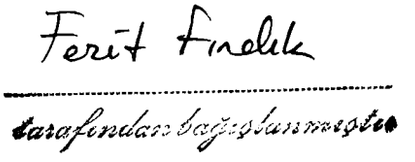
PARALLEL DIRECT VOLUME RENDERING OF UNSTRUCTURED GRIDS BASED ON OBJECT-SPACE DECOMPOSITION

A THESIS
SUBMITTED TO THE DEPARTMENT OF COMPUTER
ENGINEERING AND INFORMATION SCIENCE
AND THE INSTUTUTE OF ENGINEERING AND SCIENCE
OF BİLKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

By
Ferit Fındık
October, 1997

PARALLEL DIRECT VOLUME RENDERING OF UNSTRUCTURED GRIDS BASED ON OBJECT-SPACE DECOMPOSITION

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



By
Ferit Fındık
October, 1997

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(Principal 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. Atilla Gürsoy

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. Uğur Güdükbay

Approved for the Institute of Engineering and Science:

Prof. Dr. Mehmet Baray
Director of Institute of Engineering and Science

ABSTRACT

PARALLEL DIRECT VOLUME RENDERING
OF UNSTRUCTURED GRIDS
BASED ON OBJECT-SPACE DECOMPOSITION

Ferit Fındık
M.S. in Computer Engineering and Information Science
Supervisor: Assoc. Prof. Cevdet Aykanat
October, 1997

This work investigates object-space (OS) parallelization of an efficient ray-casting based direct volume rendering algorithm (DVR) for unstructured grids on distributed-memory architectures. The key point for a successful parallelization is to find an OS decomposition which maintains the OS coherency and computational load balance as much as possible. The OS decomposition problem is modeled as a graph partitioning (GP) problem with correct view-dependent node and edge weighting. As the parallel visualizations of the results of parallel engineering simulations are performed on the same machine, OS decomposition, which is necessary for each visualization instance because of the changes in the computational structures of the successive parallel steps, constitutes a typical case of the general remapping problem. A GP-based model is proposed for the solution of the general remapping problem by constructing an augmented remapping graph. The remapping tool RM-MeTiS, developed by modifying and enhancing the original MeTiS package for partitioning the remapping graph, is successfully used in the proposed parallel DVR algorithm. An effective view-independent cell-clustering scheme is introduced to induce more tractable contracted view-dependent remapping graphs for successive visualizations. An efficient estimation scheme with high accuracy is proposed for view-dependent node and edge weighting of the remapping graph. Speedup values as high as 22 are obtained on a Parsytec CC system with 24 processors in the visualization of benchmark volumetric datasets and the proposed DVR algorithm seems to be linearly scalable according to the experimental results.

Key words: Parallel Direct Volume Rendering, Unstructured Grids, Object-Space Decomposition, Graph Partitioning, Remapping, Scalability.

ÖZET

DÜZENSİZ IZGARALARIN OBJE UZAYI BÖLÜNMESİNE DAYANAN PARALEL HACİM GÖRÜNTÜLENMESİ

Ferit Fındık
Bilgisayar ve Enformatik Mühendisliği, Yüksek Lisans
Tez Yöneticisi: Doç. Dr. Cevdet Aykanat
Ekim, 1997

Bu çalışma düzensiz ızgaraların ışın izlemeye dayanan verimli doğrudan hacim görüntüleme (DHG) algoritmasının çok-işlemcili dağıtık bellekli bilgisayarlarda obje-uzayı (OU) paralelleştirilmesini araştırmaktadır. Başarılı bir paralelleştirmenin sırrı, OU benzerliğini koruyan ve mümkün olduğunca hesapsal yük dengesini sağlayan OU bölümünü bulmaktır. OU bölümü problemi çizge bölünmesi (ÇB) problemi olarak bakış açısına bağımlı doğru düğüm ve kenar ağırlığı verilmesiyle modellendi. Paralel mühendislik simulasyonlarının sonuçları aynı makinada paralel göröntülendiği için, ardışık paralel adımlarda oluşan hesapsal yapıdaki değişiklik, OU bölünmesini gerektirir ve bu bölünme genel yeniden eşleme problemine örnek teşkil eder. Genel yeniden eşleme probleminin çözümü için eklentili yeniden eşleme çizgesi oluşturarak, ÇB'ne dayalı bir model sunuldu. MeTiS ÇB aracını değiştirerek, yeniden eşleme aracı RM-MeTiS geliştirildi ve bu araç sunulan paralel DHG algoritmasında başarıyla kullanıldı. Ardışık görüntülemeler için bakış açısına bağımlı olmayan hücre gruplamasına gidilerek daha hızlı bölünebilen yeniden eşleme çizgesi oluşturuldu. Yeniden eşleme çizgesindeki düğüm ve kenarlarının ağırlık hesaplamaları için verimli ve hassas bir tahmin yöntemi geliştirildi. 24 işlemcili Parsytec CC sisteminde 22'ye varan hızlanma değerleri elde edildi. Deneysel sonuçlar, sunulan DHG algoritmasının doğrusal ölçeklenebilir olduğunu gösterdi.

Anahtar Kelimeler: Paralel Doğrudan Hacim Görüntüleme, Düzensiz Izgaralar, Obje Uzayı Bölünmesi, Çizge Bölünmesi, Yeniden Eşleme, Ölçeklenebilirlik.

ACKNOWLEDGMENTS

I would like to express my deep gratitude to my supervisor Dr. Cevdet Aykanat for his guidance, suggestions, and enjoyable discussions. I would like to thank Dr. Atilla Gürsoy and Dr. Uğur Güdükbay for reading and commenting on the thesis. I also owe special thanks to Dr. Tuğrul Dayar for his help. I would also like to thank my dear friend Hakan Berk and Tahsin Kurç for their help and suggestions on the coding. Finally, I am very grateful to my family and my friends for their support and patience.

Contents

1 Introduction **1**

2 Sequential DVR Algorithm **7**

2.1 Preliminaries 7

2.2 Major Data Structures 9

2.3 Koyamada's Algorithm 10

2.4 Execution Time Analysis 13

3 Issues in Object-Space Decomposition **15**

3.1 Graph Partitioning (GP) Problem 15

3.2 A GP-based OS Decomposition Model 16

3.3 Remapping Problem in OS Decomposition 18

4 A GP-based Remapping Model **20**

4.1 Problem Definition 20

4.2 Two-Phase Solution Model 21

4.3 One-Phase Solution Model 22

5 MeTiS-based Remapping Heuristic **24**

5.1 Graph Partitioning Heuristics 24

5.2 Modifying MeTiS Package for Remapping 25

5.2.1 Coarsening Phase 26

5.2.2 Initial Partitioning Phase 27

5.2.3 Uncoarsening Phase 29

6 Parallel DVR Algorithm **30**

6.1 Clustering Phase 30

6.2 Remapping Phase 33

6.2.1 Graph Update (GU) Step 34

6.2.2 Graph Partitioning (GP) Step 37

6.2.3 Cluster Migration (CM) Step 38

6.3 Local Rendering Phase 38

6.4 Pixel Merging Phase 40

6.4.1 Pixel Assignment (PA) Step 41

6.4.2 Ray-segment Migration (RSM) Step 44

6.4.3 Local Pixel Merging (LPM) Step 45

7 Experimental Results **46**

7.1 Clustering Phase 49

7.2 Remapping Phase 51

7.3 Pixel Merging Phase 54

7.4 Overall Performance Analysis 56

8 Conclusion **60**

List of Figures

2.1 Ray-casting based direct volume rendering. 8

2.2 Ray-face intersection and linear sampling scheme in Koyamada's algorithm. 12

6.1 Local rendering of shaded subvolume by processor P_1. 39

7.1 Rendered images of the volumetric datasets used in performance analysis. 48

List of Tables

7.1 List of volumetric datasets used for experimentation. 49

7.2 Variation of normalized average parallel rendering (T_{par}) and view-dependent preprocessing (T_{pre}) times with α_{avg}. 50

7.3 The clustering quality of various weighting schemes. 50

7.4 Estimated and measured percent load imbalance values, and percent errors in the estimation of ray-face intersection counts (I), sampling counts (S), local rendering times (T_{LR}). 52

7.5 Worst-case performance evaluation of RM-MeTiS compared to MeTiS. 52

7.6 Performance comparison of various OS decomposition schemes for K=24. 54

7.7 Relative performance comparison of pixel assigment (PA) schemes on the parallel pixel merging (PM) phase. 55

7.8 View-dependent clustering overhead as a percent of view-dependent parallel rendering time T_{par}, and percent dissection of T_{par}. 57

7.9 Average speedup (S_K) and efficiency (E_K) values. 59

Chapter 1

Introduction

The increasing complexity of scientific computations and engineering simulations necessitates more powerful tools for the interpretation of the acquired results. At this point, *scientific visualization* algorithms are utilized for the detailed interpretation of the resulting datasets to reach useful conclusions. *Volume rendering* is a very important branch of scientific visualization and makes it possible for scientists to visualize 3-Dimensional (3D) volumetric datasets.

Volumetric data used in volume rendering is in the form of grid superimposed on a volume. The vertices of this grid contain the scalar values that represent the simulation results. Type of the grid defines spatial characteristics of the volumetric dataset, which is important in the rendering process. Volumetric grids can be divided into two categories as: *structured* and *unstructured* [1]. In structured grids, the distribution of grid (sample) points in 3D-space exhibit a structure. As these grids preserve a regularity in the distribution of sample points, they can be represented by 3D arrays, therefore they are usually called *array oriented* grids. The mapping from the array elements to sample points and the connectivity relation between cells are implicit. On the other hand, in unstructured grids, the distribution of sample points do not follow a regular pattern, and there may be voids in the grid. Another term used for unstructured grids is *cell oriented* grids, because these grids are represented by a list of cells in which each cell contains pointers to sample points that form the cell. Due to the cell oriented nature and the irregularity of those grids,

the connectivity information must be provided explicitly. With the recent advances in generating higher quality adaptive meshes, unstructured grids are becoming increasingly popular in the simulation of scientific and engineering problems with complex geometries.

There are two major categories of volume rendering methods; *indirect* and *direct methods*. Indirect methods try to track, and extract intermediate geometrical representation of the data, and render those surfaces via conventional surface rendering methods. Direct methods render the data without generating an intermediate representation. However, these methods are slow due to massive computations performed, but give more accurate renderings. As the direct methods do not rely on the extraction of surfaces, they are more general and flexible.

Although volume rendering algorithms have become practical to use, there are still some problems to overcome. The most important of these problems is the speed of the rendering algorithms. Especially direct volume rendering (DVR) of unstructured grids, which is the major concern of this work, is still far from interactive response times. The slowness of the DVR process create the lack of interactivity which in turn prevents its wide use. This is one of the major reasons why there is a need for faster DVR algorithms through parallelization. In addition, it is very important for scientists to be able to change the simulation parameters so that the simulation is steered in the correct direction. Interactive visualization could help the scientists experiment with the parameters to select the useful ones.

Visualization of vast amount of volumetric dataset produced by scientific computations and engineering simulations requires large computer memory space. Hence, DVR is a good candidate for parallelization on distributed-memory multicomputers. In addition, most of the engineering simulations are done on multicomputers. Visualization of results on the same parallel machine, where simulations are done, avoids the extra overhead of transporting large amounts of data.

Existing DVR algorithms for unstructured grids can be classified into two categories; *object-space* and *image-space*. In object-space methods, the volume

is traversed in object-space to perform a view-dependent depth sort on the cells. Then, all cells are projected onto the screen, in this visibility order, to find their contributions on the image-plane and composite them. Image-space methods are also called *ray-casting* methods [2, 3, 4]. In these methods, for each pixel of the screen, a ray is cast and followed through the volume by intersecting it with the cells to find ray-face intersections. Samples are computed along the ray and they are composited to generate the color of the pixel. For non-convex datasets, the rays may enter and exit the volume more than once. The parts of the ray that lie inside the volume, which in fact determines the contributions of the dataset to the pixel, are referred to here as *ray-segments*. Ray-casting approach is a desirable choice for DVR because of its capability of rendering non-convex and cyclic grids, and its power of generating images of high quality.

In this work, Koyamada's algorithm [5], being one of the outstanding algorithms of image-space methods, is selected for parallelization. Koyamada's algorithm inherently exploits the object-space coherency available in the volume through connectivity relation. Furthermore, the algorithm handles the first ray-cell intersections for ray-segment generations very efficiently by exploiting the image-space coherency. Another superiority of the algorithm is the way that it handles the resampling operations utilizing the linear sampling method. Moreover, its novel approach of determining ray-face intersections enables the use of these results in resampling phase to reduce the amount of interpolation operations.

Parallelization of ray-casting based DVR constitutes a very interesting case for domain decomposition and mapping. This application can be considered as containing two interacting domains, namely *image-space* and *object-space*. Object space is a 3D domain containing the scene (volume data) to be visualized. Image space is a 2D domain containing pixels from which rays are shot into the 3D object domain to determine the color values of the respective pixels. Based on these domains, there are basically two approaches for parallel DVR; *image-space parallelism* and *object-space parallelism*. The focus of this work is object-space (OS) parallelism for DVR. In OS parallelism, 3D object domain is decomposed into K disjoint subvolumes and each subvolume is concurrently rendered by a distinct processor of a parallel machine with K processors. At

the end of this local rendering phase, partial image structures are created at each processor. In the pixel-merging phase, these image structures are merged over the interconnection network.

Most of the previous work on parallel DVR is for structured grids. Some of these approaches and related references can be found in [6, 7, 8, 9, 10, 11, 12]. The research work on parallel DVR of unstructured grids is relatively sparse [13, 14, 15, 16, 17, 18]. Most of these works are for shared-memory architectures and only Ma's work [18] considers the parallel DVR of unstructured grids on distributed-memory architectures. The multicomputer used is an Intel Paragon with 128 processors. Ma's OS-parallel algorithm uses the graph partitioning (GP) tool Chaco [19] for the OS-decomposition. Unit node and edge weighting is used in his graph model, so that the GP algorithm generates subvolumes containing equal number of cells. The ray-casting based DVR algorithm of Garrity [20] is used to render local subvolume in each processor. This static mapping is not altered when viewing parameters change. For pixel merging, image-space is evenly divided into horizontal strips, which are assigned to distinct processors. Pixel merging and local rendering computations are overlapped to reduce time.

The disadvantages of Ma's OS-parallel DVR algorithm can be summarized as follows. Experimental observations indicate that having equal number of cells in each subvolume may result in extremely poor load balance due to large cell-size variations in unstructured grids. Our experimentations on benchmark volumetric datasets show that computational load imbalance can be as high as 500% on 24 processors. Similar situation holds for unit edge weighting scheme. Furthermore, the view-independent OS decomposition adopted in Ma's algorithm does not consider the fact that the computational structure of DVR for unstructured grids substantially change with changing viewing parameters. Finally, the sequential DVR algorithm [20] employed in the local rendering computations is slow, thus hiding many overheads of the parallel implementation.

The contributions of our work can be summarized as follows. The OS decomposition problem for OS-parallel DVR is modeled as a GP problem with

correct view-dependent node and edge weighting. In the proposed model, minimizing the cutsize of a partition of the visualization graph corresponds to minimizing both the total volume of inter-processor communication during pixel merging computations due to ray-segment migration, and the total amount of redundant calculations during local rendering computations due to the additional ray-segments generated as a result of the partitioning. Maintaining the balance criterion corresponds to maintaining the computational load balance among processors during the local rendering computations.

As the visualization process will be carried out on the same parallel machine where simulations are done, the volumetric dataset representing the simulation results initially resides on the parallel machine in a distributed manner. However, the computational structure of parallel DVR of the first visualization instance is substantially different from that of the parallel simulation. Furthermore, the computational structures of successive visualization instances with different viewing parameters may also considerably change. Thus, a new OS decomposition is necessary for each parallel visualization instance, which will change the mapping of the cells of the dataset. Hence, each OS decomposition should also consider the minimization of the amount of cell migration to incur because of the difference in the new and existing mappings of the cells. This problem constitutes a very typical case of a general problem known as the *remapping* problem. In this work, we propose a novel graph theoretical model which enables a one phase solution of the general remapping problem through GP. The proposed model generates a remapping graph by node and edge augmentations to the graph representing the computational structure of the parallel application. A MeTiS based remapping tool, which is referred to here as *ReMapping-MeTiS* (RM-MeTiS), is developed by modifying and enhancing the original MeTiS package for partitioning the remapping graph.

In this work, a novel OS-parallel DVR algorithm is proposed and implemented. The proposed algorithm consists of four consecutive phases, namely, *clustering*, *remapping*, *local rendering* and *pixel merging* phases. In the clustering phase, GP is used for top-down clustering of the cells of the volumetric dataset to induce more tractable contracted visualization graph instances for the following view-dependent remappings. In the remapping phase, the nodes

and edges of the contracted visualization graph are weighted according to the current viewing parameter by exploiting the efficient node weight and edge weight estimation schemes proposed in this work. The contracted visualization graph is augmented to construct the remapping graph by using the current cluster mapping. Then, the remapping graph is partitioned using RM-MeTiS to find the new mappings for the clusters. Finally, those clusters whose new mappings differ from the current mappings migrate to their new home processors. In the local rendering phase, each processor concurrently renders its local subvolume formed by the set of local clusters by using Koyamada's sequential algorithm. In the pixel merging phase, each processor produces the final image of a distinct screen subregion. In this work, three screen decomposition and assignment schemes are proposed and implemented which consider the minimization of load imbalance and/or communication overhead due to ray-segment migration. Note that clustering is a view-independent preprocessing phase which is performed only once before the first visualization instance. So, the last three phases are effectively executed for the successive visualization instances.

