FAST DIRECT VOLUME RENDERING OF UNSTRUCTURED GRIDS

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

By
Bülent Özüt
September, 1997

FAST DIRECT VOLUME RENDERING OF UNSTRUCTURED GRIDS

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

Hakan Berk

By

Hakan Berk

September, 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.

Prof. Bülent Özgüç

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

FAST DIRECT VOLUME RENDERING
OF UNSTRUCTURED GRIDS

Hakan Berk
M.S. in Computer Engineering and Information Science
Supervisor: Assoc. Prof. Cevdet Aykanat
September, 1997

Scientific Computing has become more and more important with the evolving technology. The vast amount of data that the scientific computing applications produce need new ways to be processed and be interpreted by scientists. The large amount of data makes it very difficult for scientists to extract useful information from the data, and interpret it to reach a useful conclusion. Thus, visualization of such numerical data as an image, which is named as *Scientific Visualization*, is an indispensable tool for researchers. *Volume Rendering* is a very important branch of Scientific Visualization and makes it possible for scientists to visualize the 3-dimensional (*3D*) volumetric datasets.

Volume Rendering algorithms can be classified into two categories: *Indirect* and *Direct* methods. Indirect methods are faster, but direct methods are more flexible, and accurate. Direct methods can be classified into three categories: *image-space (ray-casting)*, *object-space (projection)* and *hybrid*. The efficiency of a direct volume rendering (DVR) algorithm is strongly related to the way that it solves the underlying *point location* and *view sort* problems. Although these problems are almost trivial ones to solve in *structured grids*, they become more complex ones to deal with for *unstructured grids*. Researchers have tried to speed up the volume rendering of unstructured grids by using special graphics hardware, and parallel architectures, but the need for software solutions to these problems will always exist. This thesis is involved in solving those problems in unstructured grids via software methods. It investigates three distinct categories, namely *image-space* methods, *object-space* methods and *hybrid* methods for fast direct volume rendering of unstructured grids.

The main objective of the thesis is to identify the relative superiorities and inferiorities of the algorithms in these three categories. A survey of existing methods is enriched by a discussion of their merits and shortcomings. Three new and fast algorithms to overcome the existing inefficiencies are proposed, and one existing algorithm is investigated in detail for a better comparison. All of the proposed algorithms are aimed at producing correct, high quality images. Two of the proposed algorithms are pure ray-casting based solutions that support *early ray termination* and can handle *cyclic* grids. The relative performances of the proposed algorithms are experimented on a wide range of benchmark grids in a common framework for software methods and they are found to be faster than the existing best DVR algorithms.

Key words: Volume Rendering, Direct Volume Rendering (DVR), Unstructured Grid, Volume Visualization, Ray Casting.

ÖZET

DÜZENSİZ IZGARALARIN HIZLI DİREK HACİM GÖRÜNTÜLENMESİ

Hakan Berk. Bilgisayar ve Enformatik Mühendisliği. Yüksek Lisans
Tez Yöneticisi: Doç. Dr. Cevdet Aykanat
Eylül. 1997

Bilimsel Hesaplama her geçen gün gelişen teknoloji ile daha da önem kazanmaktadır. Bilimsel uygulamalar tarafından üretilen çok miktardaki verilerin bilimadamlarınca işlenmesi ve daha kolay anlaşılabilmesi için yeni yöntemlere ihtiyaç duyulmaktadır. Üretilen verilerin yüksek miktarda olmasından dolayı bilimadamlarının bu verilerden anlamlı ve işe yarar bilgileri çıkarmaları zorlaşmaktadır. Bu yüzden bu sayısal verilerin görüntülenmesi bilimadamları için vazgeçilemez bir araçtır, ve bilgisayar grafiklerinin bu konuyla uğraşan dalına da *Bilimsel Görüntüleme* adı verilir. Amaci 3-boyutlu hacimsel verilerin görüntülenmesi olan *Hacim Görüntüleme* ise Bilimsel Görüntüleme'nin en önemli alt dallarından birisidir.

Hacim Görüntüleme yöntemleri iki sınıfa ayırılabilir; *Dolaylı* ve *Direk*. Dolaylı yöntemler daha hızlıdır, fakat Direk yöntemler daha esnektir ve daha doğru sonuçlar verirler. Direk hacim görüntüleme yöntemleri de kendi içinde üçe ayrılırlar: *ekran-uzayı (ışın izleme)*, *cisim-uzayı* ve *karma* yöntemler. Direk Hacim Görüntüleme (DHG) yöntemlerinin verimliliği daha çok *nokta-yeri tespiti* ve *görüntü-sıralama* problemlerini ne şekilde çözdüğüne bağlıdır. Bu problemlerin çözümü *düzenli ızgaralarda* basit olmasına rağmen, *düzensiz ızgaralarda* daha zordur. Araştırmacılar düzensiz ızgaraların hacim görüntülenmesini özel grafik donanımı, yada paralel mimariler kullanarak hızlandırmaya çalışmaktadırlar, ama bu alanda yazılım çözümlerine her zaman ihtiyaç duyulacaktır. Bu tez düzensiz ızgaraların hacim görüntülenmesindeki problemlerin yazılım yöntemleri ile çözülmesi üzerinedir. Düzensiz ızgaraların hacim görüntülenmesi için olan üç ayrı kategorideki yöntemleri inceler. Tezin en önemli amaçlarından birisi değişik kategorilerdeki yöntemlerin avantaj ve dezavantajlarını saptamaktır. Bu konudaki önemli yöntemlerin tartışması,

bunların dezavantajlarının ve avantajlarının belirtilmesi ile zenginleştirilmeye çalışılmıştır. Bu problemleri çözmeye yönelik üç yeni ve hızlı yöntem geliştirilmiş ve daha iyi bir karşılaştırma için olan bir yöntem detaylı bir şekilde incelenmiştir. Tezde öne sürülen tüm yöntemler doğru ve yüksek kalite resim üretmeyi amaç edinmişlerdir. Bunlardan iki tanesi tamamen ışın-izleme üzerine geliştirilmiş yöntemler olup, *erken ışın sonlanmasını* desteklemekte ve *döngüsel* ızgaraları görüntüleyebilmektedirler. Bu yöntemlerin göreceli performansları geniş bir veri kümesi üzerinde deneysel olarak ölçülmüş ve olan en iyi DHG yöntemlerinden daha hızlı oldukları sonucuna varılmıştır.

Anahtar Kelimeler: Hacim Görüntüleme, Direk Hacim Görüntüleme, Düzensiz Izgara, Işın Izleme.

ACKNOWLEDGMENTS

I wish to express my deepest gratitude and thanks to Assoc. Prof. Cevdet Aykanat and Prof. Dr. Bülent Özgüç for their supervision, encouragement, and invaluable advice throughout the development of this thesis.

I would like to thank Asst. Prof. Uğur Güdükbay for reading my thesis, and his invaluable comments, suggestions and remarks.

I wish to extend my sincere thanks to all of my friends for their support during the development of the thesis. I would like to thank Ferit Fındık for his corporation on the codes written for the thesis work.

Finally, I would like to express my endless thanks to my family and Gaye Tüzemen for their endless support and patience.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The contribution of this thesis is primarily in *Volume Rendering*. Volume Rendering has become one of the most important sub fields of *Scientific Visualization*. The main focus of this thesis is on fast direct volume rendering of unstructured grids.

