00	0.00	0.00	0.00
3rd	512 × 512	2	41.05	43.33	30.65	32.85	4.35	4.37	10.06	10.12
3rd	512 × 512	4	22.88	32.92	12.28	21.71	3.96	5.39	10.52	10.93
3rd	512 × 512	8	12.93	22.05	6.05	11.67	2.01	4.95	5.14	10.92
1st	1024 × 1024	1	546.84	546.84	546.84	546.84	0.00	0.00	0.00	0.00
1st	1024 × 1024	2	289.50	289.60	275.40	275.50	6.56	6.62	13.46	13.53
1st	1024 × 1024	4	137.00	176.20	123.10	161.90	6.46	7.29	13.78	14.18
1st	1024 × 1024	8	70.21	101.40	61.84	81.75	3.50	9.06	7.30	19.37
2nd	1024 × 1024	1	261.28	261.28	261.28	261.28	0.00	0.00	0.00	0.00
2nd	1024 × 1024	2	137.60	162.20	123.60	148.20	6.55	6.57	13.36	13.40
2nd	1024 × 1024	4	76.27	94.26	60.51	78.80	6.21	8.12	14.86	15.64
2nd	1024 × 1024	8	41.16	61.15	29.69	42.95	3.57	9.83	10.24	20.93
3rd	1024 × 1024	1	184.77	184.77	184.77	184.77	0.00	0.00	0.00	0.00
3rd	1024 × 1024	2	104.90	109.50	91.67	96.38	6.11	6.14	12.62	12.62
3rd	1024 × 1024	4	58.99	64.14	44.66	49.49	5.71	7.52	14.09	14.49
3rd	1024 × 1024	8	30.27	46.00	17.92	33.50	3.51	7.52	10.15	19.13

Table A.8. Recursive division results using *bounding box* with *post data*.

RESULTS IN SECONDS			DELTA DATA SET							
EXECUTION TIMES			TOTAL		RENDERING		SWAPPING		DIVIDING	
VIEW	RESOLUTION	P	MIN	MAX	MIN	MAX	MIN	MAX	MIN	MAX
1st	256 × 256	1	75.72	75.72	75.72	75.72	0.00	0.00	0.00	0.00
1st	256 × 256	2	56.86	65.51	36.56	41.30	10.91	14.96	19.97	23.57
1st	256 × 256	4	28.83	40.62	15.87	22.91	4.22	7.93	12.89	17.42
1st	256 × 256	8	18.79	29.86	7.61	12.81	2.78	9.01	10.89	18.43
2nd	256 × 256	1	60.89	60.89	60.89	60.89	0.00	0.00	0.00	0.00
2nd	256 × 256	2	46.91	62.76	31.07	32.00	6.23	22.48	14.54	31.33
2nd	256 × 256	4	29.91	31.25	15.82	16.65	4.01	6.56	13.84	14.37
2nd	256 × 256	8	15.71	25.16	8.19	9.74	2.57	7.01	7.47	15.79
3rd	256 × 256	1	35.52	35.52	35.52	35.52	0.00	0.00	0.00	0.00
3rd	256 × 256	2	31.63	40.59	18.23	18.83	4.62	13.73	12.40	22.04
3rd	256 × 256	4	21.88	24.01	8.51	10.37	4.11	6.15	13.24	13.61
3rd	256 × 256	8	14.26	20.03	3.89	6.09	2.48	6.94	8.10	14.11
1st	512 × 512	1	219.86	219.86	219.86	219.86	0.00	0.00	0.00	0.00
1st	512 × 512	2	118.50	158.50	89.72	133.90	13.35	17.43	23.48	28.40
1st	512 × 512	4	55.31	96.89	37.44	69.79	6.88	10.42	17.82	26.20
1st	512 × 512	8	28.30	63.67	16.52	39.01	3.23	13.01	11.46	31.54
2nd	512 × 512	1	177.38	177.38	177.38	177.38	0.00	0.00	0.00	0.00
2nd	512 × 512	2	111.90	124.90	89.35	93.30	8.15	25.40	17.76	35.34
2nd	512 × 512	4	56.43	73.19	42.37	49.82	4.37	11.22	13.97	23.37
2nd	512 × 512	8	32.66	52.60	20.17	26.98	2.91	11.27	12.29	27.90
3rd	512 × 512	1	96.82	96.82	96.82	96.82	0.00	0.00	0.00	0.00
3rd	512 × 512	2	62.35	77.11	42.85	56.14	10.49	11.18	19.16	20.28
3rd	512 × 512	4	34.32	52.91	18.36	33.05	4.95	9.72	15.64	19.48
3rd	512 × 512	8	16.91	38.82	8.71	17.81	3.21	10.00	8.18	20.75
1st	1024 × 1024	1	650.86	650.86	650.86	650.86	0.00	0.00	0.00	0.00
1st	1024 × 1024	2	343.60	407.70	272.00	380.20	13.11	58.33	25.63	71.29
1st	1024 × 1024	4	140.80	254.20	122.90	219.90	7.57	17.85	17.59	33.81
1st	1024 × 1024	8	68.49	165.20	55.11	136.00	4.17	20.48	13.24	44.31
2nd	1024 × 1024	1	532.26	532.26	532.26	532.26	0.00	0.00	0.00	0.00
2nd	1024 × 1024	2	290.70	311.40	250.10	289.90	9.34	29.08	20.27	40.36
2nd	1024 × 1024	4	133.40	178.80	116.20	147.20	4.55	14.74	17.13	31.30
2nd	1024 × 1024	8	69.83	113.60	55.35	94.39	2.98	14.10	14.37	30.05
3rd	1024 × 1024	1	285.92	285.92	285.92	285.92	0.00	0.00	0.00	0.00
3rd	1024 × 1024	2	174.20	176.70	137.80	152.00	11.54	28.09	21.21	38.38
3rd	1024 × 1024	4	78.08	115.20	56.84	88.68	6.72	11.75	21.14	25.98
3rd	1024 × 1024	8	35.08	79.85	26.30	54.46	2.81	12.46	8.75	24.86