The organization of the thesis is as follows. In Chapter 2, basic concepts of DVR, major data structures, Koyamada's DVR algorithm and execution-time analysis of Koyamada's algorithm are presented. Issues in OS decomposition and the GP-based OS decomposition model are discussed in Chapter 3. Chapter 4 presents the definition of the remapping problem and the proposed GP-based remapping model. The enhancements and modifications introduced to MeTiS package for developing RM-MeTiS are discussed in Chapter 5. In Chapter 6, the details of the proposed DVR algorithm are presented and discussed. Finally, Chapter 7 gives the experimental results obtained by running the proposed DVR algorithm on a Parsytec CC-24 system, which is a distributed memory multicomputer, for the visualization of benchmark volumetric datasets.

Chapter 2

Sequential DVR Algorithm

In this chapter, basic concepts of DVR, major data structures, Koyamada's DVR algorithm, and execution-time analysis of Koyamada's algorithm are presented.

2.1 Preliminaries

A dataset is called *volumetric dataset* or *volume data* if data points of the set are defined in 3D space in a volume. The term *sample point* refers to a point with 3D spatial coordinates for which a numerical value is associated. Sample points in the volume data are connected in a predetermined way to form *volume elements*, also referred to here as *cells*. Sample points that form a cell are called *vertices* of the cell. There are various cell shapes: rectangular prism, hexahedron, tetrahedron and polyhedron being the most common ones. Our work is based on the tetrahedral cell type. A tetrahedral cell permits the direct interpolation of a point inside it by its four vertices. It also has the advantage that the data distribution is linear in any direction inside the cell. Note that other cell types mentioned can be divided into tetrahedral cells through a preprocessing step called tetrahedralization [21, 22].

In the tetrahedral cell model, each cell contains four vertices and four triangular faces. The *face normal* is defined to be oriented outward from the parent

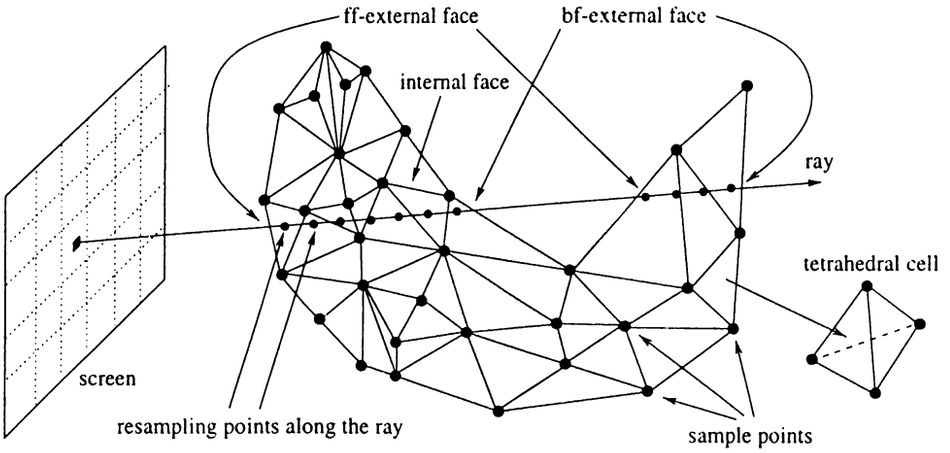


Figure 2.1: Ray-casting based direct volume rendering.

cell. If a face of a cell is shared by two cells, that face is called *internal*. If it is not shared by any other cell, the face is called *external*. A cell with at least one external face is called an *external cell*. Otherwise, it is called an *internal cell*.

Visualization of a volumetric dataset $\mathcal{V}$ for the given viewing parameter v is called a *visualization instance* denoted by the 2-tuple $(\mathcal{V}, v)$. The viewing parameter v consists of view-direction vector, view-up vector, view-reference point, view-plane window and image resolution. The vertices of the volumetric dataset $\mathcal{V}$ are initially in *world space coordinate* (WSC) system, and they are transformed into the *normalized projection coordinate* (NPC) system by using the viewing parameter v. In NPC system, $\mathcal{V}$ is viewed in the positive z direction, and the (x, y) coordinate values of vertices of $\mathcal{V}$ are in fact their (x, y) coordinate values on the image-plane. In a visualization instance $(\mathcal{V}, v)$, a face of a cell of $\mathcal{V}$ is a *front-facing* (ff) face or a *back-facing* (bf) face if the z component of the face normal in NPC is negative or positive, respectively. In other words, a ray enters a cell through an ff-face and exits the cell through a bf-face. An external cell containing at least one ff-external face is called as a *front-external* cell.

There are various types of sampling schemas in the literature. The major factors that affect the type of sampling schema used are data resolution, the variation of data values, and the variation of the transfer function. The most

common three types of schemas are: *mid-point sampling*, *equi-distant sampling*, and *adaptive sampling*. In this work, equi-distant sampling is used. In this schema, samples are generated at fixed intervals of length Δz. Hence, for some cells more than one samples will be generated, but there will be cases such that no sample is taken in a cell. But this problem may be solved by choosing Δz small enough so that at least one sample will be taken in each cell [18]. Figure 2.1 displays the some of the concepts introduced in this section.

2.2 Major Data Structures

The first of the data structures is the data structure that stores the tetrahedral cell data. This consists of two arrays: the *VertexArray* and the *CellArray*. *VertexArray* structure keeps the scalar values, the WSC values and the NPC values of the vertices of the volumetric dataset. The *CellArray* structure keeps the following two components for each cell of the dataset:

1. A *Vertices* component stores the identifiers of four vertices of the cell.
2. A *NeighborCells* component stores the indices of four neighbors of the cell through its four faces and also identifies each of its internal faces by its index in the respective cell. A sentinel value is used for external faces. Each of the four local face indices represents a value between 0-3 which is an index to one of the four faces in the neighbor cell. This component is exploited to avoid the search for finding the entry face of the ray to the next cell during the ray-casting process.

The second major data structure is the *RayBuffer*. It is a 2D virtual array that holds a linked list of composited ray-segments for each pixel location of the screen. Each element of a linked list stores the following 2 components:

1. A *ZDepth* component which denotes the z coordinate of the exit point of the respective ray-segment. The lists are maintained in sorted increasing order according to this component.

2. A *Color-Opacity* component which holds the composited RGB color values and the opacity value of the respective ray-segment.

2.3 Koyamada's Algorithm

Koyamada's algorithm [5] is a ray-casting approach that makes use of the image space coherency to generate rays, and follows those rays in the object-space. Therefore, the first step of his algorithm generates the ray-segments to be traced. In his original algorithm, he sorts the ff-external faces with respect to z coordinates of their centroids in increasing order. This, in fact, is an approximate order, which may be wrong in some cases [5]. As the objective in this work is producing high quality and correct images in a fast way, this step of the original algorithm is slightly modified. Instead of sorting the external faces in the beginning, we scan convert them one by one in any order and for each pixel covered by the projection area of an ff-external face, we generate a ray-segment, traverse it through the volume for composition until it exits from a bf-external face, and finally we insert the composited ray-segment into the respective list in the *RayBuffer* structure.

Each ray is followed in the volume data utilizing the connectivity information between cells. To trace a ray inside the volume, two things have to be known for each cell that is hit by the ray; the entry face and the (z, s) values at the entry point to the cell, and the exit face and the (z, s) values at the exit point from the cell. Note that the exit face and the (z, s) values at the exit point from the cell, are the entry face and the (z, s) values at the entry point to the next cell that the ray intersects. As for each ray-segment, the values at the point that the ray-segment first penetrates into the volume is determined during the scan conversion of ff-external faces, the problem of tracing a ray-segment inside the volume reduces to the problem of determining the exit point from a cell, where the entry point is given. Koyamada's method relies on the observation that if a ray intersects a face, then the pixel that the ray is shot must be covered by the projection area of that face on the screen. So, he uses the projected area of a face to determine if the ray exits the cell from that face. His method uses the *NPC* values of the vertices of a face to take this decision.

Consider a ray r shot from the pixel location (x_r, y_r) which intersects a tetrahedral cell $ABCD$ through entry-point P of entry-face ABD as shown in Fig. 2.2. Let triangle ACD be the face of the cell that is subject to the ray-face intersection test. If the projection of vectors $\vec{AC}$ and $\vec{AD}$ are parallel, that is the face is perpendicular to screen, then the ray does not leave the cell through that face, so another face of the cell is tested. Otherwise, the ray r intersects the plane determined by the triangle ACD at a point Q. As the ray passes through points P and Q, (x, y) coordinate values (in NPC) of points P and Q are equal to (x_r, y_r), i.e., $x_r = x_P = x_Q$ and $y_r = y_P = y_Q$. Then, the vector $\vec{AQ}$ can be expressed as a linear combination of the vectors $\vec{AC}$ and $\vec{AD}$ as:

$$\vec{AQ} = \alpha\vec{AC} + \beta\vec{AD}. \tag{2.1}$$

Here, α and β are weighting values, and are found by solving

$$\begin{bmatrix} x_C - x_A & x_D - x_A \\ y_C - y_A & y_D - y_A \end{bmatrix} \times \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} x_r - x_A \\ y_r - y_A \end{bmatrix} \tag{2.2}$$

where (x_C, y_C), (x_D, y_D) and (x_A, y_A) are the coordinates of points C, D and A in NPC, respectively. If α and β do not satisfy the three conditions $\alpha \geq 0$, $\beta \geq 0$ and $\alpha + \beta \leq 1$, then the point Q is not inside the triangle ACD, so another face is tested. Otherwise, the point Q is inside the triangle ACD, so no further tests need to be done on other candidate faces. As the exit face is identified, the (z, s) values (z_Q, s_Q) at the exit point Q are calculated as:

$$z_Q = z_A + \alpha(z_C - z_A) + \beta(z_D - z_A), \tag{2.3}$$

$$s_Q = \alpha s_C + \beta s_D + (1 - \alpha - \beta)s_A. \tag{2.4}$$

Equation 2.3 is in fact writing Eq. 2.1 for the z components. Equation 2.4 is 2D inverse distance interpolation, where α, β, and $(1-\alpha-\beta)$ denote the ratios of areas of triangles QAD, QAC, and QCD to the area of triangle ACD, respectively.

As the exit-point (z_Q, s_Q) values are computed and the entry-point (z_P, s_P) values are already known, the next step is to resample and composite along the ray between the entry and exit points P and Q according to the equi-distant sampling scheme. Koyamada's algorithm exploits the fact that the change of the scalar in any direction is linear in a tetrahedral cell to speed up

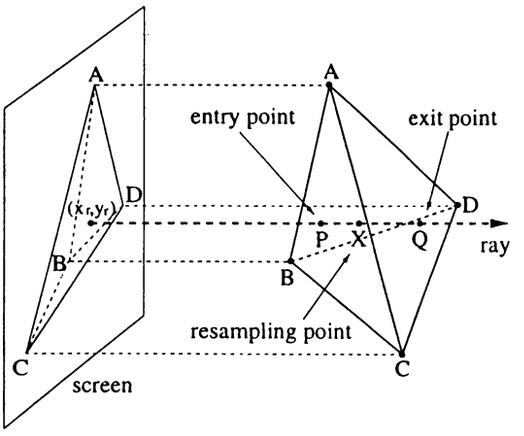


Figure 2.2: Ray-face intersection and linear sampling scheme in Koyamada's algorithm.

the interpolation operations for resampling through utilizing *linear sampling* method. That is, the scalar value s_X at each resampling point X along the line segment PQ is efficiently computed using 1D inverse distance interpolation formula as $s_X = \gamma s_Q + (1-\gamma) s_P$, where γ is the ratio of the length of the line segment PX to the length of line segment PQ. Then, the scalar value s_X is mapped to a color C_X and an opacity value O_X through applying a *transfer function* which converts the numerical value s_X to color and opacity values to represent the characteristics of the physical environment and simulation results. These color and opacity values are composited in *front-to-back* composition order as

$$O_{i+1} = O_i + O_X(1 - O_i), \tag{2.5}$$

$$C_{i+1} = (C_i O_i + C_X O_X(1 - O_i))/O_{i+1}. \tag{2.6}$$

where (C_i, O_i) and (C_{i+1}, O_{i+1}) values are the composited (color,opacity) values before and after processing the resampling point X, respectively [3]. Initially, C_0 and O_0 are set to zero.

At the end of this process, the *RayBuffer* structure contains all the composited ray-segments. The lengths of individual ray-lists in the *RayBuffer* structure are completely dependent on the visualization instance. If the rate of non-convexity is high for the given viewing parameters, then this means that a ray shot at a pixel has entered, and exited the volume many times. For example, the ray shown in Fig. 2.1 generates two ray-segments. For convex

datasets, the length of each ray list in the *RayBuffer* structure is either 0 or 1, hence the color field of the single ray-segment of each active pixel contains the final color of the respective pixel of the image. On the other hand, non-convex datasets necessitate a final composition process over each ray-segment list of multiple length.

The nice features of Koyamada's DVR algorithm can be summarized as follows. It makes use of image-space coherency for ray-segment generation through scan-conversion. It performs 2 ray-face intersection tests per cell on the average whereas the conventional approach always checks 3 faces [20]. It performs a 1D inverse distance interpolation for each resampling, and a 2D inverse distance interpolation for each ray-cell intersection, whereas the conventional approach performs expensive 3D inverse distance interpolation for each resampling. Furthermore, it uses the results of the ray-face intersection operations to reduce the amount of 2D interpolation operations.

2.4 Execution Time Analysis

The parallelization scheme used in this work is based on graph-theoretical static decomposition and mapping of the volumetric dataset. This scheme necessitates the estimation of computational load of rendering a subvolume for static load balancing.

The execution-time $T_{\mathcal{V}}^{v}$ of Koyamada's algorithm for a visualization instance $(\mathcal{V}, v)$ can be dissected into four components, T_{NT}, T_R, T_I and T_S. The node transformation time T_{NT} involves the computations for transforming WSC values of vertices to NPC values. This component can be neglected in the computational load estimation since it is extremely small compared to the total rendering time $T_{\mathcal{V}}^{v}$ (below 0.1%). The ray-segment generation time T_R involves the scan-conversion of ff-external faces to compute the intersections of ray-segments with ff-external faces and the (z, s) values at these intersection points. The ray-face intersection time T_I involves the computation of the intersections of the ray segments with bf-faces, the scalar values, and the scalar gradients at the intersections (Eqs. 2.2–2.4). The sampling time T_S involves

the computation of the scalar values at sampling points, mapping the scalar values to colors and opacities using a transfer function and the composition of these values. Hence, the total execution time T_V^v can be written as:

$$T_V^v = T_R + T_I + T_S = R_V^v t_R + I_V^v t_I + S_V^v t_S, \tag{2.7}$$

where R_V^v, I_V^v and S_V^v denote the numbers of ray-segments generated, ray-face intersections performed and samples taken respectively. In Eq 2.7, t_R, t_I and t_S represent the unit cost of respective computations. These unit costs can not be determined through measurement because of the highly interleaved execution manner of the respective types of computations. Instead, we have estimated these unit costs statistically using the *Least-Squares Approximation* method. Our experimental analysis show that the average error in estimating T_V^v using Eq. 2.7 is below 3.5%. Efficient schemes for estimating the R_V^v, I_V^v and S_V^v counts will be described in Section 6.2.1.

Chapter 3

Issues in Object-Space Decomposition

Koyamada's algorithm, being an image-space method, successfully exploits the *object-space* (OS) coherency through the connectivity information available in the volume data. Therefore, the key point for a successful OS parallelization is to find an OS decomposition which maintains the OS coherency and computational load balance as much as possible. In OS decomposition, 3D object domain is subdivided into disjoint subvolumes and both the computations and the data associated with all cells in each subvolume are assigned to a distinct processor for rendering. In this work, we model the OS decomposition problem as a *graph partitioning* problem.

3.1 Graph Partitioning (GP) Problem

A weighted undirected graph $\mathcal{G} = (\mathcal{N}, \mathcal{E}, w)$ is defined as a set of nodes $\mathcal{N}$, a set of edges $\mathcal{E}$, and weighting functions w defined on nodes and edges. Every edge $e_{ij} \in \mathcal{E}$ connects a pair of distinct nodes n_i and n_j. The degree d_i of a node n_i is equal to the number of edges incident to n_i. The functions $w(n_i)$ and $w(n_i, n_j)$ denote the weights of a node $n_i \in \mathcal{N}$ and an edge $e_{ij} \in \mathcal{E}$, respectively.