Scientific Visualization is a developing field of research. As scientific applications become more complex, the methods to process data and interpret acquired results need to improve in parallel. At this point, Scientific Visualization algorithms are utilized for detailed interpretation of those complex datasets to reach useful conclusions. Research in scientific visualization encompasses methods that act like a filter to transfer raw data into various kinds of representations where it is easier for human perception to interpret. This transformation usually occurs as mapping numerical data to sounds, images, etc. Therefore, scientific visualization can help the human mind process enormous amount of data that normally would be beyond our capacity to deal with.

Scientific visualization tries to establish a general set of principles for visually representing complex data structures. The most primitive scientific visualization techniques include tables, different charts *(e.g. bar, pie)*, and point plots. These are mainly useful for simple representation of functions of the

kind $f(x)$. Then comes the contours and images which help a lot in understanding functions of the kind $f(x,y)$. These techniques are largely used in weather forecasting, and geography. So far, the techniques presented are suitable for use with scalar data, so none of them are able to represent a dataset where both magnitude and direction are of vital importance. In such cases, a vector representation will be inevitable. Although, vectors are thought as tools to visualize multiple components of quantities like velocity and magnetic fields, they are also useful for interpretation of multidimensional gradients of scalar functions. As the new parameters added, the existing methods become useless in representing the original dataset in the appropriate way that will be best to capture for human perception. For instance, none of the methods mentioned above are suitable for understanding the heat distribution on a wing of an aeroplane, or representing the data acquired by a *Magnetic Resonance (MR)* scan device. The reason is that none of the above methods are suitable for the representation of 3 dimensional spatial data. The sub field of scientific visualization the main concern of which is to visualize 3 dimensional spatial datasets is called *Volume Rendering (Volume Visualization)* [1, 2].

Volume rendering seeks methods to generate a meaningful image of volumetric data. The main goal is to produce a graphical representation of the volumetric data such that the scientist or engineer can interpret this complex dataset. Therefore, it must, somehow, be able to map various properties of the volumetric data in an intuitive and consistent way. More formally, volume rendering deals with the visualization of data sets that consists of large amount of numerical data values associated with points in 3 dimensional space.

Three dimensional datasets are usually given as a set of points defined on 3 dimensional Cartesian space where each data point represents a scalar, vector, etc. value about an entity like a bone, tissue, or plane. These data points are called *sampling points* as they represent sample results about the property of the entity to be visualized. For instance, in medical imaging they can represent the density, whereas in the simulation of an aeroplane wing they may represent heat. The spatial topology of the distribution of the sample points in 3 dimensional space is of vital importance in the visualization process. The major distinction that categorizes the grids that are formed of sampling points is the

regularity/irregularity of the structure. If the samples are distributed over a grid with equal or variable spacing where an implicit connectivity between the grid points exists, then it is called a *structured grid*. In this case, the sample points can be supplied as a simple 3D computational virtual array where each neighbor relation in this array corresponds to the same neighbor relation in the 3D Cartesian space. This sort of topology is common to medical imaging applications such as *Computer Tomography (CT)* or *Magnetic Resonance (MR)*. The grids that are in the form of the other type of topology are called unstructured grids. As the name implies, there does not exist an implicit structure in the distribution so the connectivity relation must be supplied explicitly. These types of grids are common in *Computational Fluid Dynamics (CFD)*, *Finite Volume Analysis (FVA)*, and *Finite Element Methods (FEM)*.

Usually, volume data is represented by 3D *voxels (Volume Elements)* that constitute the atomic pieces of the overall data structure in the context of domain discretization. The shapes of the voxels are variant. The vertices of those voxels are the sample points discussed above. Therefore, there is a strong relation between the distribution of sample points, and the shapes of the voxels. The most common voxel shape in structured grids is a hexahedral that consists of 8 vertices. In unstructured grids, tetrahedral, and hexahedral voxel shapes are quite popular. The information about the attribute of the entity to be visualized is either stored in those voxels or in the sample points. This difference in the data is also reflected to visualization methods. Note that some authors use cell and voxel interchangeably, but in this work cell will be used to avoid confusion. Also throughout this thesis, the data model will be based on the assumption that the data values are stored at the sample points instead of the cells.

Volumetric datasets are rendered by finding the contributions of sample points to the pixels on the image plane. These contributions are determined via processing of the cells. The data values in the sample points are somehow mapped to color and opacity values via an optical model[1], and then those color, and opacity values are composited in the appropriate order to produce a meaningful image.

[1]For a complete discussion of optical models, please refer to [3]

There are two major categories of volume rendering methods: *Indirect (Surface based)* and *Direct methods*. The methods that fall into the former category 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. Indirect methods are potentially faster, and are more suitable for medical imaging, and biological applications where the visualization of surfaces inside a volume makes sense *(e.g. visualization of tissues to identify a tumor)*. Direct 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. In one of the famous classifications of direct methods, they are classified into three categories: *image-space (ray-casting)*, *object-space (projection)* and *hybrid*.

In general, volume rendering methods consist of two main phases. These are *resampling* and *composition* phases. In resampling phase, new samples are interpolated by using the original sample points. In composition phase, those new samples generated are mapped to color, and opacity values, and those color and opacity values are composited. In direct methods, resampling phase must be repeated for any change in the viewing parameters, but in indirect methods, once the isosurfaces are extracted, it should only be repeated if visualization parameters other than viewing parameters change. This is one of the reasons why indirect methods are faster than direct methods. During the resampling phase, the rendering method should somehow locate the new sample point in the cell domain, because the vertices of that cell, in which the new sample is being generated, will be used in the interpolation for the new sample. This problem is known as *point location* problem. In the composition phase, the color, and opacity values must be composited to determine the contribution of the data on a pixel. The composition operation is associative, but not commutative, therefore those color and opacity values should be composited in a predetermined *(front-to-back or back-to-front)* order. The determination of the correct composition order is known as *view sort* problem. These two problems are almost trivial ones to solve in structured grids, but the way that a volume rendering algorithm handles them is a crucial issue that strongly affects the performance of the rendering process for unstructured grids. The lack of

implicit connectivity between cells and the irregularity of the distribution of the sample points in the unstructured grids are the major factors that produce this difference.

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 rendering of unstructured grids, which is the major concern of this thesis, is still far from interactive response times. The slowness of the volume rendering process create the lack of interactivity which in turn prevents it from being widely used. Therefore, indirect methods have become very popular in spite of the fact that they give approximate and sometimes inaccurate images. Their interactive speed prepared a platform for their widespread use. Then scientists began to use indirect methods for interactivity and experimentation, and rely on direct methods for accurate renderings. This is one of the major reasons why there is a need for faster direct volume rendering algorithms. 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 techniques could help the scientists experiment with the parameters to select the useful ones.

One of the attempts to speed up the visualization process is to rely on special graphics hardware, while some other researchers try to develop parallel algorithms. But nevertheless, the need for a software solution for fast rendering of unstructured grids will always exist. Therefore the main concern of this thesis will be on fast rendering of unstructured grids via software solutions.