Table A.9. Recursive division results using *bounding box* with *delta* data.

RESULTS IN SECONDS			BLUNT DATA SET							
EXECUTION TIMES			TOTAL		RENDERING		SWAPPING		DIVIDING	
VIEW	RESOLUTION	P	MIN	MAX	MIN	MAX	MIN	MAX	MIN	MAX
1st	256 × 256	1	40.79	40.79	40.79	40.79	0.00	0.00	0.00	0.00
1st	256 × 256	2	30.12	33.08	20.24	22.86	3.36	3.45	9.55	10.11
1st	256 × 256	4	21.17	22.96	10.82	12.56	3.20	4.68	10.00	10.99
1st	256 × 256	8	13.61	24.86	5.94	7.14	2.18	7.96	7.00	18.23
1st	256 × 256	16	9.29	42.08	3.31	4.38	1.28	15.72	5.09	38.02
1st	256 × 256	32	6.67	277.80	2.10	2.95	0.82	267.30	4.25	275.30
2nd	256 × 256	1	22.50	22.50	22.50	22.50	0.00	0.00	0.00	0.00
2nd	256 × 256	2	18.55	24.08	12.25	12.29	2.78	5.65	6.11	11.69
2nd	256 × 256	4	12.51	20.36	6.09	7.01	2.16	3.85	6.29	13.40
2nd	256 × 256	8	9.24	19.10	3.17	4.40	1.57	5.16	4.78	15.05
2nd	256 × 256	16	6.07	12.04	1.81	2.97	1.03	3.12	3.70	9.36
2nd	256 × 256	32	28.63	242.20	0.35	2.93	1.08	38.61	26.68	240.30
3rd	256 × 256	1	26.92	26.92	26.92	26.92	0.00	0.00	0.00	0.00
3rd	256 × 256	2	24.08	88.01	12.95	15.72	2.62	11.90	8.32	74.85
3rd	256 × 256	4	14.84	18.83	6.80	9.08	1.75	3.00	6.41	10.76
3rd	256 × 256	8	12.07	1426.00	3.72	5.47	1.17	667.20	8.28	1422.00
3rd	256 × 256	16	6.03	10.67	1.98	3.13	0.71	2.66	3.45	8.02
3rd	256 × 256	32	3.69	6.88	1.27	2.10	0.60	1.78	2.06	5.22
1st	512 × 512	1	115.39	115.39	115.39	115.39	0.00	0.00	0.00	0.00
1st	512 × 512	2	74.39	74.74	59.27	59.62	4.23	4.35	14.66	15.02
1st	512 × 512	4	44.07	52.03	30.01	31.62	3.20	6.32	13.84	20.62
1st	512 × 512	8	26.17	58.66	15.25	17.71	2.22	10.66	9.42	40.84
1st	512 × 512	16	14.91	28.19	7.08	10.06	1.23	5.24	6.14	19.43
1st	512 × 512	32	8.73	18.83	3.87	5.85	0.79	4.22	4.06	13.21
2nd	512 × 512	1	65.01	65.01	65.01	65.01	0.00	0.00	0.00	0.00
2nd	512 × 512	2	42.13	43.62	33.66	34.42	2.55	3.32	7.40	9.82