$\Pi = \{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_K\}$ is a *K-way partition* of $\mathcal{G}$ if the following conditions

hold: each part $\mathcal{P}_k, 1 \leq k \leq K$, is a nonempty subset of $\mathcal{N}$, parts are pairwise disjoint ($\mathcal{P}_k \cap \mathcal{P}_\ell = \emptyset$ for all $1 \leq k < \ell \leq K$), and union of K parts is equal to $\mathcal{N}$ (i.e., $\bigcup_{k=1}^{K} \mathcal{P}_k = \mathcal{N}$). A K-way partition is also called a *multiway* partition if $K > 2$ and a *bipartition* if $K = 2$. A partition is said to be balanced if each part $\mathcal{P}_k$ satisfies the *balance criterion*

$$W_{avg}(1 - \varepsilon) \leq W_k \leq W_{avg}(1 + \varepsilon), \quad for\ k = 1, 2, \ldots, K. \tag{3.1}$$

In Eq. 3.1, the size W_k of a part $\mathcal{P}_k$ is defined as the sum of the weights of the vertices in that part (i.e., $W_k = \sum_{n_i \in \mathcal{P}_k} w(n_i)$), $W_{avg} = (\sum_{n_i \in \mathcal{N}} w(n_i))/K$ denotes the size of each part under ideal balance condition, and ε represents a predetermined maximum imbalance ratio allowed.

In a partition Π of $\mathcal{G}$, an edge is said to be *cut* if its pair of vertices belong to two different parts, and *uncut* otherwise. The set of cut edges for a partition Π constitutes its edge-cut, and is denoted here as $\mathcal{E}_{cut}$. The *cutsize* definition for representing the cost $\chi(\Pi)$ of a partition Π is

$$\chi(\Pi) = \sum_{e_{ij} \in \mathcal{E}_{cut}} w(n_i, n_j). \tag{3.2}$$

In Eq. 3.2, each cut edge e_{ij} contributes its cost $w(n_i, n_j)$ to the cutsize. Hence, K-way graph partitioning problem can be defined as the task of dividing a graph into K parts such that the cutsize is minimized, while the balance criterion among part sizes is maintained.

3.2 A GP-based OS Decomposition Model

A visualization instance $(\mathcal{V}, v)$ is represented as a weighted undirected graph $\mathcal{G}_{\mathcal{V}}^{v} = (\mathcal{N}_{\mathcal{V}}, \mathcal{E}_{\mathcal{V}}, w_{\mathcal{V}}^{v})$. The nodes in the node set $\mathcal{N}_{\mathcal{V}}$ correspond to the cells of the dataset. Each node $n_i \in \mathcal{N}_{\mathcal{V}}$ corresponds to the atomic task of rendering computations associated with cell c_i. For any cell, rendering computations involve ray-face intersection tests performed on its back faces, and sampling and composition operations performed along the rays inside the cell. A front external cell involves the additional computation of ray-segment generation for its ff-external face(s), through scan-conversion. So, the computational load

of an internal cell c_i and a front-external cell c_j are $w(n_i) = I_i t_I + S_i t_S$ and $w(n_j) = I_j t_I + S_j t_S + R_j t_R$, respectively. Here, I_i and S_i denote the numbers of ray-face intersections and sampling operations associated with cell c_i respectively, and R_j denotes the number of ray-segments generated by the front external cell c_j. Recall that t_I, t_S and t_R are the unit costs of the respective operations (Eq. 2.7).

In the edge set $\mathcal{E}_V$, $e_{ij} \in \mathcal{E}_V$ if and only if cells c_i and c_j share a face f_{ij}. In tetrahedral cell model, the given edge-set definition generates exactly one edge between each neighbor cells, since each pair of neighbor cells shares exactly one face. Furthermore, each cell can have at most four neighbor cells, hence $d_i \leq 4$. Relative to the given viewing parameter v, either c_i is *behind* c_j, or vice versa. Here, we define the behind relation $<_v$ such that $c_i <_v c_j$ if and only if cells c_i and c_j share a face f_{ij} and any ray $r \in \mathcal{R}_{ij}$ intersecting face f_{ij} first hits cell c_j and then hits cell c_i [17]. The relation $c_i <_v c_j$ also denotes that f_{ij} is a bf-face of cell c_j, whereas it is a ff-face of cell c_i. Without loss of generality let us assume that c_i is behind c_j. Edge e_{ij} represents the dependency of cell c_i to c_j on the composition operations along the rays in $\mathcal{R}_{ij}$, since the composition operation is not commutative. Fortunately, this sequential nature of the composition operation can be avoided by exploiting its associativity. That is, if cells c_i and c_j are mapped to two distinct processors, cell c_i can generate ray-segments for the rays in $\mathcal{R}_{ij}$ and initiate the traversal and composition of these ray-segments without waiting the composition results of the respective rays from cell c_j. However, these partial composition results obtained by cells c_i and c_j for rays in $\mathcal{R}_{ij}$ should be merged according to the visibility order determined by the behind relation. Hence, any cut edge e_{ij} will incur interprocessor communication because of these merge operations. The volume of communication will be proportional to the number of rays in $\mathcal{R}_{ij}$. Note that $|\mathcal{R}_{ij}| = R_{ij}$ is in fact the number of pixels covered by the projection area of the face f_{ij}.

Each cut edge e_{ij} will also disturb the OS coherency utilized by Koyamada's algorithm. In Koyamada's algorithm, for any ray $r \in \mathcal{R}_{ij}$ intersecting face f_{ij}, the exit-point (z, s) values computed by cell c_j are directly used by the cell c_i as the entry-point (z, s) values for linear sampling and composition. So, any

cut edge e_{ij} will introduce the redundant computation of entry-point (z, s) values from scratch for each ray in $\mathcal{R}_{ij}$ during ray-segment generation. The amount of redundant computation will also be proportional to R_{ij}.

By setting the weight $w(n_i, n_j) = R_{ij}$ for each edge e_{ij}, OS decomposition for a visualization instance, reduces to the K-way partitioning of its associated graph $\mathcal{G}_V^v$ according to the balance criterion (Eq. 3.1) and the cutsize definition (Eq. 3.2), where K denotes the number of processors. Each part $\mathcal{P}_k$ in a partition Π of $\mathcal{G}_V^v$ corresponds to a subvolume $\mathcal{V}_k$ to be rendered simultaneously and independently by a distinct processor P_k for $k = 1, 2, \ldots, K$. Minimizing the cutsize according to Eq. 3.2 corresponds to minimizing both the total volume of interprocessor communication and the total amount of redundant computation. Maintaining the balance criterion according to Eq. 3.1 corresponds to maintaining the computational load balance among processors during local rendering calculations.

Consider a cut-edge e_{ij} in a partition Π such that $n_i \in \mathcal{P}_k$ and $c_i <_p c_j$. Then, e_{ij} incurs the above mentioned redundant computation to processor P_k because of cell c_i. This redundant computation associated with cell c_i is same as the ray-segment generation operations performed for the front external cells in the sequential algorithm. In fact, face f_{ij} is not shared by two cells in subvolume $\mathcal{V}_k$, and it can be considered as a ff-external face of the subvolume $\mathcal{V}_k$. Each face f_{ij} corresponding to cut edge e_{ij}, and any cell which contains at least one such face will be referred to here as ff-*boundary* face and front-boundary cell, respectively. Hence, the node-weight computation scheme mentioned for front-external cells should also be used for the front-boundary cells.

3.3 Remapping Problem in OS Decomposition

As mentioned earlier, the main objective in parallel DVR is to visualize the volumetric datasets produced by engineering simulations and scientific computations on the same parallel machine, where these computations are held.

So, the volumetric dataset representing the simulation results resides on the parallel machine in a distributed manner. However, this data distribution is according to the efficient parallelization of simulation computations, which may drastically differ from the decomposition criteria for efficient parallelization of DVR. Hence, a new decomposition is necessary. Furthermore, multiple visualizations of the same dataset for different viewing parameters are usually needed for a better understanding of the visualization results. However, the structure of rendering computation may substantially change with changing viewing parameters. Thus, new decompositions are also necessary between successive visualizations. However, each new decomposition and mapping may incur excessive cell migration because of the difference in the new and existing mappings of the cells. Therefore, the OS decomposition model should also consider the minimization of this data redistribution overhead. This problem constitutes a very typical case of a general problem known as the *remapping problem*.

Chapter 4

A GP-based Remapping Model

In the remapping problem, the computational structure of an application to be parallelized changes from one phase of the computation to another. The quality of the existing mapping may deteriorate both in terms of load balance and interprocessor communication. So, we should adapt the mapping in accordance with these changes in the computation which necessitate the migration of computational tasks together with their associated data structures. The objective in each remapping step is to minimize the total overhead due to the task migration and the mapping of interacting tasks to different processors, while improving the load balance.

4.1 Problem Definition

A remapping problem instance is defined by the three-tuple $(\mathcal{G}, \mathcal{M}, \mathcal{T})$. Here, $\mathcal{G} = (\mathcal{N}, \mathcal{E}, w)$ denotes the computational graph [23] representing the *modified* computational structure of the application. Nodes represent atomic computations which can be executed simultaneously and independently. The weight $w(n_i)$ of node n_i denotes the computational load of the respective task. Each edge denotes the need for the bidirectional interaction between the respective pair of computations. The weight $w(n_i, n_j)$ of edge e_{ij} denotes the amount of

respective interaction. In other words, it represents the amount of communication and redundant computation to incur when n_i and n_j are remapped to distinct processors. $\mathcal{M}$ denotes the current K-way task-to-processor mapping function, where $\mathcal{M}(n_i) = k$ means that the respective task currently resides in processor P_k. $\mathcal{T}$ denotes the task-migration cost function, where $\mathcal{T}(n_i)$ is the cost of migrating the respective task from its current processor $P_{\mathcal{M}(n_i)}$ to another processor due to remapping. This cost usually refers to the communication cost for the data associated with a task.

4.2 Two-Phase Solution Model

A straightforward approach for solving the remapping problem is to follow a *two-phase* scheme. In the first phase, K-way partitioning is performed on graph $\mathcal{G}$ as described in Section 3.1, and a partition Π is obtained. Then, Π, $\mathcal{M}$ and $\mathcal{T}$ are used to construct a weighted bipartite graph for the second phase. The K parts of Π and K processors constitute the two partite nodes sets $\mathcal{X}$ and $\mathcal{Y}$ of the bipartite graph $\mathcal{B} = (\mathcal{X}, \mathcal{Y}, \mathcal{E})$ such that x_k and y_ℓ denote part $\mathcal{P}_k$ and processor P_ℓ, respectively. There exist an edge $\epsilon_{k\ell}$ between x_k and y_ℓ if there exists at least one task n_i in $\mathcal{P}_k$ that currently resides in processor P_ℓ (i.e., $n_i \in \mathcal{P}_k$ and $\mathcal{M}(n_i) = \ell$). The weight of an edge $\epsilon_{k\ell}$ is equal to the sum of the migration costs of those tasks of part $\mathcal{P}_k$ which currently reside in processor P_ℓ. An optimal part-to-processor assignment can be found by solving the maximum weight perfect matching problem in $\mathcal{B}$. Each matched edge $e_{k\ell}$ in the matching incurs the remapping of the tasks of the part $\mathcal{P}_k$ to processor P_ℓ. In two phase approaches, solving the partitioning problem separately from the assignment problem usually restricts the quality of the remapping, because decisions made during the partitioning phase may prevent finding a good remapping in the second phase.

4.3 One-Phase Solution Model

In this work, we propose a novel graph-partitioning based model for the solution of the general remapping problem using a one phase approach. In the proposed model, we construct a new graph $\tilde{\mathcal{G}} = (\tilde{\mathcal{N}}, \tilde{\mathcal{E}}, \tilde{w})$, referred to here as the *remapping graph*, through augmenting the computational graph $\mathcal{G}$. We add a node p_k for each processor P_k to the node set $\mathcal{N}$ of $\mathcal{G}$ to obtain the node set $\tilde{\mathcal{N}}$ of $\tilde{\mathcal{G}}$. These added nodes and the original nodes of $\mathcal{G}$ will be referred to here as the *processor-nodes* (p-nodes) and *task-nodes* (t-nodes) of $\tilde{\mathcal{G}}$, respectively. In order to obtain the edge set $\tilde{\mathcal{E}}$ of $\tilde{\mathcal{G}}$, we augment the edge set $\mathcal{E}$ by connecting each p-node to those t-nodes corresponding to the tasks residing in the respective processor according the current mapping function $\mathcal{M}$. These added edges and the original edges will be referred to here as *processor-to-task* edges (pt-edges) and *task-to-task* edges (tt-edges), respectively. That is,

$$\tilde{\mathcal{N}} = \tilde{\mathcal{N}}_t \cup \tilde{\mathcal{N}}_p \text{ and } \tilde{\mathcal{E}} = \tilde{\mathcal{E}}_{tt} \cup \tilde{\mathcal{E}}_{pt} \quad \text{where} \tag{4.1}$$

$$\tilde{\mathcal{N}}_t = \mathcal{N} \text{ and } \tilde{\mathcal{N}}_p = \{p_1, p_2, \ldots, p_K\}, \tag{4.2}$$

$$\tilde{\mathcal{E}}_{tt} = \mathcal{E} \text{ and } \tilde{\mathcal{E}}_{pt} = \{(n_i, p_k) : n_i \in \tilde{\mathcal{N}}_t, p_k \in \tilde{\mathcal{N}}_p \text{ and } \mathcal{M}(n_i) = k\}. \tag{4.3}$$

In node weighting, weights of t-nodes remain the same, whereas p-nodes are assigned zero weights. In edge weighting, weights of tt-edges remain the same, whereas each pt-edge will be assigned the task-migration cost of the respective task. That is,

$$\tilde{w}(n_i) = w(n_i) \;\; \forall n_i \in \mathcal{N}_t \;\; \text{and} \;\; \tilde{w}(p_k) = 0 \;\; \forall p_k \in \mathcal{N}_p, \tag{4.4}$$

$$\tilde{w}(n_i, p_k) = \mathcal{T}(n_i) \;\; \forall e_{ik} \in \mathcal{N}_{pt} \;\; \text{and} \;\; \tilde{w}(n_i, n_j) = w(n_i, n_j) \;\; \forall e_{ij} \in \mathcal{N}_{tt}. \tag{4.5}$$

A K-way partition $\tilde{\Pi} = \{\tilde{\mathcal{P}}_1, \tilde{\mathcal{P}}_2, \ldots, \tilde{\mathcal{P}}_K\}$ of graph $\tilde{\mathcal{G}}$ is defined to be feasible if it satisfies the *mapping constraint*

$$\left|\tilde{\mathcal{P}}_k \bigcap \tilde{\mathcal{N}}_p\right| = 1 \quad \text{for} \quad k = 1, 2, \ldots, K. \tag{4.6}$$

That is, each part $\tilde{\mathcal{P}}_k$ of $\tilde{\Pi}$ contains exactly one p-node. Then, a feasible partition $\tilde{\Pi}$ of $\tilde{\mathcal{G}}$ induces the remapping $\tilde{\mathcal{M}}$ in which tasks corresponding to the t-nodes in each part are all assigned to the same processor corresponding to the unique p-node in that part. That is, $\tilde{\mathcal{M}}(n_i) = k$ if both the t-node n_i and the p-node p_k are in the same part of $\tilde{\Pi}$.

One phase solution to the remapping problem reduces to the K-way partitioning of graph $\dot{\mathcal{G}}$ according to the mapping constraint (Eq. 4.6), balance criterion (Eq. 3.1), and the cutsize definition (Eq. 3.2). Maintaining the balance criterion corresponds to maintaining the computational load balance among processors, since p-nodes are assigned zero weights. Minimizing the cutsize corresponds to minimizing the total overhead due to task migration and interactions between tasks mapped to different processors. Note that there might be two types of edges in the edge-cut. A tt-edge e_{ij} in the edge-cut incurs communication between processors $P_{\dot{\mathcal{M}}(n_i)}$ and $P_{\dot{\mathcal{M}}(n_j)}$ because of the interactions between atomic tasks corresponding to the t-nodes n_i and n_j. Such tt-edges in the edge-cut may also incur redundant computations. A pt-edge e_{ik} in the edge-cut incurs interprocessor communication due to migration of the task corresponding to the t-node n_i from processor P_k to processor $P_{\dot{\mathcal{M}}(n_i)}$. Note that the task-migration cost function $\mathcal{T}$ should be wisely selected to incorporate the appropriate scaling between weights of tt-edges and pt-edges. In the absence of redundant computation, the scaling can be inherently solved by weighting both types of edges in terms of their relative communication volume requirements.

The proposed graph-partitioning based remapping model is exploited for OS decomposition in our parallel DVR. In the OS decomposition, the remapping instance is identified by three-tuple $(\mathcal{G}_{\mathcal{V}}^{v}, \mathcal{M}, \mathcal{T})$. Here the graph $\mathcal{G}_{\mathcal{V}}^{v}$ represents the computational structure of the visualization instance $(\mathcal{V}, v)$ as mentioned in Section 3.2. $\mathcal{M}$ is the current cell-to-processor mapping inherited either from the parallel simulation or the previous parallel visualization. The cell-migration cost $\mathcal{T}$ can be constructed through a scaling which considers only the communication volume overhead. So, $\mathcal{T}(n_i)$ can be set equal to the ratio of the size of the data associated with cell c_i to the size of an individual ray-segment information.