1.1 Contribution of the Thesis

As discussed in the previous sections, there is a need for fast volume rendering algorithms of unstructured grids on a software basis. This thesis investigates three distinct categories, namely *image-space* methods, *object-space* methods and *hybrid* methods for fast direct volume rendering of unstructured grids. The main objective of the thesis is to identify the relative superiorities and inferiorities of the algorithms in these three categories. At least one algorithm

from each category is implemented and experimented in a common framework for software methods.

Koyamada's algorithm [4], being one of the outstanding algorithms of image-space methods, has been selected as the representative of image-space approaches in this framework.

A new object-space method called Melting Cells (MC) algorithm, which differs from existing methods in its capability of producing high quality images and the sorting schema based on exploiting both image and object-space coherencies, is proposed.

Existing hybrid methods suffer from inability to support early ray termination and performing redundant computations especially in low resolutions as the portion of data contributing to the image is relatively small. A new hybrid method, called Span-Buffer Ray-Casting (SBRC) algorithm is proposed to overcome all these deficiencies of the existing methods by changing the computational traversal order from object-then-image to image-then-object without comprimising full utilization of both image and object space coherencies exploited by Koyamada's and MC algorithm.

A fourth algorithm, namely Hybrid-SBRC (H-SBRC), stemmed from the idea of refining image-space and hybrid methods to extract the best features of each, is developed by blending Koyamada's and SBRC approaches. It exploits the computational traversal order proposed by SBRC algorithm to process small cells, where the overhead of using image-space coherency does not afford the gains expected, in a more cost effective way by employing a ray-casting schema which ignores image-space coherency, but still performs better. Two heuristics, namely Exact Area (EA) and Bounding Box Area (BBA) approximation, have been proposed which are used in determining the cell-processing schema to be employed.

Another contribution of the thesis is the proposal of optimization schemes to avoid the potential memory overheads without comprimising the run-time efficiency of the proposed algorithms.

The algorithms are implemented and experimented using tetrahedralized

versions of curvilinear NASA data sets widely used for benchmarking rendering algorithms. The relative performances of the algorithms are evaluated by the visualization of each of these data sets using four distinct views for three different resolutions for a better average case analysis.

The respective performances of the presented algorithms are compared with one of state-of-the-art algorithms called Lazy Sweep Ray-Casting (LSRC) [5] algorithm which is a hybrid approach and is reported to be two orders of magnitude faster than existing software methods. Considering the run-time estimates of LSRC algorithm on our benchmark machine, MC algorithm is found to be 1.8 to 2.4. SBRC and H-SBRC algorithms are found to be 1.5 to 3.2 times faster than LSRC algorithm on the overall average. Restricted reported results of LSRC algorithm reveals that it scales slightly better than SBRC and H-SBRC algorithms, on the other contrary SBRC and H-SBRC algorithms scale much better than LSRC algorithm with increasing data set size.

1.2 Organization of the Thesis

The rest of the thesis is organized as follows.

In Chapter 2, an overview of volume rendering is presented. It serves as a kind of literature survey that describes the terminology, and existing volume rendering methods. The main focus in this chapter will be on the previous work done for direct volume rendering of unstructured grids.

In Chapter 3, four algorithms that are the main concern of this thesis are introduced. The first algorithm is an existing practical algorithm, called *Koyamada's* algorithm. Following that, three new algorithms proposed are explained, and discussed in detail.

In Chapter 4, the test data and experimental results obtained from the four algorithms discussed in this thesis are presented.

Finally, in Chapter 5, the results of the work done are discussed as conclusion.

Chapter 2

Volume Rendering: An Overview

"Visualization is a method of computing. It transforms the symbolic into the geometric, enabling the researchers to observe their simulations and computations. Visualization offers a method for seeing the unseen. It enriches the process of scientific discovery and fosters profound and unexpected insights. In many fields it is already revolutionizing the way the scientists do science."

B. McCormick, T. DeFanti, and M. Brown, 1987. [6]

Scientific Computing has become more and more important with the evolving technology. The vast amount of data produced by the scientific computing applications need new ways to be processed and be interpreted by scientists. This need emerged a new field in computer graphics which is called *Scientific Visualization*. Scientific Visualization covers a broad area from *Medical Imaging* to *Computational Fluid Dynamics (CFD)* simulations, and from weather forecasting to *Geographical Information Systems (GIS)*. *Volume Rendering* is a very important branch of Scientific Visualization.

2.1 Volume Rendering

Volume Rendering is a technique to visualize 3-dimensional (3D) volumetric data that is in the form of a grid superimposed on a volume. The nodes of this grid contain the scalar values that represent the physical entity or the physical environment. For example, in medical imaging, magnetic resonance imaging (MRI) or positron emission tomography (PET) techniques are used to scan a specific part of the human body. The output of these techniques are in fact large amounts of numerical data in the form of a volumetric dataset, which when visualized in an abrupt way, will let the doctors see the different properties of tissues in that part of the body. In aerospace research, the scientists may want to visualize the heat distribution on a space shuttle, while in CFD simulations they will concentrate on visualization of vortices that are formed during the flow. The large amount of data makes it very difficult for scientists to extract useful information from the data, and interpret it to reach a useful conclusion. Thus, visualization of such numerical data as an image is an indispensable tool for researchers. Volume rendering makes it possible for scientists to visualize those volumetric data sets.

2.2 Terminology and Overview

Volumetric Dataset can be defined as a set of points in 3D space in a volume. The term *sample point* is used to refer to a point in 3D spatial coordinates for which a numerical value is associated. Sample points 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 types of cell shapes: rectangular prism, hexahedra, tetrahedra, and polyhedra being the most popular ones. In volumetric data sets, one or more cells can share a face. This, in turn, brings out a connectivity relation between cells. If a face is shared by two or more cells, then the face is called an *internal (interior) face*, otherwise, if it belongs to only one cell in the dataset, then it is called an *external (exterior) face*. A cell that has at least one external face is said to be an *external cell*, and a cell all the faces of which are internal is called an

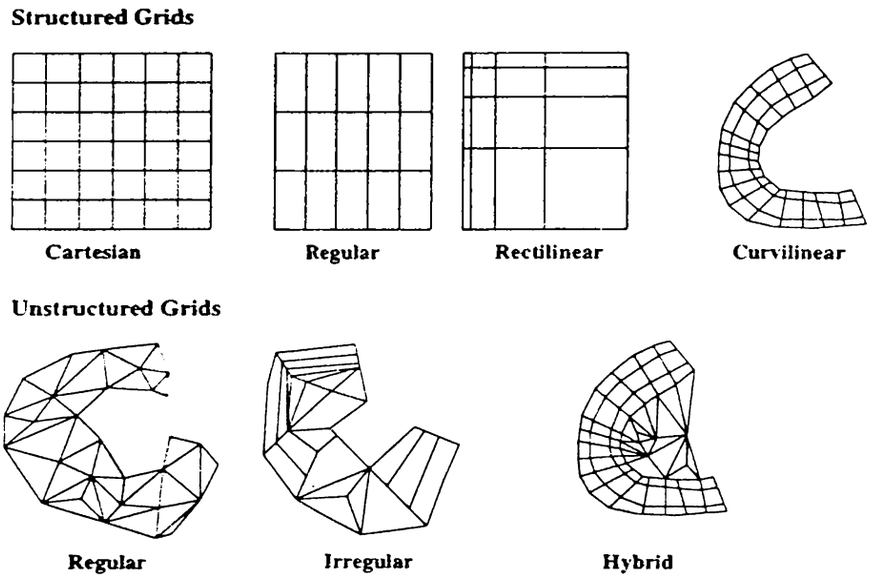


Figure 2.1: Types of grids encountered in volume rendering.

internal cell.