2nd	512 × 512	4	26.03	31.54	17.40	18.51	2.80	4.79	8.01	12.91
2nd	512 × 512	8	17.49	35.19	8.99	10.24	2.12	6.32	7.20	25.09
2nd	512 × 512	16	10.85	19.56	4.86	6.63	1.21	4.43	4.80	14.06
2nd	512 × 512	32	8.12	15.56	2.65	4.20	1.08	5.96	4.41	11.36
3rd	512 × 512	1	81.76	81.76	81.76	81.76	0.00	0.00	0.00	0.00
3rd	512 × 512	2	52.95	56.19	40.08	43.66	3.05	3.12	12.45	12.58
3rd	512 × 512	4	31.82	37.17	20.75	23.44	2.22	4.31	10.90	15.60
3rd	512 × 512	8	19.16	40.27	10.68	13.80	1.51	7.25	7.39	28.78
3rd	512 × 512	16	11.86	20.40	5.58	7.36	0.94	3.63	4.86	14.53
3rd	512 × 512	32	7.55	17.12	3.24	4.25	0.76	9.10	3.82	13.51
1st	1024 × 1024	1	353.49	353.49	353.49	353.49	0.00	0.00	0.00	0.00
1st	1024 × 1024	2	200.40	218.40	176.60	185.50	5.52	6.17	23.72	32.46
1st	1024 × 1024	4	107.70	123.90	88.91	97.85	3.29	6.00	18.70	29.37
1st	1024 × 1024	8	58.44	99.00	43.25	51.94	2.45	12.03	15.16	54.63
1st	1024 × 1024	16	31.11	51.16	20.38	28.31	1.09	5.94	9.87	28.43
1st	1024 × 1024	32	71.40	534.90	10.09	15.98	0.98	16.97	56.96	524.80
2nd	1024 × 1024	1	202.79	202.79	202.79	202.79	0.00	0.00	0.00	0.00
2nd	1024 × 1024	2	115.50	135.90	100.20	106.80	4.55	7.88	15.25	28.73
2nd	1024 × 1024	4	65.70	75.40	50.11	57.24	3.46	5.43	15.53	21.85
2nd	1024 × 1024	8	41.75	93.73	25.71	29.70	1.76	10.89	16.00	65.84
2nd	1024 × 1024	16	23.46	37.85	12.39	16.75	1.33	6.76	7.81	22.66
2nd	1024 × 1024	32	16.37	26.18	6.77	10.43	1.14	5.62	7.16	16.62
3rd	1024 × 1024	1	257.06	257.06	257.06	257.06	0.00	0.00	0.00	0.00
3rd	1024 × 1024	2	155.90	158.60	125.90	134.20	3.58	5.48	21.61	32.36
3rd	1024 × 1024	4	88.11	97.61	64.96	70.16	2.66	6.33	18.31	31.24
3rd	1024 × 1024	8	50.81	60.91	31.86	38.12	1.76	6.18	14.52	29.02
3rd	1024 × 1024	16	29.48	34.76	16.39	19.74	1.08	3.80	10.21	17.97
3rd	1024 × 1024	32	18.87	107.60	9.35	11.36	0.90	4.96	7.75	97.63

Table A.10. Horizontal division results using rasterization with blunt data.