Chapter 5

MeTiS-based Remapping Heuristic

5.1 Graph Partitioning Heuristics

Kernighan-Lin (KL) based heuristics are widely used for graph/hypergraph partitioning because of their short run-times and good quality results. KL algorithm is an iterative improvement heuristic originally proposed for bipartitioning [24]. KL algorithm, starting from an initial bipartition, performs a number of passes until it finds a locally minimum partition. Each pass consists of a sequence of vertex swaps. Fiduccia-Mattheyses (FM) [25] introduced a faster implementation of KL algorithm by proposing vertex move concept instead of vertex swap. This modification as well as proper data structures, e.g., bucket lists, reduced the time complexity of a single pass of KL algorithm to linear in the size of the graph.

The performance of FM algorithms deteriorates for large and too sparse graphs. Furthermore, the solution quality of FM is not *stable* (*predictable*), i.e., average FM solution is significantly worse than the best FM solution, which

is a common weakness of move-based iterative improvement approaches. Random multi-start approach is used in VLSI layout design to alleviate this problem by running FM algorithm many times starting from random initial partitions to return the best solution found [26]. However, this approach is not viable in parallel computing since decomposition is a preprocessing overhead introduced to increase the efficiency of the underlying parallel algorithm/program. Most users will rely on one run of the decomposition heuristic, so that the quality of the decomposition tool depends equally on the worst and average decompositions than on just the best decomposition.

Recently, *multilevel* graph partitioning methods have been proposed leading to successful graph partitioning tools Chaco [19] and MeTiS [27]. These multilevel heuristics consists of 3 phases, namely *coarsening*, *initial partitioning*, and *uncoarsening*. In the first phase, multilevel clustering is successively applied starting from the original graph by adopting various matching heuristics until number of vertices in the coarsened graph reduces below a predetermined threshold value. In the second phase, coarsest graph is partitioned using various heuristics including FM. In the third phase, partition found in the second phase is successively projected back towards the original graph by refining the projected partitions on intermediate level uncoarser graphs using various heuristics including FM.

5.2 Modifying MeTiS Package for Remapping

In this work, we exploit the state-of-the-art MeTiS graph partitioning tool (KMeTiS option) to partition the remapping graph $\tilde{\mathcal{G}}$ for solving the K-way remapping problem. However, MeTiS can not handle the mapping constraint (Eq. 4.6) directly. In this section, we present our modifications and enhancements to each phase of the MeTiS package to make it support the mapping constraint. This version of MeTiS will be called as *ReMapping-MeTiS* (RM-MeTiS).

5.2.1 Coarsening Phase

In this phase of MeTiS, the given graph $\tilde{\mathcal{G}} = \tilde{\mathcal{G}}_0 = (\tilde{\mathcal{N}}_0, \tilde{\mathcal{E}}_0, \tilde{w}_0)$ is coarsened into a sequence of smaller graphs $\tilde{\mathcal{G}}_1 = (\tilde{\mathcal{N}}_1, \tilde{\mathcal{E}}_1, \tilde{w}_1)$, $\tilde{\mathcal{G}}_2 = (\tilde{\mathcal{N}}_2, \tilde{\mathcal{E}}_2, \tilde{w}_2)$, ..., $\tilde{\mathcal{G}}_m = (\tilde{\mathcal{N}}_m, \tilde{\mathcal{E}}_m, \tilde{w}_m)$, satisfying $|\tilde{\mathcal{N}}_0| > |\tilde{\mathcal{N}}_1| > |\tilde{\mathcal{N}}_2| > \ldots > |\tilde{\mathcal{N}}_m|$, until $|\tilde{\mathcal{N}}_m|$ reduces below a threshold value. This coarsening is achieved by coalescing disjoint subsets of vertices of graph $\tilde{\mathcal{G}}_\ell$ into *supernodes* of next level graph $\tilde{\mathcal{G}}_{\ell+1}$ through various randomized *matching* schemes. The weight of each supernode of $\tilde{\mathcal{G}}_{\ell+1}$ is set equal to the sum of its constituent nodes in $\tilde{\mathcal{G}}_\ell$. The edge set of each supernode is set equal to the weighted union of the edge sets of its constituent nodes. In randomized matching, nodes of $\tilde{\mathcal{G}}_\ell$ are visited in a random order. If a node $n_i \in \tilde{\mathcal{N}}_\ell$ has not been matched yet, one of its unmatched *adjacent* nodes is selected according to a criterion. If such a node n_j exists, the matched pair n_i and n_j is merged into a supernode of $\tilde{\mathcal{G}}_{\ell+1}$. If there is no unmatched adjacent node of n_i, then node n_j remains unmatched. Among various matching criteria available in MeTiS, the heavy edge matching scheme (HEM) is selected for RM-MeTiS. In HEM, a node n_i is matched with node n_j such that the weight of the edge between these two nodes, is maximum over all valid edges incident to node n_i.

At each level ℓ of the coarsening phase of MeTiS, the coarse graph $\tilde{\mathcal{G}}_\ell$ effectively induces a $|\tilde{\mathcal{N}}_\ell|$-way partition of the original graph $\tilde{\mathcal{G}}_0$. The idea behind RM-MeTiS, is to maintain the mapping constraint (Eq. 4.6) in a relatively relaxed manner such that the node cluster corresponding to each supernode of $\tilde{\mathcal{G}}_\ell$ contains at most one p-node of the original graph $\tilde{\mathcal{G}}_0$. In RM-MeTiS, we maintain a flag for each node to indicate whether it is a t-node or p-node. Matching two t-nodes in $\tilde{\mathcal{G}}_\ell$, produces a *t-supernode* in $\tilde{\mathcal{G}}_{\ell+1}$, whereas matching a t-node with a p-node produces a *p-supernode*. In randomized HEM matching, two unmatched supernodes are considered for matching only if at least one of them is a t-supernode. That is, at any coarsening level ℓ, the constituent nodes of a t-supernode in $\tilde{\mathcal{G}}_\ell$ are all t-nodes whereas the constituent nodes of a p-supernode in $\tilde{\mathcal{G}}_\ell$ are all t-nodes except one p-node. Note that there exist exactly K p-supernodes in $\tilde{\mathcal{G}}_\ell$ at each level ℓ during the coarsening phase.

5.2.2 Initial Partitioning Phase

The objective of this phase is to find a good K-way partition of the coarsest graph $\check{\mathcal{G}}_m$ which minimizes the cutsize (Eq. 3.2) while maintaining the balance criterion (Eq. 3.1). KMeTiS [28] constructs a K-way partition of $\check{\mathcal{G}}_m$ by recursive bisection using the multilevel bisection algorithm PMeTiS [29]. In this scheme, first a 2-way partition of $\check{\mathcal{G}}_m$ is obtained, and then this bipartition is further partitioned in a recursive manner. After $\lg_2 K$ phases graph $\check{\mathcal{G}}_m$ partitioned into K parts.

In RM-MeTiS, the objective of the initial partitioning phase is to find a good K-way partition of graph $\tilde{\mathcal{G}}_m$ which satisfies the mapping constraint (Eq. 4.6) in addition to the pure graph-partitioning criteria (Eqs. 3.2 and 3.1). In RM-MeTiS, we adopt a direct K-way partitioning scheme instead of recursive bisection scheme of original MeTiS because of the following two reasons. First, it is harder to maintain the mapping constraint during the recursive bisection steps. Second, the intrinsic features of coarsest graph can be efficiently exploited in direct K-way partitioning as described below.

The most successful initial partitioning algorithm in PMeTiS is reported to be the greedy graph growing algorithm (GGGP). GGGP algorithm starts from an initial bipartition where a randomly selected node and all the remaining nodes constitute the bipartitioning. The former and later parts are referred here as *growing* and *shrinking* parts. Then, boundary nodes of the shrinking part are moved to the growing part according to their FM gains until the balance criteria becomes satisfied. The FM gain of a node move is defined as the decrease in the cutsize if the move is realized. A positive gain means a decrease, whereas a negative gain means an increase in the cutsize. A node in a part is said to be a boundary node if it is incident to at least one cut-edge in the partition.

The performance of GGGP algorithm is sensitive to the choice of the initial node. The extension of GGGP algorithm to K-way partitioning requires the selection of $K-1$ such nodes for the initial growing parts, which may degrade the performance in the general partitioning case. Fortunately, the property that $\tilde{\mathcal{G}}_m$ contains exactly K p-supernodes and $|\check{\mathcal{N}}_m| - K$ t-supernodes can

be considered as inducing a $(K+1)$-way initial partition of $\dot{\mathcal{G}}_m$ so that each p-supernode constitutes a part (p-part), whereas all t-supernodes constitute a single part (t-part). It is clear that K p-parts are the inherent and natural candidates for $K-1$ initial growing parts, so that the remaining p-part merged with the t-part will constitute the shrinking part. So, the problem is the selection of a good shrinking part, which is in fact finding a good p-part assignment for all the t-supernodes. In the current implementation of RM-MeTiS, all the t-supernodes are assigned to most attractive p-part. The attraction of a p-part is defined as the sum of all pt-edges between the respective p-supernode and all the t-supernodes.

As in GGGP algorithm for bisection, t-supernodes in the shrinking part are moved to growing parts according to their FM move gains. Each t-supernode in the shrinking part is associated with $K-1$ moves, since it can move to $K-1$ different growing parts. So, $K-1$ FM move gains for each boundary node are computed and that node is inserted into a priority queue (implemented as a max-heap) according to its maximum move gain. At each step of the algorithm, the move with the maximum gain is realized if the size of the destination growing part does not exceed the maximum part size after the move. If that move violates the maximum part size constraint on the destination part, the new maximum move gain of the respective t-supernode to the remaining shrinking parts is computed and re-inserted into the priority queue. If it is detected that all the moves associated with the t-supernode violate the maximum part weight constraint, then this t-supernode is not considered for further moves and it remains assigned to the shrinking part. After each move, the maximum move gains of the t-supernodes in the shrinking part which are adjacent to the moved t-supernode are updated accordingly. These moves are realized even if the corresponding gains are negative until the size of the shrinking part drops below the maximum part size. Then, only moves with positive gains are permitted, so that the algorithm terminates either if a move with negative gain is encountered or the weight of the shrinking part decreases below the minimum part size. Note that the minimum and maximum part sizes are determined by the balance criterion (Eq. 3.1).

5.2.3 Uncoarsening Phase

At each level ℓ (for $\ell = m, m-1, \ldots, 1$), the K-way partition $\tilde{\Pi}_\ell$ found on $\tilde{\mathcal{G}}_\ell$ is projected back to a K-way partition $\tilde{\Pi}_{\ell-1}$ on $\tilde{\mathcal{G}}_{\ell-1}$. The constituent nodes of each supernode of $\tilde{\mathcal{G}}_\ell$ is assigned to the part of their supernode. Obviously, this new K-way partition $\tilde{\Pi}_{\ell-1}$ has the same cutsize with the previous partition $\tilde{\Pi}_\ell$. As the finer graph $\tilde{\mathcal{G}}_{\ell-1}$ is more flexible and has more freedom of possible node moves, the initial partition $\tilde{\Pi}_{\ell-1}$ is refined using an iterative improvement heuristic based on node moves. Among the two refinement schemes, global Kernighan-Lin refinement (GKLR) algorithm is used and modified in RM-MeTiS.

GKLR algorithm can be considered as a simplified version of the general K-way FM partitioning algorithm. An FM-based algorithm iterates a number of passes over the nodes of the graph until a locally minimum partition is found. All nodes are *unlocked* at the beginning of each pass. At each step in a pass, the move with the maximum gain which does not disturb the balance criterion is tentatively performed, and the node associated with the move is locked. The locking mechanism enforces each node to be moved at most once during a pass. At the end of a pass, a maximum prefix subsequence of moves with the maximum prefix sum is constructed from the sequence of tentative node moves and their respective gains. Then, the moves in this maximum prefix subsequence are realized, and the next pass starts from this resulting partition if the maximum prefix sum is positive. The partitioning process terminates if the maximum prefix sum is not positive, i.e. no further decrease in the cutsize is possible.

The locking mechanism employed in GKLR algorithm is efficiently exploited in RM-MeTiS to maintain the feasibility of all partitions obtained during the uncoarsening phase by simply keeping the p-supernodes always in the locked-state. As the partition $\tilde{\Pi}_m$ obtained for the coarsest graph $\tilde{\mathcal{G}}_m$ is a feasible partition, this simple locking mechanism on the p-supernodes maintains the feasibility of further refinements by automatically preventing the moves of p-supernodes to the other parts.

Chapter 6

Parallel DVR Algorithm

As mentioned earlier, the remapping process for OS decomposition is a view-dependent operation that has to be performed at the beginning of each visualization instance. It should be considered as a preprocessing overhead for the sake of efficient object-space parallelization of DVR. So, this overhead must be minimized as much as possible to make it affordable. In this work, we perform a view-independent clustering on the cells of the volumetric dataset, to induce more tractable computational graph instances for the following view-dependent remappings. That is, cell-clusters will constitute the atomic tasks for rendering computations instead of individual cells. So, object-space DVR algorithm implemented in this work consists of 4 consecutive phases, namely, *clustering*, *remapping*, *local rendering* and *pixel merging* phases. Note that the clustering phase is executed only once before the first visualization instance, so that the last three phases are effectively executed for the successive visualization instances. The details of these computational phases are described in the following sections.

6.1 Clustering Phase

This is a *view-independent* preprocessing phase for the sake of efficiency of the following parallel visualizations. In this phase, graph partitioning is used as

a top-down clustering algorithm to decompose the volumetric dataset $\mathcal{V}$ into a set $\mathcal{C}$ of disjoint $|\mathcal{C}| = \alpha$ subvolumes (cell clusters) where $K \ll \alpha \ll |\mathcal{V}|$. A view-independent visualization graph $\mathcal{G}_{\mathcal{V}} = (\mathcal{N}_{\mathcal{V}}, \mathcal{E}_{\mathcal{V}}, w_{\mathcal{V}})$ is constructed as an effort towards estimating the computational structure for the visualization of the volumetric dataset $\mathcal{V}$. As in the view-dependent visualization graph $\mathcal{G}^v_{\mathcal{V}}$, the nodes of $\mathcal{G}_{\mathcal{V}}$ represent the cells of $\mathcal{V}$, and there exist an edge between two nodes if the respective cells share a face. Each node is assigned a weight proportional to the volume of the respective cell. The rationale is that the number of samples to be taken in a cell is linearly proportional to its volume as will be mentioned in Section 6.2.1. Furthermore, more rays are likely to hit a cell with larger volume. Thus, the volume (in WSC) of a cell approximates a view-independent load for the cell, relative to other cells. Similarly, more rays are likely to intersect a face with larger area, so that the area of a face approximates a view-independent amount of interaction between the pair of cells sharing that face. Hence, each edge of $\mathcal{G}_{\mathcal{V}}$ is weighted with the area (in WSC) of the respective face. The α-way partitioning of $\mathcal{G}_{\mathcal{V}}$ will serve our purpose for view-independent α-way clustering. The total number of clusters, α, is determined empirically as will be discussed in Section 7.1.

As mentioned earlier, the volumetric dataset which is assumed to be the result of an engineering simulation is already distributed among the processors because of the parallel execution of the respective application. The simulation graph $\mathcal{G}_{\mathcal{S}}$ representing the computational structure of computational fluid dynamics type applications usually differs from the view-independent visualization graph $\mathcal{G}_{\mathcal{V}}$ in only node and edge weights. In such applications, both computational costs of cells and amounts of interactions between neighbor cells through shared faces are mostly identical, which means unit node and edge weights in $\mathcal{G}_{\mathcal{S}}$. Hence, without loss of generality we can assume that the subvolumes corresponding to the parts of a K-way partition of $\mathcal{G}_{\mathcal{S}}$ and hence $\mathcal{G}_{\mathcal{V}}$, reside in K processors of the parallel machine. Since the volumetric data is already distributed, construction of the global view-independent visualization graph $\mathcal{G}_{\mathcal{V}}$ for global clustering may be very expensive. In this work, instead of global clustering, we adopt a parallel local clustering scheme to minimize the view-independent preprocessing overhead.

In the local clustering scheme, each processor P_k concurrently constructs its local view-independent visualization graph $\mathcal{G}_{\mathcal{V}_k}$. The local graph $\mathcal{G}_{\mathcal{V}_k}$ of processor P_k is in fact the subgraph of the global graph $\mathcal{G}_{\mathcal{V}}$ induced by the node subset corresponding to the local cells of P_k. Then, each processor P_k concurrently performs α_k-way clustering of its local graph $\mathcal{G}_{\mathcal{V}_k}$ through α_k-way partitioning of $\mathcal{G}_{\mathcal{V}_k}$ by using MeTiS. The local number of resulting clusters α_k in each processor P_k is selected to be proportional to the total node weight in its local graph $\mathcal{G}_{\mathcal{V}_k}$ such that $\sum_{k=1}^{K} \alpha_k = \alpha$. This scheme is an effort towards reducing the variations in cluster weights for the sake of the efficiency of graph-partitioning based remapping. Note that α_k-way partitioning performed in each processor P_k, for $k = 1, 2, \ldots, K$, effectively constitute a α-way global clustering solution $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_\alpha\}$.