The topology of sample points impose a grid on the volume. Type of the grid also defines spatial characteristics of the volumetric dataset, which is important in the rendering process. Various classifications of grids exist in the literature [7, 8, 9, 10]. Figure 2.1 shows one of the most popular classifications of grids made by Yagel [9]. In his work, he divides the grids into two categories which are further classified as:

- Structured Grids
  - Cartesian
  - Regular
  - Rectilinear
  - Curvilinear
- Unstructured Grids
  - Regular
  - Irregular
  - Hybrid

In structured grids, the sample points are distributed regularly in 3D space. The distance between points may be constant or variable. This property is easily observed in *cartesian. regular* and *rectilinear* grids. In *curvilinear* grids it is not so obvious. In curvilinear grids, a structure in the distribution of sample points still exists, but is harder to capture. The sample points are distributed to fit onto a curvature in space. The cell shapes in structured grids are hexahedral cells. They have eight vertices. As these grids preserve a structure 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 in these grids is implicit, and one-to-one. Due to array oriented nature of structured grids, the connectivity relation between cells is also 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. The spacing between sample points is variable, and there exists no constraint on the cell shapes. Most common cell shapes in unstructured grids are hexahedral and tetrahedral shapes. Unstructured grids are common in engineering simulations especially in Computational Fluid Dynamics (CFD), Finite Volume Analysis (FVA). In addition, curvilinear grids are also common in CFD simulations. 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 if necessary. Unstructured grids can be further divided into three subtypes as *regular, irregular* and *hybrid*. In regular unstructured grids, the cell shapes are consistent, and one face is shared between at most two cells. In irregular unstructured grids, there is no consistency in cell shapes, and one face can be shared by more than two cells. Hybrid unstructured grids appear as a combination of structured and unstructured grids.

2.3 Taxonomy of Volume Rendering Algorithms

Volume rendering algorithms can be classified into two main categories [11]:

- Indirect Volume Rendering algorithms (Isosurface extraction)
- Direct Volume Rendering algorithms

The main aim of the algorithms that fall into the first category is to extract intermediate geometric representation of a surface from the volumetric data, and render the surface using conventional rendering techniques, whereas direct methods aim to visualize the volume directly without using intermediate geometric primitives. Direct methods are computationally more expensive, but give more accurate results compared to the indirect methods.

2.3.1 Indirect Volume Rendering algorithms

Isosurface extraction algorithms try to detect level surfaces throughout the volume as best as possible (a level surface is defined herein as a continuous surface within the volume such that, at each point on the surface the value of the scalar is the same [12]. Most of the work done on this category is for structured grids. Within this category, there are two subdivisions: first, techniques that operate on slices by applying 2D tracking algorithms and then extracting a surface between consecutive contours [13, 14]; and a technique that applies an isovalue surface detector to each set of eight voxel vertices in the dataset, to produce a large set of voxel-sized polygons which are then rendered [15, 16].

In [13] contours, which are detected using an edge tracking algorithm, are used to define regions of interest in each slice. A skimming algorithm [17] is then employed to connect contours in adjacent slices to form a polygon mesh.

Both [13] and [14] are two stage methods. In the first stage, the contours are detected in the 2D slice domain, and in the second stage, adjacent slices, or contours are used for final rendering. The major problem with both of these approaches is that there is no guarantee that the tracking will detect only single contours in each slice. For example, a blood vessel which branches into two may result with one contour in one slice, and two contours in the adjacent slice. The problem becomes more difficult to handle when the distance between

slices become large with respect to the size of cells on the slices [18]. The major deficiency of those methods arise from their non-isotropic nature in 3D space. One algorithm is used to track contours on the slices, whereas a completely different heuristic is employed to connect contours in adjacent slices.

The next approach which is called *Marching Cubes Algorithm* does not use slices to detect isosurfaces, but it extracts a surface directly from the acquired data and therefore is isotropic in 3D space.

In [16], the algorithm is said to have two primary steps for surface construction problem. In each voxel, the surface corresponding to a user-specified threshold is located, and the triangles that are formed by the intersection of surfaces and voxels are classified. After this stage, at each vertex of each triangle the normals of surfaces are calculated, and using the gradient values isosurfaces are formed. The next stage is to display isosurfaces using conventional rendering techniques. In his subsequent work, Lorenson introduces a new algorithm that he names as *Dividing Cubes* algorithm. Both Marching Cubes and Dividing Cubes algorithm produce 3D surface contours from volume data, the former creates triangles as primitives whereas the latter uses points with normals. In both techniques, a user specified threshold is used to select the surface to be visualized. Another important point to state is that the former is more suitable for cases where the number of triangles is less than the number of pixels, whereas the latter is more appropriate for cases where the number of triangles approaches the number of pixels.

In [17], a method which supports the extraction of isosurfaces from irregular volume data represented in tetrahedral decomposition in optimal time is presented. The method is based on a data structure called *interval tree*, which encodes a set of intervals on the real line, and supports efficient retrieval of all intervals containing a given value. Each cell in the volume data is associated with an interval bounded by the extreme values of the field in the cell. All cells intersected by a given isosurface are extracted in $O(m + logh)$ time, with m the output size and h the number of different extreme values (min or max).

Isosurface extraction algorithms are very suitable for medical imaging applications, since they need to display multiple tissue densities and specific tissue

boundaries. It is possible to assign a particular color and effective opacity to each tissue in boundary based on its isodensity value. Then surface contours that define each tissue type can be used to determine the actual structure and location of the surface, its color and opacity. Contours can be made visible to varying degrees by assigning opacities according to the order of the boundaries between different tissues in the dataset.

2.3.2 Direct Volume Rendering algorithms

One of the main problems with indirect methods is that in those methods a discontinuity is forced into the data by introducing artificial surfaces. These discontinuities are highly non-linear and therefore introduce artifacts into the final image. Thus, a surface visualized in the rendered image may not reflect the actual structure of the data. Direct methods overcome this problem by assuming the volume data to be made up of small particles that emits, absorbs or reflects light depending on the lighting model used. Therefore, the images produced are like semi-transparent clouds with less artifacts.

Direct Volume Rendering (DVR) methods can be divided into two phases: *resampling* and *composition phase*. In resampling phase the volume is reconstructed by creating imaginary sample points in the volume by making use of interpolation methods from the existing sampling points. The new sample points are created along the path of rays cast from each pixel into the volume data. To create a new sample, the vertices of the cell in which the new sample will be created are used. Most approaches use *Inverse Distance Interpolation* [19] to find the value of the scalar in the new sampling point. That is, first, distance of each vertex to the new sample point is calculated, and then contributions of each vertex to the sample point is calculated inversely to the distance of the vertex to the new sample point. Therefore, the smaller is the distance between the vertex and the new sample point, the larger is the contribution of the vertex.