RESULTS IN SECONDS			BLUNT DATA SET							
EXECUTION TIMES			TOTAL		RENDERING		SWAPPING		DIVIDING	
VIEW	RESOLUTION	P	MIN	MAX	MIN	MAX	MIN	MAX	MIN	MAX
1st	256 × 256	1	40.79	40.79	40.79	40.79	0.00	0.00	0.00	0.00
1st	256 × 256	2	30.11	33.73	20.22	23.22	2.96	3.79	9.58	10.39
1st	256 × 256	4	19.78	23.47	10.43	12.52	2.95	4.74	9.17	12.44
1st	256 × 256	8	13.60	23.94	5.51	7.34	2.21	9.13	7.61	18.39
1st	256 × 256	16	7.25	17.95	3.00	4.40	1.14	4.23	4.23	14.31
1st	256 × 256	32	10.56	144.00	1.52	2.78	0.90	8.83	8.33	142.20
2nd	256 × 256	1	22.50	22.50	22.50	22.50	0.00	0.00	0.00	0.00
2nd	256 × 256	2	19.73	20.33	10.13	13.80	2.39	4.00	5.72	10.14
2nd	256 × 256	4	11.18	15.98	5.30	7.91	1.98	3.05	5.34	10.29
2nd	256 × 256	8	9.74	80.77	2.83	4.76	1.20	6.01	4.93	76.67
2nd	256 × 256	16	56.42	257.10	1.59	2.86	0.56	6.50	54.68	255.20
2nd	256 × 256	32	3.09	8.97	0.96	2.04	0.43	2.16	1.81	7.60
3rd	256 × 256	1	26.92	26.92	26.92	26.92	0.00	0.00	0.00	0.00
3rd	256 × 256	2	19.48	25.71	13.47	15.31	2.13	4.01	5.82	10.35
3rd	256 × 256	4	14.40	20.90	6.60	9.12	1.90	2.73	6.56	12.87
3rd	256 × 256	8	8.39	18.53	3.49	5.79	1.06	4.29	4.67	12.72
3rd	256 × 256	16	5.96	61.62	1.86	3.65	0.67	2.52	3.95	58.87
3rd	256 × 256	32	7.60	320.80	1.01	3.04	0.39	150.00	6.59	318.70
1st	512 × 512	1	115.39	115.39	115.39	115.39	0.00	0.00	0.00	0.00
1st	512 × 512	2	75.11	75.57	59.71	60.92	4.09	4.18	14.53	14.95
1st	512 × 512	4	43.88	51.58	29.81	31.69	3.31	6.35	13.67	20.32
1st	512 × 512	8	26.92	54.02	13.45	20.01	2.19	10.68	9.43	39.60
1st	512 × 512	16	11.97	26.80	6.35	10.81	1.29	4.59	5.61	17.92
1st	512 × 512	32	7.84	20.80	3.41	6.24	0.67	7.28	3.88	15.70
2nd	512 × 512	1	65.01	65.01	65.01	65.01	0.00	0.00	0.00	0.00
2nd	512 × 512	2	35.30	50.48	25.74	40.40	3.27	3.32	9.51	9.74
2nd	512 × 512	4	21.48	36.00	13.54	21.50	2.41	5.07	7.92	14.40
2nd	512 × 512	8	12.19	38.50	6.83	11.50	1.37	6.34	5.32	26.96
2nd	512 × 512	16	7.30	26.65	3.67	6.91	0.76	5.81	3.04	19.68
2nd	512 × 512	32	4.81	15.77	2.15	4.19	0.50	3.10	2.24	12.33
3rd	512 × 512	1	81.76	81.76	81.76	81.76	0.00	0.00	0.00	0.00
3rd	512 × 512	2	54.23	56.34	40.97	43.57	3.14	3.20	12.71	12.95
3rd	512 × 512	4	30.70	36.12	20.33	23.42	2.08	4.25	10.20	14.60
3rd	512 × 512	8	18.59	40.10	8.64	15.44	1.45	7.30	7.25	28.59
3rd	512 × 512	16	8.72	20.09	4.24	9.22	0.72	4.11	4.08	15.26
3rd	512 × 512	32	5.87	15.62	2.48	5.37	0.46	3.14	3.35	11.16
1st	1024 × 1024	1	353.49	353.49	353.49	353.49	0.00	0.00	0.00	0.00
1st	1024 × 1024	2	201.50	236.60	178.40	185.60	5.18	5.87	22.99	50.52
1st	1024 × 1024	4	108.10	132.40	89.25	97.03	3.55	5.98	18.84	39.01
1st	1024 × 1024	8	52.80	99.94	39.00	60.36	2.11	11.74	13.77	55.43
1st	1024 × 1024	16	44.00	135.20	18.97	33.84	1.24	27.30	14.57	106.50
1st	1024 × 1024	32	63.62	675.40	10.02	19.33	0.89	161.70	51.77	660.40
2nd	1024 × 1024	1	202.79	202.79	202.79	202.79	0.00	0.00	0.00	0.00
2nd	1024 × 1024	2	98.23	145.70	83.73	123.10	3.85	6.01	14.43	22.20
2nd	1024 × 1024	4	53.95	86.24	42.32	63.94	2.96	6.42	11.61	23.05
2nd	1024 × 1024	8	53.24	311.50	21.46	35.84	1.43	226.90	18.76	280.20
2nd	1024 × 1024	16	15.16	45.14	11.48	21.05	0.82	7.04	3.37	28.66
2nd	1024 × 1024	32	10.34	147.50	6.55	12.26	0.56	44.95	3.45	136.70
3rd	1024 × 1024	1	257.06	257.06	257.06	257.06	0.00	0.00	0.00	0.00
3rd	1024 × 1024	2	156.70	158.50	126.20	135.00	3.76	5.64	21.55	31.95
3rd	1024 × 1024	4	86.30	124.30	64.99	69.90	2.60	6.41	21.09	57.82
3rd	1024 × 1024	8	50.82	62.84	31.41	38.28	1.98	5.74	14.83	31.39
3rd	1024 × 1024	16	24.69	40.27	14.05	24.34	1.01	3.98	10.00	20.19
3rd	1024 × 1024	32	22.98	101.10	7.25	13.77	0.57	48.18	11.49	93.07