After the local clustering operations, the following local and global data structures are constructed. Each cluster in the overall environment is given a unique global cluster identifier and the global mapping function $\mathcal{M}$ for these clusters is determined. As each cell-cluster induces a subvolume to be rendered simultaneously and independently, the tetrahedral cell data associated with each cell-cluster is maintained separately. That is, each processors maintains a *ClusterData* structure for each of its local cell-clusters. Two major components of a *ClusterData* structure are the *VertexArray* and *CellArray* structures of the respective cell-cluster. Local cell and vertex indexing is utilized within *CellArray* and *VertexArray* structures for each cell-cluster. In order to maintain the connectivity information between cell-clusters within this local indexing scheme, four global neighbor-cell identifiers of each cell in the *CellArray* structure of a cell-cluster are replaced by the global-cluster identifiers and local indices of the respective neighbor cells. *ClusterData* structure also maintains the total volume (in WSC) of the respective local cell-cluster, which is computed as the sum of the volumes of the constituent cells of the cell-cluster. This view-independent volume information will be utilized in estimating the view-dependent sampling computational weights of the cell-clusters during the remapping phase as will be described in Section 6.2.1. Each processor also maintains a global *ClusterMapArray* which holds two components for each cell-cluster in the overall environment. The first component effectively

keeps the current mapping of the cluster (i.e., $\mathcal{M}_\mathcal{C}$ function), and the second component keeps a pointer to the *ClusterData* structure of the respective cell-cluster if it is a local cell-cluster in that processor.

The final step of this phase is the construction of the view-independent portions of the remapping tuple $(\mathcal{G}_\mathcal{C}^v, \mathcal{M}_\mathcal{C}, \mathcal{T}_\mathcal{C})$. The idea behind this step is to reduce the overhead of remapping phase for each visualization instance. The graph $\mathcal{G}_\mathcal{C}^v = (\mathcal{N}_\mathcal{C}, \mathcal{E}_\mathcal{C}, w_\mathcal{C}^v)$, referred to here as the contracted visualization graph, is the contraction of $\mathcal{G}_\mathcal{V}^v$ according to the global clustering solution $\mathcal{C}$ for a given viewing parameter v. That is, $\mathcal{N}_\mathcal{C} = \mathcal{C}$ and there exists an edge e_{ij} between the clusters $\mathcal{C}_i$ and $\mathcal{C}_j$ if these clusters share at least one face through two neighbor cells in these two clusters. Note that the topology of the view-dependent visualization graph $\mathcal{G}_\mathcal{C}^v$ does not change with changing viewing parameter v, and moreover it is identical to the topology of the view-independent contracted visualization graph $\mathcal{G}_\mathcal{C}$. All of these contracted graphs differ in only node and edge weights. Hence, each processor constructs its local portion of $\mathcal{G}_\mathcal{C}$, and then a copy of the global graph is replicated in each processor by performing an all-to-all-broadcast operation. In this way, the construction of the view-dependent contracted graph $\mathcal{G}_\mathcal{C}^v$ will only involve the computation of node and edge weights according to the given viewing parameter v.

Another view-independent portion of the remapping tuple is $\mathcal{T}_\mathcal{C}$ which denotes the cluster-migration-cost function for the cluster migration costs. The migration cost of a cell-cluster is taken to be equal to the ratio of the size of its associated *ClusterData* structure to the size of an individual ray-segment information. Hence, it is clear that cluster migration costs do not change with changing viewing parameter. Since each processor only knows the migration costs of its local clusters, an all-to-all broadcast operation is performed to let each processor to become aware of the migration costs of all clusters.

6.2 Remapping Phase

This phase consists of three steps, namely, *graph update* (GU), *graph partitioning* (GP) and *cluster migration* (CM) steps. In the GU step, the remapping

graph $\dot{\mathcal{G}}_{\mathcal{C}}^{v}$ is updated in parallel. In the GP step, the remapping graph is partitioned by using RM-MeTiS. Finally in the CM step, clusters are migrated according to the remapping solution. Note that the GU and GP steps constitute the view-dependent preprocessing overhead for parallelization.

The following definitions are given here for the sake of simplicity of the presentation. A face of a cell of a cluster in a clustering $\mathcal{C}$ is defined to be a *pseudo-boundary* face if it is shared by two cells belonging to two different clusters. A pseudo-boundary faces may become a *boundary* face after remapping if it is shared by two cells belonging to two different clusters which are mapped to different processors.

Figure 6.1 displays a 2-way mapping of a sample 7-way clustered dataset, where shaded and unshaded clusters (hence cells) are assigned to processors P_1 and P_2, respectively. Note that each triangle represents a tetrahedral cell and each edge of a triangle represents the respective face of the cell because of the 2D representation of the sample volumetric dataset. In this figure, each solid edge shared by two triangles represent a pseudo-boundary face, and each bold edge shared by two triangles represent a boundary face. Note that each dotted edge represents an internal face of a cluster, and each bold edge that is not shared by two triangles represent an external cell.

6.2.1 Graph Update (GU) Step

The major computation in the GU step is to calculate the view-dependent portions of the remapping tuple $(\mathcal{G}_{\mathcal{C}}^{v}, \mathcal{M}_{\mathcal{C}}, \mathcal{T}_{\mathcal{C}})$ for the new viewing parameter v. The view-dependent contracted visualization graph $\mathcal{G}_{\mathcal{C}}^{v}$ is easily constructed by recomputing the node and edge weights of the previous graph $\mathcal{G}_{\mathcal{C}}^{v_p}$ where v_p denotes the previous viewing parameter. Note that for the first visualization instance, $\mathcal{G}_{\mathcal{C}}^{v_p}$ graph will be effectively the view-independent contracted visualization graph $\mathcal{G}_{\mathcal{C}}$. After each processor concurrently computes the local portion of the weighted visualization graph $\mathcal{G}_{\mathcal{C}}^{v}$, it performs the necessary local augmentations to construct the local portion of the remapping graph $\dot{\mathcal{G}}_{\mathcal{C}}^{v}$ using the cluster-migration cost function $\mathcal{T}_{\mathcal{C}}$ and the current mapping function $\mathcal{M}_{\mathcal{C}}$.

Every processor already knows the mapping function $M_{\mathcal{C}}$, since CIA structure is maintained up to date after the remapping phase of each visualization instance. Then, an all-to-all broadcast operation is performed on the local portions of the remapping graph $\tilde{\mathcal{G}}_{\mathcal{C}}^{v}$, so that each processor gathers a copy of the global $\tilde{\mathcal{G}}_{\mathcal{C}}^{v}$.

It should be noted here that node and edge weights of the $\mathcal{G}_{\mathcal{C}}^{v}$ graphs may not always change during successive visualizations. It is clear that node and edge weights do not dependent on the transfer function used for the mapping of scalar values to color and opacity values. Although node and edge weights of graph $\mathcal{G}_{\mathcal{C}}^{v}$ change with changing image size, the relative node and edge weights remain the same, respectively, as it will become clear in the following sections. Hence, the remapping phase should be skipped in such cases.

Node Weight Estimation

The rendering cost of a cluster $\mathcal{C}_i$ for a viewing parameter v can be estimated using Eq. 2.7 as,

$$w(n_i) = I_{\mathcal{C}_i}^{v} t_I + S_{\mathcal{C}_i}^{v} t_S + R_{\mathcal{C}_i}^{v} t_R. \tag{6.1}$$

for recomputing the weight of node n_i of $\mathcal{G}_{\mathcal{C}}^{v}$. Since, the unit costs t_R, t_I and t_S are determined as mentioned in Section 2.4, the computational load estimation reduces to the efficient estimation of $I_{\mathcal{C}_i}^{v}$, $S_{\mathcal{C}_i}^{v}$ and $R_{\mathcal{C}_i}^{v}$ counts associated with the rendering of cluster $\mathcal{C}_i$ for the given viewing parameter v.

The number of ray-face intersections, $I_{\mathcal{C}_i}$, can be calculated by summing the number of pixels covered by the projection areas of the bf-faces of the cells in the cluster $\mathcal{C}_i$. But this exact computation scheme requires the scan-conversion of bf-faces which is a costly operation. Instead, we approximate the pixel-coverage count of a face f, by its projection area a_f in NPC which is simply computed as

$$a_f = x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2), \tag{6.2}$$

where x_i and y_i denote the x and y coordinates (in NPC) of the ith vertex of triangular face f, respectively. Our implementation exploits the fact that bf-internal face of a cell is in fact an ff-internal face of a neighbor cell in order

to compute the projection area of each internal face only once without checking whether the face is a bf or ff face which necessitates relatively expensive face orientation calculations. The per face errors introduced because of the discretization of the projection screen do not reflect to the total estimate for $I_{\mathcal{C}_i}$ because of the summation of the floating-point area values of the faces with contiguous projection areas.

The number of samples taken $S_{\mathcal{C}_i}$ can be estimated by calculating the volume of the cluster in NPC. The volume of the cluster in NPC denotes the number of unit cubes in that volume. The number of samples to be taken in each unit cube in NPC is equal to $1/\Delta z$ where Δz is the distance between the successive sampling points along a ray. Hence, the number of samples to be taken in the cluster $\mathcal{C}_i$ can be estimated by multiplying the volume of the cluster in NPC by $1/\Delta z$. The estimation error because of the discretization of resampling volume will be just on the boundary surface of the cluster. As NPC system is dependent to the viewing parameter v, a straightforward implementation of this scheme will require the computation and summation of the volumes (in NPC) of the individual cells of the cluster for each visualization instance. However, we adopt a much more efficient scheme which exploits the idea that there exist a constant scaling between any volume in the view-independent WSC system and the view-dependent NPC system for a given viewing parameter v in parallel projection. This scale factor γ can easily be determined by computing the volume of a unit cube of WSC in NPC. As the volume of each cluster in WSC is already computed (during the clustering phase) and stored in its *ClusterData* structure, estimation of the number of samples associated with a cluster reduces to multiplying its volume by the ratio $\gamma/\Delta z$.

The number of ray-segments $R_{\mathcal{C}_i}$ can be approximated by the sum of the projection areas of the ff-external and ff-pseudo-boundary faces of $\mathcal{C}_i$ using Eq. 6.2. Note that $R_{\mathcal{C}_i}$ denotes the number of ray-segments that need to be generated for the independent and simultaneous rendering of cluster $\mathcal{C}_i$. However, as will be described in Section 6.3, our local rendering scheme considers the whole set of local clusters of a processor as a single subvolume during rendering, so that ray-segments are generated only for the ff-boundary faces of

the subvolume. In this way, unnecessary ray-segment generation operations are avoided for the pseudo-boundary faces that effectively become the internal faces of the subvolume. As the ray-segment generation cost for a cluster depends on the partitioning of the remapping graph, incorporation of this cost to RM-MeTiS is comparably hard although not impossible. Furthermore, as mentioned earlier, the cutsize minimization during partitioning the remapping graph also fulfills the objective of minimizing the number of ray-segment generations due to the boundary faces. Hence, the ray-segment generation costs are not considered in node weighting of the contracted visualization graph $\mathcal{G}_{\mathcal{C}}^{v}$.

Edge Weight Estimation

The weight $w(n_i, n_j)$ of edge e_{ij}, which will indicate the amount of interaction between clusters $\mathcal{C}_i$ and $\mathcal{C}_j$, can be computed as follows. Each face shared by two neighbor cells in clusters $\mathcal{C}_i$ and $\mathcal{C}_j$ contributes its pixel-coverage count to the edge-weight $w(n_i, n_j)$. The pixel-coverage counts of these pseudo-boundary faces between $\mathcal{C}_i$ and $\mathcal{C}_j$ are approximated by the projection areas of these faces which are computed using Eq. 6.2. The source of errors in this approximation is due to the discretization of the projection screen similar to that of ray-face intersection estimation. If clusters $\mathcal{C}_i$ and $\mathcal{C}_j$ are remapped to different processors, then these pseudo-boundary faces will become boundary faces of the local subvolumes of the respective processors, thus incurring redundant computation overhead due to ray-segment generation and communication overhead during pixel merging. Both types of overheads are proportional to the edge-weight $w(n_i, n_j)$.

6.2.2 Graph Partitioning (GP) Step

In this step, all processors concurrently execute RM-MeTiS for K-way partitioning of their copies of the remapping graph $\mathcal{G}_{\mathcal{C}}^{v}$ using different random seeds. Then, the best partition is determined by performing a global-minimum operation on the cutsize values of the local partition solutions. Then, the processor which produced the best solution broadcasts its partition vector which

corresponds to the new mapping function $\grave{\mathcal{M}}_{\mathcal{C}}$. This scheme is an effort towards avoiding the worst case behavior of RM-MeTiS which is a randomized algorithm.

6.2.3 Cluster Migration (CM) Step

In this step, those clusters whose new mappings differ from the current mappings migrate to their new *home* processors through personalized communication of the respective *ClusterData* structures. Each processors also updates its *ClusterMapArray* structure according to the new mapping.

6.3 Local Rendering Phase

In graph partitioning, minimization of the cutsize is equivalent to the maximization of the sum of the weights of the internal edges, where an edge in a partition is said to be an internal edge if its pair of nodes reside in the same part. Note that an internal edge in a partition of the remapping graph represents the pseudo-boundary faces which are shared between two local clusters of a processor. Hence, after the remapping phase, each processor is very likely to contain a set of highly interacting clusters for local rendering. Thus, rendering local clusters independently by considering them as individual subvolumes will incur substantial amount of redundant computations because of the ray-segments to be generated for the ff-pseudo-boundary faces shared between local clusters. For that reason, our implementation considers the set of local clusters of each processor as a single local subvolume to be rendered by the Koyamada's sequential algorithm.

The local rendering begins by each processor concurrently traversing the cells of the *CellArray* structures of its local clusters. Each processor P_k checks each face f of each cell c_i of each of its local cluster C_p whether it is an external or a boundary face. The face f is identified as an external face if cell c_i does not have a neighbor cell through face f. The face f is identified as a boundary face if cell c_i is neighbor to a cell c_j in cluster C_q through face f such that C_q

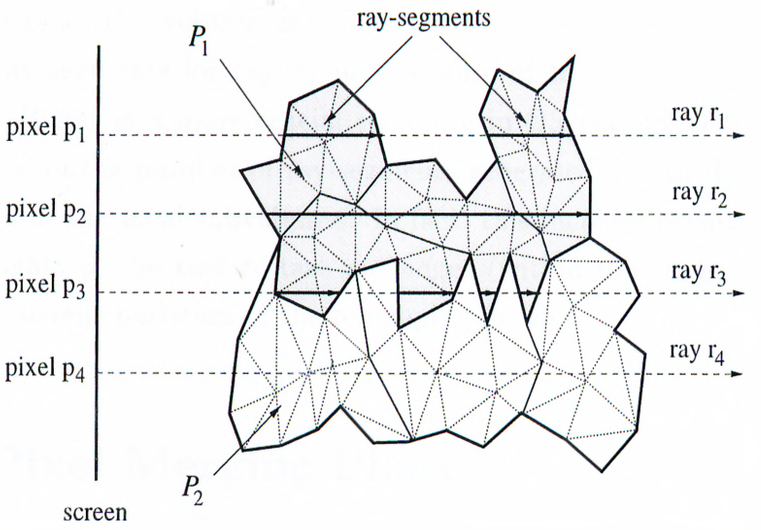


Figure 6.1: Local rendering of shaded subvolume by processor P_1.

is a non-local cluster of P_k, i.e., $\mathcal{M}_{\mathcal{C}}(C_p) \neq \mathcal{M}_{\mathcal{C}}(C_q)$. Each of the external and boundary faces is scan-converted and a ray-segment is generated for each pixel covered by the face. Each ray-segment generated is followed through the local subvolume according to Koyamada's algorithm until it exits from a bf-external face or a bf-boundary face. Note that the efficient ray traversal scheme of Koyamada's algorithm is not disturbed even if two successive cells on the route of a ray belong to two different local clusters, because the *ClusterData* structures preserve the connectivity information between the cells despite clustering. For example, in Figure 6.1, ray r_2, which passes through 4 local clusters, is processed by processor P_1 as a single ray-segment as in the sequential algorithm. As a ray-segment exits the local subvolume, its composited color and opacity values are inserted into the linked-list of the respective pixel location of the *RayBuffer* structure in sorted order accordingly to its exit z value.

Recall that *RayBuffer* structure is needed in our implementation of Koyamada's sequential algorithm, because of the possibility of multiple ray-segment generations per pixel which may incur in non-convex volumetric datasets. However, even if the dataset is convex, the OS decomposition may generate local subvolumes with virtual non-convexities, despite the fact that the cutsize minimization during the partitioning of the remapping graph involves an explicit effort towards the minimization of such virtual non-convexities. For example, in Figure 6.1, processor P_1 generates 2 ray-segments for ray r_1 because of the

nonconvexity of the volume as in the sequential algorithm, whereas it generates 4 ray-segments for ray r_3 just because of the virtual non-convexities. Thus, *RayBuffer* is a more crucial structure in OS parallel DVR, because of the increase in the number of ray-segments generated due to the virtual non-convexities in the local subvolumes. In fact, this increase is equal to the sum of the weights of the task-to-task cut edges (within the range of estimation errors) in current partition of the remapping graph.