In composition phase, the new interpolated samples are mapped to a color C_S and an opacity value O_S by applying a *mapping function*, also called the *transfer function*, which converts the numerical value of the scalar to a color and

opacity value. The selection or design of transfer functions have a high impact on the resulting image, and is an area of research that is beyond the scope of this thesis. The new color and opacity values obtained are composited in either *front-to-back*, or *back-to-front* order to from the final image. In front-to-back composition, the samples are composited starting from the first sample closest to the image plane, and in back-to-front composition, the furthest sample is the first to start the composition process. If composition is performed from back-to-front, following equation is evaluated to find the composited color at a pixel

$$C_{i+1} = C_i(1 - O_s) + C_s O_s \tag{1}$$

where C_{i+1} is the color after compositing the new sample, C_i is the color composited from the previous sampling points, C_s is the color and O_s is the opacity at the current sampling point both of which are obtained by application of interpolated scalar to the transfer function. Initially, a background with opacity $O_s = O_b = 1$ is placed or assumed to be placed behind the volume and $C_s = C_b =$ *background color* is taken as the color at starting point. For front-to-back compositions, the following equation [20] is evaluated to calculate the composited color at a pixel

$$\begin{aligned} C_{i+1}O_{i+1} &= C_i O_i + C_s O_s(1 - O_i) \\ O_{i+1} &= O_i + O_s(1 - O_i) \end{aligned} \tag{2}$$

where (C_{i+1}, O_{i+1}) is the (color,opacity) tuple after processing sample point, (C_i, O_i) is the (color,opacity) tuple before processing sample point, and (C_s, O_s) is the (color,opacity) tuple at the current sample point. Initially C_0 and O_0 are set to zero. In both cases the composition operation is associative but *not* commutative so the order of composition is very important for the accuracy of the image. This restriction requires that either the sample points should sorted after being estimated, or they should be calculated in a sorted way.

The process of determining the cell that contains a sample point in the resampling phase is called *point location* problem. Sorting sample points or estimating them in a sorted order is defined as *view sort* problem. Both of the problems are easier to solve in structured *cartesian*, *regular*, and *rectilinear* grids because the regular distribution of the sample points in the volumetric

dataset and implicit connectivity between volume elements make these problems almost trivial ones to solve. However, solving point location and view sort problems in *curvilinear* and *unstructured* grids is more difficult. In unstructured grids, data points (original sample points), hence cells, are distributed irregularly over 3D space. A naive algorithm may need to search all cells to determine the cell to be used in the calculation of a new sample point. This can result in very large execution times as expected. In addition, sorting sample points along a ray takes a lot of time, if not handled efficiently. Therefore, the performance of the algorithm heavily depends on the efficiency of the techniques that are used to solve these problems. Approaches to solve these two problems in unstructured grids are explored in more detail in Section 2.4.

We can classify direct methods into the following three categories:

- Ray casting methods
- Projection methods
- Hybrid methods

Ray casting methods are also called *image-space* methods because there is a notion of a ray which is shot from each pixel on the screen and is followed and sampled in the volume to find the contribution of the volume data onto that pixel. Projection methods are, in general, approximations to ray casting, but they are faster as they use image-space coherency in the data more efficiently. They are also called *object-space* approaches. Hybrid methods are those which process the volume data in object-space order creating spans by interpolation to find the contributions of cells.

Ray-Casting Methods

Ray casting is a popular method to construct images from volumetric data sets. In ray casting, at least one ray per image pixel into volume is cast, and is sampled along the volume with a lighting model to find the contribution of the data to that pixel in the image. It should be noted that for non-convex

data sets, 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 contribution of data to the pixel, are referred to here as *ray-segments*. In this section, we will investigate the related work on ray-casting methods for structured grids.

Blinn [21] introduces the use of density models in computer graphics where he considers plane parallel atmospheres. Other researchers have adapted his models to more general shapes. Max [22] defines clouds as densities the boundaries of which are defined by analytical functions. Voss [23] fractally generates densities with a series of plane parallel models. His technique yields very realistic images. Kajiya and Von Herzen [24] present a new algorithm to trace objects represented by densities within a volume grid. They develop light scattering equations presenting a new approximate solution to the full-3D radiative scattering problem, and they compare it with the previous models.

Drebin *et al.* [25] assumes that the measured data consists of different materials *(e.g, skin, bone, and muscle)*, and that at any point in the volume the scalar represents the sum of the materials at that point. They further assume that no more than two materials can exist in a cell, so it is possible to describe each voxel with the percentage of materials within it. Using this percentage of the mixture of materials, the opacity and color of the voxels can be assigned. The method also explicitly computes certain boundaries between different materials within the volume based upon density gradient across voxel boundaries. Sharp transition between materials result in large gradients. They model both surface color and opacity in this manner. Their approach does not threshold the data, since thresholding may produce unwanted artifacts. One drawback of their method is that one should provide as input all the expected material densities. This means that either a model that assigns effective weights to voxels exist, or that one has to be very familiar with the data. Also the assumption that only two materials can be existent in a voxel may not always hold if the volume data is sampled at a low resolution.

A different approach has been taken in a series of papers by Levoy [26, 20]. The basic concept common in these works is to assign a color (in terms of red, green, and blue), and an opacity to each data point in the 3D grid, and then to compute the color of a pixel defined by a ray emanating from the observer's

eye through the image plane, and penetrating into the data. Samples may be taken at each voxel intersected by the ray, or at fixed size intervals. The advantage of their techniques over the method proposed by Drebin *et al.* [25] is that one does not need a density model of the dataset for the process.

Lacroute [27] introduces a new algorithm which makes use of shearing and warping transformations to render rectilinear grids. His algorithm seems to be the fastest algorithm to render rectilinear grids. He reports that a typical volume of size $256 \times 256 \times 256$ is rendered approximately at 1 second on an R4000 Indigo machine by his *Shear Warp* algorithm, but his method is not applicable to unstructured grids.

Avila, Sobierajski and Kaufman [28, 29] introduce the idea of using special hardware to speed up volume rendering. In [28], they present an algorithm named *PARC (Polygon Assisted Ray-casting)* which makes use of the *Z-Buffer* [11] algorithm to find the closest and furthest possibly contributing cells. In [29], they revise their method by proposing a new technique which approximates the effects of contributing cells better.

Projection Methods

Another way to produce the 2D image correspondent of a volumetric dataset is to find the contribution of each cell to the image. These projection methods are also known as object-space methods. [25, 30] are examples of such methods. The most important problem in projection methods is the view-sort problem that necessitates a view dependent sorting of the cells to find their contribution in the correct order. This problem is easier to solve in structured grids. This section presents the most common projection algorithms for structured grids.

Westover [31] describes an approximation technique called *Splatting*. The name comes from the similarity of the technique with the action of striking snowballs to a wall. The cells are projected onto the image space, and their contribution is summed up to form the final image. He calls the contribution of each cell as *footprint*. Formally, his method reconstructs the signal that represents the original object, and samples it computing the image from the

resampled signal. For each voxel, the algorithm estimates the contribution to the image plane, its footprint, and accumulates that footprint in the image plane buffer. He shows that both front-to-back, and back-to-front composition of contributions is possible. The accuracy of his method depends highly on the correct computation of the footprints. In [30], he proves that for parallel projections, the footprint can be approximated by a lookup table, because it does not depend on the spatial location of the voxel itself for such projections.