Table A.11. Recursive division results using rasterization with blunt data.

Bibliography

- [1] Challinger, J. Parallel Volume Rendering for Curvilinear Volumes. In Proceedings of the Scalable High Performance Computing Conference (1992), IEEE Computer Society Press, pp. 14-21.
- [2] Challinger, J. Scalable Parallel Volume Raycasting for Nonrectilinear Computational Grids. In Proceedings of the Parallel Rendering Symposium (1993), IEEE Computer Society Press, pp. 81-88.
- [3] Committee, 1993 Parallel Rendering Symposium Proceedings. ACM Press, (1993).
- [4] Corrie, B., and Mackerras, P. Parallel Volume Rendering and Data Coherence. In Proceedings of the Parallel Rendering Symposium (1993), IEEE Computer Society Press, pp. 23-26.
- [5] Dani, S., Tohline, J. E., Waggenspack, W. N., and Thompson, D. E. Parallel Rendering of Curvilinear Volume Data. *Computers and Graphics* 18, 3 (1994), pp. 363-372.
- [6] Elvins, T. T. Volume Rendering on a Distributed Memory Parallel Computer. In *Visualization '92* (1992), IEEE, pp. 93-98.
- [7] Elvins, T. T. A Survey of Algorithms for Volume Visualization. *Computer Graphics* 26, 3 (1992), pp. 194-201.
- [8] Garrity, M. P. Raytracing Irregular Volume Data. *Computer Graphics* 24, 5 (1990), pp. 35-40. In Proceedings of San Diego Workshop on Volume Visualization.
- [9] Giertsen, C. Volume Visualization of Sparse Irregular Meshes. *IEEE Computer Graphics and Applications* 12, 2 (1992), pp. 40-48.
- [10] Hearn, D., and Baker, M. P. *Computer Graphics*. Prentice-Hall International, (1986).