6.4 Pixel Merging Phase

It is clear that the projection areas of the local subvolumes of processors may overlap on the image-screen, despite the explicit minimization effort during the partitioning of the remapping graph. Thus, composited ray-segments residing in the local *RayBuffer* structures of different processors may contribute to the same pixel location of the screen. Thus, a global merge operation is needed on the ray-segment lists of the local *RayBuffer* structures for compositing the ray-segments of each pixel location in visibility order to obtain the final image. For example, in Figure 6.1, 4 and 5 ray-segments generated and composited by processors P_1 and P_2 for pixel location p_3, respectively, should be gathered and composited to determine the final color of pixel p_3.

In the parallel pixel merging phase, the screen is partitioned into K subregions such that each processor is held responsible for producing the final image of a distinct subregion through local pixel merging. The objective in the screen decomposition and assignment is to minimize the communication overhead due to ray-segment migration while maintaining the load balance during the following local pixel merging step. Hence, this phase consists of 3 steps, namely, *pixel assignment* (PA), *ray-segment migration* (RSM), and *local pixel merging* (LPM).

6.4.1 Pixel Assignment (PA) Step

The problem in this step can be considered as the independent task assignment problem where merging of the ray-segments belonging to individual pixels constitute the independent atomic tasks to be assigned to processors. The assignment operation is a preprocessing step for the sake of efficient parallelization of pixel merging phase. However, the overhead of this step must be kept minimal because of the fine granularity of the pixel merging operations. For that reason, a 2D coarse mesh is imposed on the 2D image-screen. The elements of this mesh are called *mesh-cells*. This coarse mesh is selected such that the shapes of the individual mesh-cells are close to square as much as possible for the sake of scalability of the decomposition. The merging of the ray-segments belonging to the pixels in a mesh-cell is considered as an atomic task to be performed by a single processor. Hence, the problem reduces to the assignment of mesh-cells to the processors.

The following definitions are given here for the sake of the clarity of the presentation. For a mesh-cell m, let $\Psi_m \subseteq \Psi = \{P_1, P_2, \ldots, P_K\}$ denote the subset of processors which generated ray-segments for the pixels in the mesh-cell m during the local rendering phase. The local ray-segment count r_{km} of a processor P_k for a mesh-cell m denotes the total number of local ray-segments generated by P_k for m. The global ray-segment count R_m of a mesh-cell m denotes the total number of ray-segments produced by all processors for the mesh-cell m, i.e., $R_m = \sum_{P_k \in \Psi_m} r_{km}$. The computational load of a mesh-cell m during the LPM step is assumed to be proportional to its global ray-segment count R_m.

In this work, we propose three simple yet effective schemes for the assignment of the mesh-cells, namely, *scattered* assignment (SCA), *minimum-communication* assignment (MCA), and *balanced-load minimized-communication* assignment (BLMCA) schemes.

Scattered Assignment (SCA)

In this scheme, the mesh-cells are assigned to the processors in a scattered fashion for load balancing in the LPM step. The performance of this scheme depends on the assumption that the neighbor mesh-cells are likely to have equal work load since they are likely to have similar views of the volume. The main advantage of this scheme is its simplicity, and therefore the speed of the task assignment process. As each processor concurrently determines the mesh-cell to processor assignment without communication, this scheme involves negligible computation and no communication overhead.

In SCA scheme, the total volume of communication due to the ray-segment migration is not heavily dependent on the mesh size, so increasing the mesh size is more likely to obtain a better load balance, and thus it can be expected to yield a better assignment. However, the major deficiency of this scheme is the lack of the explicit minimization of the communication overhead for RSM step.

Minimum-Communication Assignment (MCA)

In the MCA scheme, it is assumed that the bottleneck in parallel pixel merging is the RSM step rather than the LPM step, so the total volume of communication for the RSM step is minimized while totally ignoring the load balancing for the LPM step. The MCA scheme exploits the following simple observation for the minimization of the communication overhead. Assigning the mesh-cell m to a processor P_k will incur the migration of the respective ray-segments from each processor $P_\ell \in (\Psi_m - \{P_k\})$ to P_k, while avoiding the migration of the respective local ray-segments of processor P_k. That is, this assignment will incur a communication volume of $R_m - r_{km}$. Hence, the greedy assignment of each mesh-cell m to the processor $P_q \in \Psi_m$, which has the maximum local ray-segment count for the mesh-cell m, will produce an assignment with globally minimum volume of communication.

The MCA is performed in parallel as follows. Each processor concurrently traverses its local *RayBuffer* to construct a local workload (WL) matrix, whose

size is equal to coarse mesh size such that each entry of WL contains the local ray-segment counts for respective mesh-cell. Then, a global-maximum operation is performed on the local WL matrices through a fold and expand communication step, so that each processor determines the index of the processor which has the maximum local ray-segment count for each mesh-cell. That is, at the end of this global communication step, each processor holds the processor assignment for all mesh-cells.

Balanced-Load Minimized-Communication Assignment (BLMCA)

This scheme considers both the volume of communication in the RSM step and the load imbalance in the LPM step for minimization. The MCA scheme constitutes an optimal solution to the former objective, whereas the latter one corresponds to the K-way number-partitioning problem which is NP-hard. The numbers in the number-partitioning problem correspond to the global ray-segment counts of the mesh-cells. The concept of best-fit decreasing BFD heuristic devised for bin-packing problem can be also effectively used for the solution of the number partitioning problem, and hence the pure load balancing problem. In the BFD-based load balancing, the mesh-cells are considered for assignment in decreasing sorted order of their global ray-segment counts, the best-fit criteria is the assignment of the mesh-cells to the minimally loaded bins (processors) and the bin (processor) capacity is the maximum allowable part size according to the load balance criterion 3.1.

The proposed BLMCA scheme incorporates the greedy assignment criterion of the MCA scheme to the BFD-based load balancing heuristic through replacing its best-fit criterion by the criterion of MCA scheme for feasible assignments. The proposed heuristic starts with initializing the work-loads of all processors to zero. Then, a mesh-cell m, considered in the sorted order, is assigned to processor P_k with the maximum local ray-segment count r_{km} for m, if the current load of the processor will remain below the maximum part size after the assignment. Otherwise, the mesh-cell m is assigned to the minimally loaded processor where this assignment criteria corresponds to the best-fit criteria used in BFD-based pure load balancing heuristic. After the

assignment, the load of the respective processor is incremented by R_m.

The nice property of BLMCA heuristic is that the maximum amount of communication volume that can be avoided by the assignment of a heavily loaded mesh-cell is likely to be larger than that of a lightly loaded mesh-cell. Hence, delaying the assignment of lightly loaded mesh-cells minimizes the deviation from both the minimal communication cost to be found by the MCA algorithm and the load balance quality to be found by the pure BFD-based load balancing heuristic.

The parallelization of BLMCA scheme is similar to that of the MCA scheme with the following enhancements. A global-sum operation is carried out together with the global-max operation on the local WL matrices. At the end of the fold and expand communication step, each processor knows the the global ray-segment count and the index of the processor with maximum local ray-segment count for each mesh-cell. Then, each processor determines the processor assignment for all mesh-cells by running the BLMCA algorithm after sorting the mesh-cells according to their global ray-segment counts.

6.4.2 Ray-segment Migration (RSM) Step

In this step, the local ray-segments migrate according to the mesh-cell to processor assignment found in the PA step. Each processor concurrently traverses its local *RayBuffer* structure to packetize and gather the ray-segment lists belonging to the mesh-cells assigned to other processors to the respective send-buffers. Then, these buffers are sent to the respective processors through point-to-point communication. At the end of this step, each processor gathers all the ray-segments belonging to the pixels of the mesh-cells assigned to itself. Note that, task-to-task cut-edge component of the cutsize of the current partition of the remapping graph constitutes an upper bound on the total number of ray-segments communicated in this step for all pixel-assignment schemes

6.4.3 Local Pixel Merging (LPM) Step

In this step, each processor concurrently composites the gathered ray-segments for each of its assigned pixel in visibility order. This composition operation for each local pixel involves the merging of k sorted ray-segment lists, where k denotes the number of distinct processors that generated ray-segments for that particular pixel. A binary heap is used for the k-way merging. At the end of this step, each processor holds the final color for each pixel belonging to its assigned mesh-cells.

Chapter 7

Experimental Results

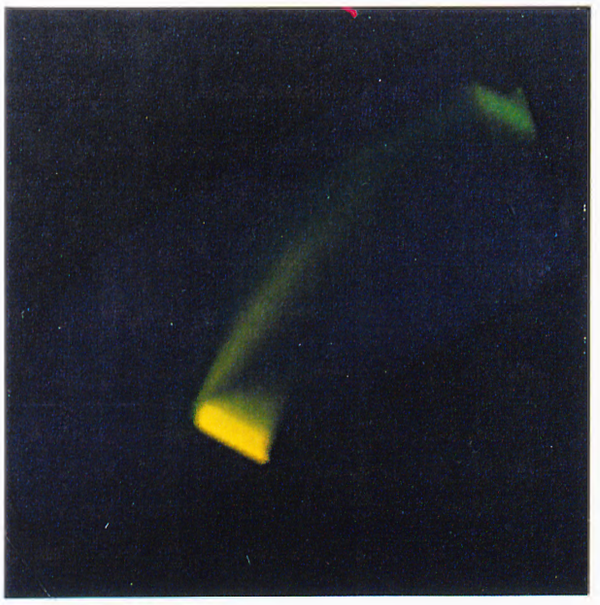
The sequential and parallel DVR algorithms presented in this thesis are implemented on a Parsytec's CC-24 system, which is based on distributed-memory MIMD architecture. Our CC system consists of 24 nodes, and each node is equipped with a 133 MHz PowerPC 604 processors, a 2×16 KB L1 cache and a 512 KB L2 cache. The system has 4 I/O nodes each with 128 MB memory and 2GB local disk. Each of the 20 compute nodes has 64 MB memory and 340 MB local disk. The interconnection network consists of sparsely connected six 8×8 crossbar switching boards such that each switching boards connects 4 processors. The network can sustain 20 MB/s point-to-point communication bandwidth [30]. Embedded Parix (EPX) [31] library, which is the native message passing library of Parsytec, is used for message passing. The algorithms were implemented using C language.

The volumetric datasets used for experimentation are *Blunt Fin* (BF), *Combustion Chamber* (CC) and *Oxygen Post* (OP) which are obtained from NASA-Ames Research Center. These datasets are commonly used by researchers in volume rendering field. All datasets are originally curvilinear in structure, and they represent the results of CFD simulations. The raw datasets consist of hexahedral cells, and they are converted into unstructured tetrahedral data format by dividing each hexahedral cell into 5 tetrahedral cells [20, 32]. Table 7.1 summarizes the properties of the datasets used in this work. Note that datasets are listed in increasing order of size, both in terms of number of cells and vertices

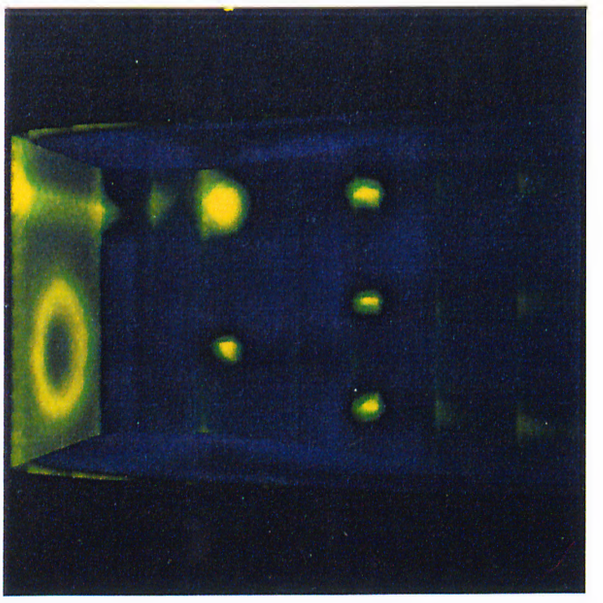
(sampling points). This order is maintained in all the following tables for a better and simpler understanding of performance analysis. In Table 7.1, each COV value represents the *coefficient of variation* of the cell sizes in terms of cell volumes of the respective dataset. COV value of a dataset is considered as an indication for the level of irregularity of the dataset. Figure 7.1 displays the rendered images of the datasets.

As these datasets represent CFD simulation results, it is assumed that the respective parallel simulations were performed on our parallel machine on which the visualization will be carried out. The parallel CFD simulation on K processors is modeled by decomposing each volumetric dataset into K subvolumes through partitioning its associated simulation graph $\mathcal{G}_{\mathcal{S}}$ using MeTiS, and assigning the data structures associated with each subvolume to a distinct processor. As mentioned in Section 6.1, $\mathcal{G}_{\mathcal{S}}$ has unit node and edge weights and its topology is identical to the topology of the respective view-independent visualization graph $\mathcal{G}_{\mathcal{V}}$. So, in all experimentations, the execution of the proposed 4-phase parallel DVR algorithm on K processors starts from such initial mappings. Note that for rendering successive visualization instances, the mapping for the current instance will constitute the initial mapping for the next instance. Although our parallel DVR implementation is capable of performing successive visualization instances without any modification, experimental results are given for only the first visualization instances for the sake of reproducibility of the experiments. Recall that the clustering phase is a view-independent preprocessing which is performed only once for successive visualizations, and it becomes negligible after only a few successive visualizations as will be experimentally verified later. Hence, here and hereafter, parallel rendering time T_{par} for a visualization instance will refer to the view-dependent parallel execution time, which involves remapping, local rendering and pixel merging phases.

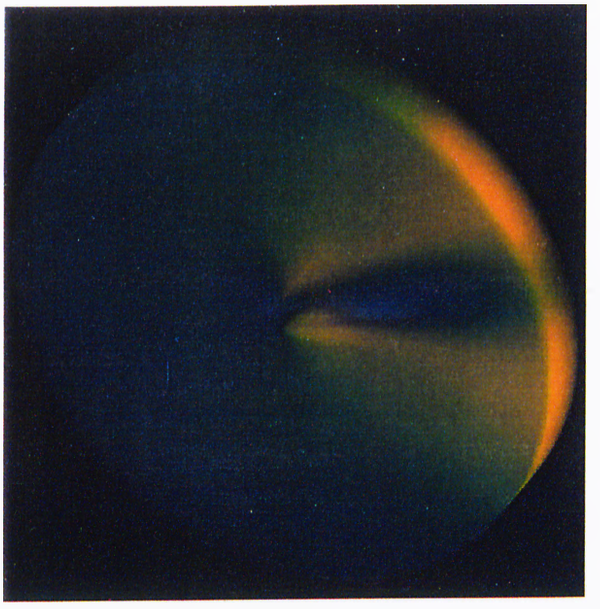
Four different viewing directions, which include the standard viewing direction, are used in the experimentations to display average case performances. In the standard view, the dataset is viewed along the z-axis in the positive z direction in WSC. The other three views are selected such that different views of the datasets are rendered as much as possible. Experiments are carried out for three image sizes: 400×400, 600×600 and 900×900. The window on the



(a) Blunt Fin



(b) Combustion Chamber



(c) Oxygen Post

Figure : Rendered images of the volumetric data sets used in performance analysis.

Dataset	#Vertices	#Cells	COV
Blunt Fin (BF)	40,960	187,395	5.50
Combustion Chamber (CC)	47,025	215,040	0.42
Oxygen Post (OP)	109,744	513,375	4.26

Table 7.1: List of volumetric datasets used for experimentation.

view plane is selected such that all the volume is visible through the window leaving a thin margin between the borders of the window and the projected volume.

As RM-MeTiS used in the remapping phase is a randomized algorithm, each parallel DVR algorithm is executed 10 times for each parallel visualization instance and the average values are used for displaying in the following tables.

7.1 Clustering Phase

Table 7.2 illustrates the results of the experiments performed for determining the appropriate values for the total number of clusters α to be generated in the clustering phase. In this work, the value α to be used for a parallel visualization instance on K processors is determined through $\alpha = K\alpha_{avg}(K)$, where $\alpha_{avg}(K)$ is an empirically found parameter. Table 7.2 shows the variation of the average performance of the parallel DVR algorithm with varying α_{avg}, where averaging is over rendering all datasets for the standard viewing direction and the medium image size 600×600. In this table, T_{par} and T_{pre} denote the parallel rendering and view-dependent preprocessing times, respectively, normalized with respect to those for the $\alpha_{avg} = 25$ case, where T_{pre} involves the graph update (GU) and graph partitioning (GP) steps in the remapping phase.