Neumann and Cullip [32] introduce a new method of rendering volumes that leverages the 3D texturing hardware in Silicon Graphics Reality Engine workstations. Their method defines the volume data as 3D texture and utilizes the parallel texturing hardware to perform reconstruction and resampling on polygons embedded in the texture. The resampled data on each polygon is transformed into color and opacity values and composited into the frame buffer. They introduce two alternative strategies for embedding the resampling polygons, and discuss the benefits and trade-offs off each method. They report interactive rates of rendering such as 10 frames per second for a $128x128x128$ dataset.

Hybrid Methods

In hybrid methods, the volume is traversed in object-space and the contribution of cells are determined by creating spans. In other words, each cell is transformed with the viewing parameters and is intersected by the planes defined by scanlines on the screen. The resulting polygons are divided into spans and by interpolation the contributions of cells are determined efficiently. Upson and Keeler's *V-Buffer (Visible Buffer)* algorithm is an example of the hybrid approaches for regular grids [33].

2.4 Direct Volume Rendering of Unstructured Grids

As discussed in the previous sections, point location and view sort problems are the most important ones to solve in the rendering of unstructured grids. In this section, the methods developed to solve these problems efficiently are examined. Most of the approaches use the coherence in the volume and on the image space to increase the efficiency of the algorithm. The former and latter types of coherencies are referred to here as *object-space coherency* and *image-space coherency*, respectively.

The most obvious way to render an irregular grid is to resample it into a regular grid and then render it with available methods. Oppenheim and Schafer [34] survey techniques for the resampling of regular grids. The corresponding theory of irregular grids is examined in detail by Feichtinger and Grocheng [35]. One of the methods is to send rays directly into the volume and store the samples in a 3D buffer. Another approach is to impose a regular grid on the irregular one, and interpolate the original data to the new grid. As this resampling to a regular grid will be done only once, it has a direct impact on the quality of the images produced. Although good interpolation techniques exist, they take too much time. This can be tolerated as it will only be performed once, but it turns out that accurate resampling is not only time-consuming but results in regular grids so large that the actual rendering becomes very slow. The main reason for this is that the irregular grids have cells of varying size and shape. The difference between the small cells which in fact denotes the area of interest and large cells where the precision is not so important is often very large. Therefore, an interpolation to a regular grid that preserves the accuracy and precision creates very large data sets. Another drawback of these methods is that they generate huge empty areas as the regular grid has to cover the whole bounding box of the irregular grid. Moreover, the irregular data could be created in a regular way if desired, so direct ways to handle those types of grids efficiently are required. To sum up, resampling to a regular grids immense drawbacks regarding the produced data size, the data quality, and the topology information.

Ray-Casting Methods

In this section, we will look at some of the ray-casting methods for unstructured grids. Most of these approaches try to exploit the connectivity relation available in the data to speed up the ray-casting process.

Garrity [36], proposes an algorithm that makes use of the connectivity information between cells to traverse irregular grids. In his method, rays are first intersected by external faces to find the entry point of a ray into the volume. In order to further decrease the search for the first intersection, all external faces are geometrically sorted into a coarse 3D mesh. The ray is intersected with this mesh, and only the faces in the mesh locations, that are intersected by the ray, are tested for the ray intersection. Once the first entry point to the volume is determined, the rays is followed in the volume cell by cell by utilizing the connectivity information. Entry point of the ray to the next cell, which is the exit point of the ray from the current cell, is found by only considering the faces of the current cell. After an exit point is calculated, next cell that shares the face is found through the connectivity information. If the grid is a regular unstructured grid, this process is relatively faster, as one face can be shared at most by two cells. As there may be voids in the original data, when the ray exits the volume, the external faces are intersected with the ray again to find the next entrance point of the ray to the volume.

Koyamada [4] describes a fast method to traverse unstructured regular grids with tetrahedral cells. The ray-cell intersections are found by making use of the connectivity information. To find the first intersection of the ray with the volume, he projects and scan converts the external faces. To reduce the number of external faces, only the front facing external faces are processed. Actually he sorts the front facing external cells, and scan converts them one by one. He uses the centroid points of the external faces for sorting which may result in a wrong sort order. Although he reports that such cases are rather rare, the fact that it can generate artifacts is a major penalty for his approach. In his work, he states that better polygon sorting algorithms such as list-priority algorithms [37] can be used to generate high quality images. While scan converting an external face, he casts a ray from each pixel location

that is generated by the current external face, and follows it in the volume. Garrity [36] intersects all the faces with the ray as planes that grow to infinity, and chooses the minimum of the intersections as the exit point. So he tests all the faces of a cell to find the exit point. Koyamada [4] proposes a test that can directly determine if the face is intersected by the ray when applied to a face. So his method tests on the average two faces for each tetrahedral cell to determine the exit point. Moreover, he reports a direct inverse interpolation technique which allows the sampling calculations to be handled in an efficient way. As this algorithm is one of the algorithms that this thesis is directly related, the details of his work are further described in Chapter 3.

Tabatabai *et al.* [38] describe methods to visualize volumes composed of *nonlinear* elements. In his method, a set of linear equations are solved to find the intersection of a ray with a face. Again the first entrance point of a ray into the volume is carried out by considering only the external faces. The algorithm superimposes a 2D mesh that is formed of equal sized pixel regions on the image plane, and the projected bounding boxes of the faces are used to mark the regions as regions that may possibly contain this face. Hence, the ray-face intersection is calculated by only considering the faces in a region. The next cell intersected is found by making use of the connectivity information between cells.

Frühauf [8] presents a new approach for the rendering of irregularly structured volume data. His method is based on casting rays through the computational space on a nonregularly structured grid, instead of conventionally ray casting the physical space. This technique overcomes most of the difficulties in sampling data along rays, since the computational space is regular by definition. His approach is suitable especially rendering of curvilinear grids, but it is not applicable to unstructured grids as there is not a well defined computational space for such kinds of grids.

Projection Methods

Projection methods have also been investigated for unstructured grids. The common idea in all of the suggested approaches is to perform a view dependent

sort on volume elements, and find their contributions on the image plane and composite them. The methods differ either in sorting phase or in composition phase, but unfortunately all of the suggested methods in the literature are approximations.

Max *et al.* [41] present an algorithm to composite a combination of density clouds and contour surfaces. Contribution of each cell to the generic pixel is computed by analytically integrating color and opacity along the line of sight. In their work, it is reported that given a *Delaunay Triangulation* and a view point, a visibility ordering is always defined. In addition, they prove that a depth sort can be simply computed without having to manage the topological information. They use a sorting algorithm called *numerical distance sort* to determine the correct visibility order.

Projective Tetrahedra (PT) technique proposed by Shirley and Tuchman [42] approximates the contribution each cell with a set of partially transparent triangles. This polygon oriented method is faster than the previous pixel-oriented approach as conventional graphics hardware can be exploited. Their algorithm uses the *MPDO* [43] algorithm to determine the back-to-front ordering of tetrahedral cells. After ordering the cells, they classify each tetrahedron according to its projected profile relative to the view point and they find the position of tetrahedra vertices after the projection transformation is applied. Then according to the classification of the tetrahedron, they identify a set of transparent triangles that will be used as an approximation to the contribution of the original cell. They calculate the color and opacity values at the vertices of these triangles, and finally scan convert the triangles using a graphics workstation.