- [11] Hsu, W. M. Segmented Ray Casting For Data Parallel Volume Rendering. In Proceedings of the Parallel Rendering Symposium (1993), IEEE Computer Society Press, pp. 7-14.
- [12] İşler, V., Aykanat, C., and Özgüç, B. An Efficient Parallel Spatial Subdivision Algorithm for Parallel Ray Tracing Complex Scenes. In Proceedings of the 1st Bilkent Computer Graphics Conference on Advanced Techniques in Animation, Rendering, and Visualization (1993), IEEE, pp. 121-135.
- [13] Kaufman, A. E., Lorensen, W. E., and Yagel, R. Volume Visualization Algorithms and Applications. Visualization 1993, Tutorial 9, (1993).
- [14] Koyamada, K. Fast Traversal of Irregular Volumes. In Visual Computing - Integrated Computer Graphics and Computer Vision, T. L. Kunii, Ed. Springer Verlag, (1992), pp. 295-312.
- [15] Levoy, M. Display of Surfaces From Volume Data. IEEE Computer Graphics and Applications 8, 3 (1988), pp. 29-37.
- [16] Levoy, M. Design for a Real-Time High-Quality Volume Rendering Workstation. In Proceedings of the Chapel Hill Workshop on Volume Visualization (1989), Department of Computer Science, University of North Carolina at Chapel Hill, pp. 85-92.
- [17] Levoy, M. Efficient Ray Tracing of Volume Data. ACM Transactions on Graphics 9, 3 (1990), pp. 245-261.
- [18] Lorensen, W. E., and Cline, H. E. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. Computer Graphics 21, 4 (1987), pp. 163-169.
- [19] Ma, K., and Painter, J. S. Parallel Volume Visualization on Workstations. Computers and Graphics 17, 1 (1993), pp. 31-37.
- [20] Palamidese, P. Scientific Visualization (Advanced Software Techniques). Ellis Horwood Workshop Series, (1993).
- [21] Rogers, D. F. Procedural Elements for Computer Graphics. McGraw-Hill Book Company, (1985).
- [22] Sabella, P. A Rendering Algorithm for Visualizing 3D Scalar Fields. Computer Graphics 22, 4 (1988), pp. 51-58. Proceedings of SIGGRAPH'88.

- [23] Shirley, P., and Tuchman, A. A Polygonal Approximation to Direct Scalar Volume Rendering. *Computer Graphics* 24, 5 (1990), pp. 63-70. Proceedings of San Diego Workshop on Volume Visualization.
- [24] Theoharis, T. *Algorithms for Parallel Polygon Rendering*. Springer-Verlag, (1989).
- [25] Upson, C., and Keeler, M. VBUFFER: Visible Volume Rendering. *Computer Graphics* 22, 4 (1988), pp. 59-64. Proceedings of SIGGRAPH '88.
- [26] Van Gelder, A., and Wilhelms, J. Rapid Exploration of Curvilinear Grids Using Direct Volume Rendering, pp. 70-77. In *Proceedings of Visualization '93* (1993), IEEE.
- [27] Watt, A. *Fundamentals of Three-Dimensional Computer Graphics*. Addison-Wesley Publishers, (1989).
- [28] Westover, L. Footprint Evaluation for Volume Rendering. *Computer Graphics* 24, 4 (1990), pp. 367-376. Proceedings of SIGGRAPH '90.
- [29] Wilhelms, J., Challinger, J., Alper, N., Ramamoorthy, S. and Vaziri, A. Direct Volume Rendering of Curvilinear Volumes. *Computer Graphics* 24, 5 (1990), pp. 41-47. Proceedings of San Diego Workshop on Volume Visualization.
- [30] Wilhelms, J. and Van Gelder, A. A Coherent Projection Approach for Direct Volume Rendering. *Computer Graphics* 25, 4 (1991), pp. 275-284. Proceedings of SIGGRAPH '91.
- [31] Williams, P. L. Visibility Ordering Meshed Polyhedra. *ACM Transactions on Graphics* 11, 2 (1992), pp. 103-126.