The performance of the parallel DVR algorithm is expected to increase with increasing α, because of the expected increase in the quality of the partitions to be found by RM-MeTiS due to the increase in the size of the solution space. On the contrary, the parallel DVR performance is expected to decrease with increasing α because of the increase in T_{pre}. Hence, the parallel rendering time T_{par} is expected to decrease with increasing α until a turnover value

K		α_{avg}: avg. # of clusters per processor				
		25	50	100	200	300
4	T_{par}	1.000	0.941	0.926	0.920	0.915
	T_{pre}	1.000	1.058	1.144	1.295	1.430
8	T_{par}	1.000	0.978	0.976	0.979	0.981
	T_{pre}	1.000	1.141	1.348	1.741	2.117
16	T_{par}	1.000	0.975	0.973	0.984	0.995
	T_{pre}	1.000	1.260	1.684	2.418	3.074
24	T_{par}	1.000	0.989	1.001	1.028	1.059
	T_{pre}	1.000	1.376	1.943	2.900	3.791

Table 7.2: Variation of normalized average parallel rendering (T_{par}) and view-dependent preprocessing (T_{pre}) times with α_{avg}.

Weighting Scheme		Data Sets					
		BF		CC		OP	
Node	Edge	*LI*	*CS*	*LI*	*CS*	*LI*	*CS*
1	1	2.99	1.58	2.90	1.04	3.61	1.74
V	1	2.81	1.54	2.88	1.05	2.83	1.54
1	A	4.43	1.08	2.90	0.99	2.92	1.19
V	A	2.66	1.00	2.89	1.00	2.75	1.00

Table 7.3: The clustering quality of various weighting schemes.

after which T_{par} will begin to increase. This behavior can easily be seen in Table 7.2. As also seen in the table, α_{avg} values leading to the best parallel performance, decrease with increasing K, as expected. So, $\alpha_{avg} = 300$, 150, 100, 75, 60 and 50 are selected and used in our experiments for $K = 4$, 8, 12, 16, 20 and 24, respectively.

Table 7.3 illustrates the experimental validation of the node and edge weighting schemes used in the view-independent visualization graph $\mathcal{G}_V$ (used in clustering phase). In the table, 1 denotes unit node and edge weighting, V denotes the use of cell volume in node weighting and A denotes the use of the face area in edge weighting. Note that (V, A) tuple denotes our weighting scheme mentioned in Section 6.1. The quality of a weighting scheme is evaluated by the partition quality of the remapping graph $\dot{\mathcal{G}}^v_{\mathcal{C}}$ constructed through clustering $\mathcal{G}_V$ graph weighted according to the respective scheme. The quality of a partition of $\dot{\mathcal{G}}^v_{\mathcal{C}}$ is determined by the respective *cutsize* (CS) and *percent load imbalance* (LI) where LI is computed as $LI = 100 \times (W_{max} - W_{avg})/W_{avg}$. The average partition quality values displayed in the table are obtained through averaging over all viewing directions and all image sizes for $K = 16$ and $\alpha_{avg} = 200$.

Note that CS values are normalized with respect to those of (V, A) weighting scheme.

As seen in Table 7.3, the proposed (V, A) weighting scheme leads to substantially better remappings then the other schemes except for the CC dataset. The naive unit weighting scheme $(1, 1)$ already leads to sufficiently good partition quality for the CC dataset because of the almost equal sized cells which is evident from the extremely low COV value of the CC dataset displayed in Table 7.1. So, the forthcoming discussion is restricted to the BF and OP datasets. As seen in the table, the $(V, 1)$ scheme leads to 54% worse CS values than the (V, A) scheme, while leading to slightly worse LI values. The $(1, A)$ scheme leads to considerably worse partition quality values compared to the (V, A) scheme. The reason behind this experimental finding is that unit node weighting leads to a remapping graph with large node weight variation thus restricting the feasible (Eq. 3.1) solution space for the partitioning of the remapping graph. Thus, these experimental results verify the merits of the proposed node and edge weighting scheme.

7.2 Remapping Phase

Table 7.4 illustrates the performance of the proposed estimation schemes used in node and edge weighting for the view-dependent contracted visualization graph $\mathcal{G}_C^v$. Here, I^{err}, S^{err} and T_{LR}^{err} denote the average percent errors in the estimation of ray-face intersection counts, sampling counts and local rendering times, respectively, of local subvolumes rendered by distinct processors for different parallel visualization instances. Table 7.4 also displays the averages of estimated (LI_e) and measured (LI_m) percent load imbalance values for the same set of parallel visualization instances. The set of parallel visualization instances used for averaging include all viewing directions and processors $K = 4, 8, 12, 16, 20, 24$.

As seen in Table 7.4, estimation errors in both I and S counts are extremely low. As also seen in the table, percent estimation errors in S counts are drastically smaller than those of I counts as expected. Because, in the estimation

Data Set	Image Size	% error in estimates			% load imb.	
		I^{err}	S^{err}	T_{LR}^{err}	LI_e	LI_m
BF	400×400	1.335	0.014	5.031	4.402	6.492
	600×600	1.352	0.007	5.044	4.846	7.042
	900×900	1.357	0.004	5.023	4.699	6.952
CC	400×400	1.397	0.005	5.411	3.107	3.550
	600×600	1.403	0.003	5.199	2.579	3.379
	900×900	1.405	0.002	5.077	2.437	3.543
OP	400×400	1.148	0.007	6.047	3.724	5.242
	600×600	1.155	0.004	6.017	3.453	5.001
	900×900	1.155	0.002	5.960	3.198	4.755

Table 7.4: Estimated and measured percent load imbalance values, and percent errors in the estimation of ray-face intersection counts (I), sampling counts (S), local rendering times (T_{LR}).

GP Tool	Partition Quality	α_{avg}:avg. # of clus. per proc.			
		50	100	200	300
MeTiS	LI	5.66	3.92	2.95	2.62
	CS	1.00	1.00	1.00	1.00
RM-MeTiS	LI	4.91	3.36	2.56	2.48
	CS	1.24	1.23	1.23	1.25

Table 7.5: Worst-case performance evaluation of RM-MeTiS compared to MeTiS.

of I count of a cluster, the discretization errors in the estimation of the pixel counts of the faces with overlapping projection area accumulate, whereas the discretization in the estimation of S count is just on the boundary surface of the subvolume corresponding to a cluster. As seen in the table, the average percent error in the estimation of local rendering time (T_{LR}) remains between 5% and 6%. Note that T_{LR}^{err} values are considerably larger than both I^{err} and S^{err} values. This experimental finding stems from the additional errors introduced due to the estimation of unit costs t_I and t_S. Fortunately, as seen in the table, the estimation errors in T_{LR} do not directly reflect to load imbalance values, because of the cancellation affects. Note that I^{err} also determines the estimation error in the edge weighting of graph $\mathcal{G}_C^v$, since estimation of I in node weighting and estimation of edge weights both involve the same kind of face area computations.

Table 7.5 illustrates the worst-case performance evaluation of the modifications introduced to MeTiS to construct RM-MeTiS. In order to compare

RM-MeTiS with MeTiS, the pt-edges of the remapping graph are assigned zero weights, so that both tools have identical GP problem. The variation of average qualities of the partitions found by MeTiS and RM-MeTiS with different α_{avg} values are displayed in Table 7.5 for the standard viewing direction and $K = 16$ where averaging is over all datasets and all image sizes. Note that cutsize (CS) values are normalized with respect to those of MeTiS. As seen in the table, RM-MeTiS achieves slightly better load imbalance (LI) values while producing 23%–25% worse CS values than MeTiS. The relative performance degradation in CS quality stems from two factors. The mapping constraint enforced during the coarsening phase restricts the search space for the matchings. The more important factor is the direct K-way initial partitioning algorithm employed on the coarsest graph by RM-MeTiS as opposed to the recursive initial partitioning employed by MeTiS. However, the performance of the initial partitioning algorithm is heavily dependent on the weights of pt-edges. Hence, pt-edges with zero weights constitute a worst-case for the initial partitioning algorithm of RM-MeTiS, thus leading to the judgment that RM-MeTiS will perform reasonably good for the solution of the remapping problem.

Table 7.6 displays the results of experiments performed for justifying the need for view-dependent repartitioning and remapping. The values displayed in the table are for standard viewing direction and $K = 24$. In the *no-repartitioning* (NRP) scheme, the parallel rendering is conducted according to the OS decomposition inherited from the K-way partitioning of unit node and edge weighted view-independent visualization graph $\mathcal{G}_{\mathcal{V}}$ which is identical to the simulation graph $\mathcal{G}_{\mathcal{S}}$. Note that NPR scheme incurs no task migration and it is effectively equivalent to Ma's [18] implementation. In the *repartitioning* (RP) scheme, the view-dependent contracted visualization graph is partitioned using MeTiS, thus not considering the cluster migration at all. The RM scheme corresponds to the proposed remapping scheme. In Table 7.6, S_K represents the speedup values obtained on 24 processors. V_{CM} denotes the volume of communication (in Mbytes) due to cluster migration, R_{par} represents the total number of ray-segments (in thousands) generated during parallel rendering, and LI_m denotes the percent load imbalance value measured. Note that the two kinds of sources for the ray-segments are the ff-external faces of the dataset, and the ff-boundary faces which incur by the decomposition.

Data Set		Image Size								
		400×400			600×600			900×900		
		Decomp. Schemes			Decomp. Schemes			Decomp. Schemes		
		NRP	RP	RM	NRP	RP	RM	NRP	RP	RM
BF	S_K	3.71	16.06	17.12	3.75	18.06	18.70	3.78	18.99	20.02
	V_{CM}	0.00	7.87	2.12	0.00	8.01	2.52	0.00	7.90	2.86
	R_{par}	186	216	282	420	489	604	946	1090	1330
	LI_m	528.7	10.9	10.8	533.0	13.0	11.1	534.4	14.8	9.0
CC	S_K	18.01	18.60	19.75	18.82	20.46	21.05	19.23	21.39	21.55
	V_{CM}	0.00	9.21	0.58	0.00	9.23	1.40	0.00	9.12	3.42
	R_{par}	304	298	329	686	679	722	1545	1521	1629
	LI_m	17.7	4.4	3.8	17.7	4.4	3.7	17.9	4.3	4.1
OP	S_K	7.55	19.60	20.61	7.68	21.36	21.43	7.70	21.61	22.16
	V_{CM}	0.00	22.10	4.56	0.00	21.15	4.00	0.00	21.14	4.73
	R_{par}	483	425	570	1088	957	1232	2450	2155	2710
	LI_m	210.1	4.3	3.8	211.1	4.0	5.6	211.4	6.7	4.3

Table 7.6: Performance comparison of various OS decomposition schemes for K=24.

As seen in the Table 7.6, the speedup performance of RM is better than RP which is better than NRP in all parallel visualization instances. The NRP scheme, achieves good speedup values for the CC dataset which consists of almost equal sized cells. However, it performs drastically small speedup values as low as 3.71 for 24 processors on the other two datasets BF and OP, because of the extremely large load imbalance values reaching as high as 500%. As the cell size variations are inherently high in unstructured grids, view-dependent decomposition is crucial for parallel DVR of unstructured grids. The comparison of RM and RP schemes shows that the decompositions produced by the RM scheme require 5.5 times less cluster migration cost, whereas it incurs 1.2 times more ray-segment generations on the average. This experimental finding is expected, since the RM scheme considers both the cluster migration and ray-segment generation for minimization whereas the RP scheme considers only the ray-segment generation for minimization. Hence, the task migration should be considered in the remapping for better parallel performance.

7.3 Pixel Merging Phase

Table 7.7 displays relative performance comparison of pixel assignment (PA) schemes on the parallel pixel merging (PM) phase. In this table, T_{PM} denotes

K	Image Size		Pixel Assignment Schemes											
			SCA				MCA				BLMCA			
			Mesh Size				Mesh Size				Mesh Size			
			10^2	20^2	30^2	40^2	10^2	20^2	30^2	40^2	10^2	20^2	30^2	40^2
8	400×400	T_{PM}	0.48	0.46	0.47	0.46	0.62	0.63	0.64	0.65	0.54	0.54	0.56	0.56
		T_{PA}	0.00	0.00	0.00	0.00	0.07	0.08	0.08	0.09	0.07	0.08	0.09	0.09
		T_{RSM}	0.25	0.24	0.25	0.25	0.25	0.25	0.25	0.25	0.24	0.24	0.24	0.24
		T_{LPM}	0.22	0.21	0.20	0.20	0.29	0.29	0.30	0.30	0.22	0.21	0.22	0.21
		V_{RSM}	6.0	5.9	5.9	5.9	4.3	4.2	4.1	4.1	4.6	4.4	4.4	4.4
	900×900	T_{PM}	2.21	2.08	2.03	2.05	2.98	2.94	2.96	2.96	2.38	2.36	2.36	2.36
		T_{PA}	0.00	0.00	0.00	0.00	0.27	0.28	0.28	0.29	0.27	0.28	0.28	0.29
		T_{RSM}	1.16	1.09	1.08	1.09	1.17	1.14	1.12	1.14	1.11	1.09	1.08	1.08
		T_{LPM}	1.04	0.98	0.94	0.95	1.53	1.52	1.55	1.53	0.99	0.98	0.99	0.98
		V_{RSM}	27.6	27.7	27.7	27.7	19.3	18.9	18.8	18.7	21.6	20.6	20.5	20.4
24	400×400	T_{PM}	0.46	0.40	0.41	0.40	0.67	0.64	0.63	0.64	0.54	0.54	0.54	0.55
		T_{PA}	0.00	0.00	0.00	0.00	0.12	0.14	0.14	0.15	0.12	0.14	0.14	0.15
		T_{RSM}	0.28	0.25	0.25	0.26	0.30	0.28	0.28	0.29	0.28	0.27	0.26	0.27
		T_{LPM}	0.14	0.12	0.12	0.11	0.25	0.21	0.20	0.19	0.13	0.12	0.12	0.12
		V_{RSM}	11.3	11.4	11.4	11.4	9.3	9.1	9.1	9.0	10.4	9.8	9.6	9.6
	900×900	T_{PM}	1.88	1.57	1.51	1.52	2.62	2.45	2.43	2.43	2.02	1.92	1.91	1.92
		T_{PA}	0.00	0.00	0.00	0.00	0.29	0.31	0.32	0.32	0.29	0.31	0.31	0.33
		T_{RSM}	1.16	0.96	0.94	0.95	1.22	1.11	1.10	1.10	1.10	1.03	1.02	1.02
		T_{LPM}	0.69	0.58	0.54	0.54	1.09	1.02	0.99	0.99	0.61	0.56	0.56	0.56
		V_{RSM}	55.5	55.6	55.6	55.6	45.3	44.4	44.0	43.8	51.2	47.6	46.9	46.5

Table 7.7: Relative performance comparison of pixel assigment (PA) schemes on the parallel pixel merging (PM) phase.

the total execution time (in seconds) of the PM phase, and T_{PA}, T_{RSM} and T_{LPM} represent the dissection of T_{PM} into pixel assignment, ray-segment migration and local pixel merging times (in seconds), respectively. V_{RSM} denotes the total volume of communication (in MBytes) during the RSM step. Here, mesh size denotes the size of the 2D coarse mesh imposed on the 2D image-screen (e.g., 20^2 denotes a 20×20 coarse mesh). Each value displayed in the table represents the average for all datasets and all viewing directions.

As seen in Table 7.7, the SCA scheme achieves the best load balance in the LPM step for sufficiently large mesh sizes (e.g., 30^2 and 40^2), whereas it incurs the worst total volume of communication V_{RSM}. Note that T_{LPM} effectively shows load balancing quality of the respective PA scheme, since the local *RayBuffer* structures of the processors are identical along each row of the table. On the other hand, the MCA scheme achieves the lowest communication volume V_{RSM} while incurring the worst load balance in the LPM step, as expected. The BLMCA scheme shows in between performance such that it almost achieves the load balancing quality of the SCA scheme (only 3.2% worse on the average), and it approaches the communication volume quality of the

MCA scheme. It is interesting to note that although total communication volume values of both the MCA and BLMCA schemes are considerably less than those of the SCA scheme, the communication time T_{RSM} values of the SCA scheme are smaller than or equal to those of the MCA and BLMCA schemes. The scattering approach in the SCA scheme achieves the balancing of communication requirements of individual processors, thus reducing the concurrent communication volume. Hence the SCA scheme, which does not involve any pixel assignment overhead, achieves the best overall performance T_{PM} in all instances displayed in the table. The SCA scheme with coarse mesh size of 30×30 is selected in our implementation.

7.4 Overall Performance Analysis

Table 7.8 displays the percent dissection of view-dependent parallel rendering time (T_{par}) into remapping (T_{RM}), local rendering (T_{LR}) and pixel merging (T_{PM}) times. T_{RM} is further dissected into graph update (T_{GU}), graph partitioning (T_{GP}) and cluster migration (T_{CM}) times in the table. View-independent preprocessing overhead T_{CL} due to the clustering phase is displayed as a percent of T_{par}. The table also displays the communication volume (in MBytes) because of the cluster migration (V_{CM}) and ray-segment migration (V_{RSM}). The values displayed are the averages for all viewing directions.

As seen in Table 7.8, the clustering overhead is at most 37% of a single parallel rendering time. As T_{CL} is not dependent on image size, $\%T_{CL}$ decreases with increasing image size for a fixed K, such that it always stays below 9% for image size 900×900. Moreover, $\%T_{CL}$ remains almost constant with increasing K for a fixed dataset. So, the clustering overhead will become negligible after a few successive parallel visualizations.