PT algorithm became very popular among projection methods, and lots of work to find better approximations and optimizations on the original algorithm have been developed. Wilhelms and Gelder [44] apply the projective tetrahedra algorithm to curvilinear grids. Note that a cell of a curvilinear grid can be divided into five tetrahedral cells without introducing new sample points, thus preserving the original topology of the data. They also propose a *multi-pass blending* method that reduces the visual artifacts that are a result of applying linear interpolation by graphics hardware instead of exponential interpolation.

Williams [45] introduce a linear-time algorithm in the number of volume elements to be sorted to determine the visibility order of cells. In [10], he shows that his visibility ordering algorithm can be used by PT algorithm, and he reports striking run times. Max *et al.* [46] present an $O(n^2)$ method to sort n arbitrarily shaped convex polyhedra prior to visualization, and use special graphics hardware to utilize scan conversion of polygons. Cignoni *et al.* [47]. describes some optimizations that can be applied to PT algorithm to make it more efficient.

Other than these approaches, Yagel *et al.* [39, 40], propose a fast approximation algorithm based on slicing to render unstructured grids. Their method uses sweep planes *parallel* to the image plane. At each position of the sweep plane, the plane is intersected with the grid. This results in a 2D slice, and the cells intersecting the slice are then scan converted using the graphics hardware in order to obtain an image of the slice. Then this is image composited with the previously accumulated image that resulted from the sweep so far. In their work, they report several optimizations which can speed up the process at high cost of memory requirements. Although the method can produce high quality images at a cost of memory, and is able to handle general polyhedral grids without having to compute the adjacency information, the need that the user must have access to high-performance graphics hardware is a major penalty.

Hybrid Methods

In this section, some of the existing hybrid algorithms for unstructured grids are investigated. It is interesting that the most promising algorithms for fast rendering of unstructured grids fall into this category.

Challinger [48, 49, 7] solves the point location and view sort problems by employing a z-buffer based algorithm. In the former work [48], a cell-by-cell approach is used. First, a y-bucket is used to sort cells with respect to their y coordinates in the projection coordinate system. The algorithm starts from the topmost scanline on the screen, and processes each scanline. An active cell list is created for each scanline using the y-bucket list. The active cells are sorted into an x-bucket with respect to their x coordinates in increasing order. When

processing pixels in the current scanline, an active cell list is created for the current pixel using the x-bucket. In this way, the number of cells to be tested for intersection is reduced considerably. In the latter works [49, 7], a face-by-face approach is employed. The data is supplied as a list of faces instead of cells. The algorithm is based on the conventional scanline z-buffer hidden-surface removal algorithm used in polygon rendering. As the polygons are the primitive elements for the data, the cells in the original data must have planar faces. For cells that have non planar faces, those faces must be divided into smaller planar polygons which approximate the actual face. In this algorithm, instead of casting rays from pixels and finding their intersection with polygons, polygons are projected and rasterized in scanline order onto the image plane. Note that if a ray shot from a pixel intersects a polygon, then the pixel is covered by the projection area of the polygon on the image plane. In the first phase of the algorithm, all polygons are bucket sorted into a y-bucket according to their minimum y coordinate. The algorithm proceeds from one scanline to other scanline on the image plane, and from one pixel to other pixel in the same scanline. An active list of polygons are created for the current scanline using the y-bucket. The active polygons, whose y-extends cover the current scanline, are intersected with the current scanline to determine the edge intersections. The spans created by the edge intersections are sorted into an x-bucket. As pixels are processed in the current scanline, an active span list is created using the x-bucket. The spans are rasterized to generate ray polygon intersections at the current pixel. The distance of each ray-polygon intersection and related information *(e.g. a pointer to the polygon)* are inserted into a sorted linked list, called *intersection list* (referred to here as z-list), which is sorted in increasing distance values. Note that two consecutive ray-polygon intersections in the z-list corresponds to the entry and exit points of ray with the cells. During the composition phase for the current pixel, the z-list is traversed in order, and each pair of consecutive intersections is used to find the corresponding sample points along the ray. These sample points are composited during the traversal of the linked list. Projections of polygons cover consecutive scanlines on the image plane. Hence, the intersection of scanlines with active faces can be carried out by incremental calculations. Each span in the current scanline covers consecutive pixel locations. Therefore, sorting of z-intersections with polygons are avoided as long as the list of polygons intersected by the ray does

not change.

Giertsen [50] utilizes a scan-plane buffer to solve point location and view sort problems. The scan-plane buffer is a 2D array that is used to store information within the plane perpendicular to a scanline on the screen. In this approach, z-dimension is discretized in a sense due to scan-plane buffer. The algorithm proceeds from one scanline to the next scanline on the screen. At each scanline, the intersections of the respective scan-plane with the cells are calculated. The intersection calculations from one scanline to the next are done by incremental calculations using a list of *active cells*, whose y-extend covers the current scanline. The volume elements intersected by the current scan-plane are sliced by finding the edge intersections of faces of volume elements with the current scan-plane. The paper proposes two methods to handle the updating of active cell list, and slicing of cells efficiently. In the first method, all the cells are bucket sorted into a y-bucket using a conventional scanline z-buffer algorithm according to their minimum y value. The second method is more suitable for volumes with large opaque regions. In this second method, the cells are bucket sorted into a z-bucket according their minimum z value. Then, the z bucket is traversed in increasing z order so that the cell slices closest to the screen are inserted into the scan-plane buffer. The algorithm processes the segments before the opaque slice, if the foremost point of the slice is opaque. After slicing the volume elements for the current scanline, each slice is divided into triangles. Then, each triangle is further decomposed into line segments in the z direction. A line segment is stored into the scan-plane buffer location that corresponds to the foremost end of the line segment. A run-length encoding which shows the expected location for the next segment is also stored at the same location. The composition is carried out by processing the line segments in front-to-back order and linearly interpolating the ray along them. In Giertsen's algorithm, the quality of the images and the performance of the algorithm is heavily dependent on the discretization level of scan-plane buffer.

Silva *et al.* [51, 52, 5], extends the Giertsen's work so that the discretization in z direction is avoided. Furthermore, the proposed algorithm, *LSRC (Lazy Sweep Ray-Casting)*, uses many optimizations to generate line segments

along the ray. It utilizes the inter and intra scanline coherencies. Their algorithm sweeps the space with a sweep plane that is orthogonal to the screen *(x-y plane)*, and parallel to the scanlines *(parallel to x-z plane)*. In Giertsen's method, the vertices must be duplicated and stored in an auxiliary storage, but LSRC algorithm maintains an event queue (a priority queue) that stores an appropriate subset of mesh vertices, thus avoiding this extra storage. They define an *extremal vertex* as a vertex such that all of the edges incident to it lie in the closed half-space above or below it with respect to the y coordinate. The identification of extremal vertices is performed by a simple, linear time algorithm. They initialize the event queue with the extremal vertices, prioritized according to the magnitude of their inner product with the *up vector* that represents the *y-axis* in the *View Reference Coordinate (VRC)* system. As the sweep takes place, new vertices are inserted into the event queue, and some vertices are deleted from the event queue when the sweep hits a vertex. The sweep algorithm proceeds in the usual way, processing events in the order they occur. When a vertex is hit, the vertices to be inserted or deleted are determined from the signs of the dot products of the edge vectors out of the hit vertex, v, with the *up vector*. Otherwise, if the sweep does not hit a vertex, then this is an indication that sweep plane has encountered a scanline. So, they drop into a 2D sweep procedure similar to this one to simulate the ray casting process. Their algorithm does not sort all the vertices at once, but instead they take advantage of the partial order information that is encoded in the mesh data structure.