As mentioned earlier GU and GP steps constitute the view-dependent preprocessing overhead for the sake of remapping. As seen in Table 7.8, percent view-dependent preprocessing overhead decreases with increasing image size for a fixed (dataset, K) pair, and both $\%T_{GU}$ and $\%T_{GP}$ reduces almost below 1% for all parallel visualization instances with image size 900×900. This

Data Set	Image Size	K	$\%T_{CL}$	Per. dissect. of par. render. time					Comm Vol (Mbytes)	
				$\%T_{RM}$			$\%T_{LR}$	$\%T_{PM}$		
				$\%T_{GU}$	$\%T_{GP}$	$\%T_{CM}$			V_{CM}	V_{RSM}
BF	400 x 400	8	31.67	1.57	0.91	0.97	92.62	4.07	1.23	5.46
		16	28.98	2.92	2.45	1.70	86.51	6.61	1.89	8.38
		24	30.07	4.76	4.43	2.40	80.25	8.27	2.33	10.76
	600 x 600	8	14.53	0.72	0.43	0.50	94.50	3.86	1.52	11.85
		16	13.51	1.37	1.20	0.89	90.49	6.21	2.30	18.40
		24	14.59	2.30	2.44	1.16	86.63	7.66	2.53	23.51
	900 x 900	8	6.58	0.33	0.20	0.25	95.47	3.85	1.92	26.29
		16	6.20	0.63	0.56	0.41	92.47	6.08	2.59	40.50
		24	6.83	1.08	1.14	0.58	89.91	7.56	3.05	52.34
CC	400 x 400	8	24.78	1.41	0.68	0.47	94.02	3.32	0.85	5.29
		16	25.49	2.61	1.70	0.87	88.96	5.66	1.04	8.24
		24	28.26	4.42	2.79	1.38	83.66	7.50	1.06	10.34
	600 x 600	8	11.33	0.64	0.33	0.29	95.56	3.20	1.28	11.56
		16	11.87	1.22	0.93	0.51	92.10	5.24	1.56	17.99
		24	13.54	2.13	1.79	0.84	88.41	6.98	1.77	23.10
	900 x 900	8	5.12	0.29	0.15	0.17	96.37	3.06	2.05	24.99
		16	5.40	0.55	0.46	0.36	93.59	5.16	3.15	39.81
		24	6.23	0.98	0.90	0.51	91.06	6.78	3.41	51.71
OP	400 x 400	8	37.46	2.02	0.62	1.35	92.80	3.02	3.77	6.30
		16	36.27	2.90	1.55	2.05	88.39	4.80	3.60	9.28
		24	36.47	4.35	2.75	2.62	83.43	6.29	4.41	11.88
	600 x 600	8	17.47	0.94	0.28	0.59	95.20	2.91	3.59	13.38
		16	17.15	1.38	0.74	0.84	92.26	4.59	3.25	20.32
		24	17.67	2.11	1.44	1.26	89.04	5.94	4.22	26.11
	900 x 900	8	7.90	0.43	0.13	0.28	96.33	2.83	3.81	29.12
		16	7.86	0.64	0.36	0.44	94.02	4.55	3.77	45.19
		24	8.20	0.98	0.69	0.63	91.84	5.88	5.13	58.12

Table 7.8: View-dependent clustering overhead as a percent of view-dependent parallel rendering time T_{par}, and percent dissection of T_{par}.

monotonic decrease is because of the fact that both T_{GU} and T_{GP} are independent from the image size. However, as seen in the table, both $\%T_{GU}$ and $\%T_{GP}$ increase with increasing K for a fixed (dataset,image-size) pair, where the rate of increase in $\%T_{GP}$ is larger than that of $\%T_{GU}$. As node and edge weighting computations are performed in parallel, local graph update computations do not contribute to the increase in $\%T_{GU}$. The communication overhead due to the all-to-all broadcast (AABC) operation in the GU step is source for this increase. In fact, the number of communications rather than the volume of communication in AABC is the dominant factor for the increase, since ring AABC is adopted. Hence, as also seen in the table, the rate of increase in $\%T_{GU}$ with increasing K decreases with increasing dataset size for a fixed image size. The increase in $\%T_{GP}$ with increasing K is obvious since RM-MeTiS is run serially in each processor and furthermore the execution time of sequential

RM-MeTiS increases with increasing K for a fixed remapping graph. However, parallel MeTiS [33] can be considered for parallel version of RM-MeTiS for better scalability.

Recall that the PM phase is an unavoidable postprocessing overhead in OS-parallel DVR. The following observations can be extracted from Table 7.8 for the variation of the PM overhead with different parallel visualization parameters. For a fixed (dataset,image-size) pair, both V_{RSM} and $\%T_{PM}$ increase with almost the same factor of 2 when K increases with a factor of 3 (from $K=8$ to $K=24$). For a fixed (dataset, K) pair, V_{RSM} increases with the increasing image size, where the rate of increase is almost equal to the rate of increase in the screen area. Note that increasing the image size from 400×400 to 900×900 increases the screen size by a factor of approximately 5. Fortunately, $\%T_{PM}$ always decreases both with increasing image size for a fixed (dataset, K) pair and with increasing dataset size for a fixed (image-size, K) pair.

Table 7.9 displays the speedup and efficiency performance of the proposed parallel DVR algorithm. In this table, T_{seq} and T_{par} denote the averages of the sequential and view-dependent parallel rendering times (in seconds), respectively, where averaging is over all viewing directions. Here, S_K and E_K represent the average speedup and efficiency values on K processors, respectively. As expected, efficiency increases with increasing image size for a fixed (dataset, K) pair. Although efficiency increases in general with increasing dataset size for a fixed (image-size, K) pair, there are some exceptions. For the smallest image size 400×400, the efficiency values obtained for CC dataset are always higher than those obtained for OP dataset although OP is a larger dataset than CC. This interesting behavior is because of the fact that extremely small cell size variation in CC compared to OP enables RM-MeTiS to produce better partitions in CC than OP both in terms of load balance (see Table 7.4) and cutsize qualities. As seen in the table, efficiency decreases very slowly with increasing K for a fixed (dataset,image-size) pair, especially for large datasets CC and OP, and larger image sizes 600×600 and 900×900. For the dataset OP, the efficiency values remain above 80%, 87% and 90% for image sizes 400×400, 600×600 and 900×900, respectively, for all K.

The BF and OP datasets have close and high cell-size variation values (COV

Data Set	K	Image Size											
		400 x 400				600 x 600				900 x 900			
		T_{seq}	T_{par}	S_K	E_K	T_{seq}	T_{par}	S_K	E_K	T_{seq}	T_{par}	S_K	E_K
BF	4	77.4	20.0	3.84	95.9	170.3	44.1	3.86	96.4	382.9	98.8	3.87	96.7
	8		10.5	7.32	91.4		22.9	7.42	92.7		50.7	7.54	94.2
	12		7.3	10.47	87.3		15.9	10.67	88.9		34.9	10.94	91.2
	16		5.7	13.46	84.1		12.3	13.81	86.3		26.8	14.24	89.0
	20		4.9	15.74	78.7		10.3	16.48	82.4		22.5	16.98	84.9
	24		4.3	17.80	74.2		8.9	18.97	79.0		19.2	19.90	82.9
CC	4	88.7	23.1	3.84	96.0	198.7	50.8	3.91	97.8	445.2	113.8	3.91	97.8
	8		11.9	7.45	93.1		26.1	7.62	95.2		57.8	7.70	96.3
	12		8.2	10.81	90.1		17.8	11.17	93.1		39.5	11.27	93.9
	16		6.4	13.94	87.1		13.7	14.53	90.8		30.1	14.80	92.5
	20		5.3	16.85	84.2		11.2	17.74	88.7		24.5	18.19	90.9
	24		4.6	19.45	81.0		9.5	20.84	86.9		20.7	21.49	89.5
OP	4	120.2	31.9	3.78	94.4	268.8	69.7	3.86	96.5	603.7	154.5	3.92	97.9
	8		16.3	7.34	91.8		35.3	7.62	95.2		78.0	7.74	96.8
	12		11.3	10.59	88.2		24.2	11.05	92.1		53.0	11.35	94.6
	16		8.6	13.86	86.6		18.3	14.60	91.2		40.2	14.98	93.6
	20		7.1	16.74	83.7		15.0	17.84	89.2		32.6	18.41	92.1
	24		6.1	19.41	80.9		12.8	20.90	87.1		27.6	21.77	90.7

Table 7.9: Average speedup (S_K) and efficiency (E_K) values.

values). Furthermore, their high COV values make them typical representatives of unstructured datasets. Hence, the BF and OP datasets are considered for an experimental scalability analysis. As seen in Table 7.9, the problem size, taken as the sequential execution time, increases by a factor in the narrow range 1.55–1.57 from BF to OP in all image sizes.

Hence, increasing K by the same factor from K_{BF} to $K_{OP} \approx 1.56K_{BF}$ for suitable K_{BF} values will constitute an experimental framework for linear scalability analysis [34], where the efficiency value $E_{K_{OP}}(OP)$ is compared to the efficiency value $E_{K_{BF}}(BF)$ for each processor pair (K_{BF}, K_{OP}) and each image size. Among the available processor pairs, (K_{BF}, K_{OP}) = (8,12), (12,20) and (16,24) are very close to the suitable processor pairs (8,12.5), (12,18.8) and (16,25.1), respectively. As seen in Table 7.9, $E_{K_{OP}}(OP) \approx 0.96 E_{K_{BF}}(BF)$ for each processor pair (K_{BF}, K_{OP}) in the smallest image size 400×400. However, $E_{K_{OP}}(OP) > E_{K_{BF}}(BF)$ for each processor pair (K_{BF}, K_{OP}) for image size 900×900. This situation also holds for image size 600×600 except for the processor pair (8,12). Hence, depending on the restricted available data, the proposed parallel DVR algorithm can be considered as linearly scalable for medium to large image sizes, where large image size is necessary for producing high quality images.

Chapter 8

Conclusion

An efficient object-space (OS) parallel direct volume rendering (DVR) algorithm was developed for the visualization of unstructured grids on distributed-memory architectures. A fast ray-casting based DVR algorithm was selected as the underlying sequential algorithm. The OS decomposition problem was modeled as a graph partitioning (GP) problem with correct view-dependent node and edge weighting to minimize the amount of redundant computation and interprocessor communication while maintaining the computational load balance. A GP-based model for the solution of the general remapping problem was proposed to enhance the OS decomposition model to make it also consider the task migration overhead for minimization for a better performance in successive visualizations. A remapping tool RM-MeTiS was developed by modifying and enhancing the original MeTiS package for partitioning the remapping graph. An effective view-independent cell-clustering scheme was introduced to induce more tractable contracted view-dependent remapping graphs for successive visualizations. An efficient estimation scheme with high accuracy was proposed for view-dependent node and edge weighting of the remapping graph.

The performance of the proposed parallel DVR algorithm was experimented on a Parsytec CC system with 24 processors for the visualizations of volumetric datasets. Efficiency values as high as 92% were obtained on 24 processors. The proposed parallel algorithm was found to be linearly scalable according to the experimental results. A realistic volumetric dataset consisting of half-million

cells was rendered at an image size of 400×400 in 6 seconds on 24 processors, thus approaching to interactive rendering speed.

Bibliography

[1] R. Yagel, "Volume viewing: state of the art survey," in *Visualization '93, Tutorial #9, Course Notes: Volume Visualization Algorithms and Applications*, pp. 82–102, 1993.

[2] M. Levoy, "Display of surfaces from volume data," *IEEE Computer Graphics and Applications*, vol. 8, no. 3, pp. 29–37, 1988.

[3] M. Levoy, "Efficient ray tracing of volume data," *ACM Transactions on Graphics*, vol. 9, no. 3, pp. 245–261, 1990.

[4] C. Upson and M. Keeler, "Vbuffer: Visible volume rendering," *Computer Graphics*, vol. 22, no. 4, pp. 59–64, 1988.

[5] K. Koyamada, "Fast traversal of irregular volumes," in *Visual Computing, Integrating Computer Graphics with Computer Vision*, pp. 295–312, 1992.

[6] K. Ma and J. S. Painter, "Parallel volume visualization on workstations," *Computers & Graphics*, vol. 17, no. 1, pp. 31–37, 1993.

[7] C. Montani, R. Perego, and R. Scopigno, "Parallel rendering of volumetric data set on distributed-memory architectures," *Concurrency: Practice and Experience*, vol. 5, no. 2, pp. 153–167, 1993.

[8] A. State, J. McAllister, U. Neumann, H. Chen, T. J. Cullip, D. T. Chen, and H. Fuchs, "Interactive volume visualization on a heterogeneous message-passing multicomputer," in *1995 Symposium on Interactive 3D Graphics*, pp. 64–74, 1995.

[9] R. Yagel and R. Machiraju, "Data-parallel volume rendering algorithms," *The Visual Computer, International Journal of Computer Graphics*, vol. 11, pp. 319–338, 1995.

[10] K. Ma, J. S. Painter, C. D. Hansen, and M. F. Krogh, "Parallel volume rendering using binary-swap compositing," *IEEE Computer Graphics and Applications*, vol. 14, no. 4, pp. 59–67, 1994.

[11] M. B. Amin, A. Grama, and V. Singh, "Fast volume rendering using an efficient, scalable parallel formulation of the shear-warp algorithm," in *Proceedings of 1995 Parallel Rendering Symposium*, pp. 7–14, October 1995.

[12] P. Lacroute, "Real time volume rendering on shared memory multiprocessors using the shear-warp factorization," in *Proceedings of 1995 Parallel Rendering Symposium*, pp. 15–22, October 1995.

[13] J. Challinger, "Parallel volume rendering for curvilinear volumes," in *Proceedings of the Scalable High Performance Computing Conference*, pp. 14–21, IEEE Computer Society Press, April 1992.

[14] J. Challinger, "Scalable parallel volume raycasting for nonrectilinear computational grids," in *Proceedings of the 1993 Parallel Rendering Symposium*, pp. 81–88, IEEE Computer Society Press, October 1993.

[15] J. Challinger, *Scalable Parallel Direct Volume Rendering for Nonrectilinear Computational Grids*. PhD thesis, University of California, 1993.

[16] B. Lucas, "A scientific visualization renderer," in *Proceedings of IEEE Visualization '92*, pp. 227–234, IEEE Computer Society Press, October 1992.

[17] P. L. Williams, *Interactive Direct Volume Rendering of Curvilinear and Unstructured Data*. PhD thesis, University of Illinois at Urbana-Champaign, 1992.

[18] K. Ma, "Parallel volume ray-casting for unstructured-grid data on distributed-memory multicomputers," in *Proceedings of 1995 Parallel Rendering Symposium*, pp. 23–30, October 1995.

[19] B. Hendrickson and R. Leland, "A multilevel algorithm for partitioning graphs," tech. rep., Sandia National Laboratories, 1993.

[20] M. P. Garrity, "Raytracing irregular volume data," *Computer Graphics*, vol. 24, no. 5, pp. 35–40, 1990.

[21] A. Doi and A. Koide, "An efficient method of triangulating equi-valued surfaces by using tetrahedral cells," *IEICE Transactions*, 1991.

[22] K. Koyamada and T. Nishio, "Volume visualization of 3d fem results," *IBM Journal of Research and Development*, 1991.

[23] W. Camp, S. Plimpton, B. Hendrickson, and R. Leland, "Massively parallel methods for engineering and science problems," *Communications of the ACM*, vol. 37, April 1994.

[24] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell System Technical Journal*, vol. 49, pp. 291–307, February 1970.

[25] C. M. Fiduccia and R. M. Mattheyses, "A linear-time heuristic for improving network partitions," in *Proc. 19th ACM/IEEE Design Automation Conf.*, pp. 175–181, 1982.

[26] C. J. Alpert and A. B. Kahng, "Recent directions in netlist partitioning," *VLSI Journal*, vol. 19, no. 1–2, pp. 1–81, 1995.

[27] G. Karypis and V. Kumar, "Metis: Unstructured graph partitioning and sparse matrix ordering system, version 2.0." Dept. of Computer Science, University of Minnesota. http://www.cs.umn.edu/~karypis.

[28] G. Karypis and V. Kumar, "Multilevel k-way partitioning scheme for irregular graphs," Tech. Rep. 95-064, University of Minnesota, Dept. of Computer Science, 1995.

[29] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," tech. rep., University of Minnesota, Dept. of Computer Science, August 1995.

[30] H. Kutluca, T. M. Kurç, and C. Aykanat, "Some experiments with communication on a parsytec cc system," tech. rep., Dept. of Computer Eng. and Information Sci., Bilkent University, 1997.

[31] Parsytec GmbH, Germany, *Embedded Parix (EPX) ver. 1.9.2 User's Guide and Programmers Reference Manual*, 1996.

[32] P. Shirley and A. Tuchman, "A polygonal approximation to direct scalar volume rendering," *Computer Graphics*, vol. 24, no. 5, pp. 63–70, 1990.

[33] G. Karypis and V. Kumar, "Parallel multilevel graph partitioning," tech. rep., University of Minnesota, Dept. of Computer Science, 1995.

[34] V. Kumar, A. Grama, A. Gupta, and G. Karypis, *Introduction to Parallel Computing, Design and Analysis of Algorithms.* California, USA: The Benjamin/Cummings Publishing Company, Inc., 1994.