The LSRC algorithm [5] is reported to have the highest rendering speed achieved for unstructured grids. This method has 4 major advantages over Giertsen's:

1. It avoids the explicit transformation, and sorting phase. This also means avoidance from the need to keep an extra copy of the vertices.

2. It makes no requirements or assumptions about the level of connectivity or convexity among cells, so it can handle cyclic data sets where there is not a visible order among the cells, but it's a fact that the performance degrades with high level of disconnectedness and non-convexity.

3. It avoids the discretization of the scan-plane buffer, therefore allowing accurate rendering even for grids cells of which vary greatly in size.

4. It can handle parallel and perspective projections within the same framework.

Chapter 3

Fast DVR of Unstructured Grids

In this chapter, *Koyamada's* algorithm and the new algorithms that are the main contributions of this thesis will be described. In the first section, the data model, the lighting model, major data structures and linear interpolation method that are common to all four algorithms will be described. Then Koyamada's algorithm will be presented in detail, and in the following three sections the new fast algorithms that we propose will be described.

3.1 Preliminaries

3.1.1 Data Model

The data model common to all four subject algorithms that are the main concern of this thesis is based on the tetrahedral cell model[1] [53]. In the tetrahedral model, the concept of external face is the same as described in Chapter 2, that is, an external face is a face that belongs to only one cell, and is not shared by any other cell. Therefore, the set of external faces forms the

[1]Note that the tetrahedral model used in this thesis is slightly different than the one described by Koyamada [53]

boundary of the volume. The faces are triangles, and the internal faces are shared exactly by two cells. The face normal is defined to be oriented outward from the parent cell. The tetrahedral model has three major components:

- A nodal data component, whose entries contain the position coordinates, and scalar data to be rendered.
- A cell topology component, whose entries contain the identifiers of four nodal components (vertices).
- A cell connectivity component, whose entries contain the identifiers of the four cells connected to a cell through its four faces. If no cell is connected through a face, then an external face identifier is put.

In fact, cells in unstructured grids need not be tetrahedral cells, but tetrahedral cells permit 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, which makes resampling phase very efficient. Therefore, if a volume to be visualized contains cells other than tetrahedral cells, then those cells must be divided into tetrahedral cells in a preprocessing step, which is called tetrahedrization. In the literature, there exist algorithms for tetrahedrization of various types of cells. Doi and Koide [54], propose a method for tetrahedrization of cubic cells, generating five tetrahedral cells, in accordance with the rules that guarantee consistency among the faces of adjacent cells in a regularly ordered grid. For unstructured irregular grids, no such rules can be defined. For this reason, it is necessary to consult the state of previously subdivided cells for consistent alignment as stated by Koyamada and Nishio [53].

3.1.2 Lighting Model

In all of the four algorithms the same low-density particle light source model is employed for lighting calculations [4]. This model assumes the volume to be visualized to consist of low-density particle light sources.

All four algorithms use the front-to-back composition schema. The reason for this choice is that this schema allows *early ray termination*. Early ray

termination is an optimization method used by many ray-casting based DVR algorithms. The main idea is to stop following the ray when the opacity of the composited sample along the ray reaches a user defined threshold. For instance, ray-casting algorithms that support early ray termination will render the surface of a volume data much faster than an object-space algorithm, although object-space algorithms are known to be faster than image-space algorithms.

3.1.3 Linear Sampling Method

As discussed earlier, a DVR algorithm should resample the volume from the new viewing direction. After this resampling, the new data (scalar) values are fed to a transfer function to obtain the color and opacity values to be used in the composition step. In general, a scalar value F_X at a point X inside a tetrahedral cell $(ABCD)$ can be interpolated by the following formula:

$$F_X = \frac{[\vec{XD}, \vec{XC}, \vec{XB}]F_A + [\vec{XC}, \vec{XD}, \vec{XA}]F_B + [\vec{XB}, \vec{XA}, \vec{XD}]F_C + [\vec{XA}, \vec{XB}, \vec{XC}]F_D}{[\vec{AC}, \vec{AB}, \vec{AD}]} \quad (1)$$

where

- $[..]$ means a vector triplet product. For instance, $[\vec{AC}, \vec{AB}, \vec{AD}] = (\vec{AC} \times \vec{AB}).\vec{AD}$, and $[\vec{AC}, \vec{AB}, \vec{AD}]/6$ represents the volume of a tetrahedron $(ABCD)$. Hence, $[\vec{AC}, \vec{AB}, \vec{AD}] = [\vec{XD}, \vec{XC}, \vec{XB}] + [\vec{XC}, \vec{XD}, \vec{XA}] + [\vec{XB}, \vec{XA}, \vec{XD}] + [\vec{XA}, \vec{XB}, \vec{XC}]$.
- F_A, F_B, F_C, and F_D are the scalar values at the vertices A, B, C, and D respectively.

Equation (1) is simply inverse distance interpolation of four vertices with respect to point X inside the tetrahedral cell $(ABCD)$. For example, the contribution of vertex A is found by multiplying the value F_A of scalar at A by the ratio of the volume of tetrahedron $(XCDB)$ to the volume of tetrahedron $(ABCD)$. Although the operation $[..]$ gives 6 times of a volume, they are cancelled because the formula is repeated both at the nominator, and at the denominator.

This simple approach estimates the scalar at any point inside a tetrahedral cell using the vertices of it, but this operation should be repeated as many times as new samples are generated inside a cell. Moreover, for a single ray the new sample points are distributed on a line segment which is defined by two end points; the entry point of the ray to the cell, and the exit point of the ray from the cell. In addition, the scalar interpolated at the exit point of a cell can be used as the scalar at the entry point of the next cell. Since the change of the scalar in any direction is linear in a tetrahedral cell, we can make use of this coherence to speed up this interpolation operation. This interpolation method is called linear sampling, and is described below.

Assuming that the ray enters the cell at point P, and exits the cell at point Q, and the values of the scalars are F_P, and F_Q respectively, the scalar at point X along the line segment PQ can be calculated as $F_X = F_P + r\Delta s$. Here, r is the ratio of the length of the line segment PX to the length of the line segment PQ, and Δs is the rate of change of scalar along line segment PQ. Hence, by substituting $\Delta s = F_Q - F_P$, we obtain

$$F_X = rF_Q + (1 - r)F_P \tag{2}$$

Note that Eq. (2) is the inverse distance interpolation formula for two points. So for a sequence of resamplings inside a cell, one must calculate the scalars at the entry and exit points initially. Then using them in Eq. (2), the samples in between can be interpolated very fast. The scalars, F_P and F_Q, at the entry and exit points can be estimated by using the inverse distance interpolation formula for 3 points described in [19].

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 mid-point sampling, a new sample is generated in the middle of the ray-segment formed by the entry and exit points of each cell. This guarantees that a sample is taken in each cell. Although this is a very easy and strong approach, the major drawback is that the transfer function may be variant inside the cell. So one looses this information by only allowing one sample in each cell. The next approach, equi-distant sampling,

overcomes this deficiency by generating samples at fixed intervals of length Δt. So 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 Δt small enough that at least one sample will be taken in each cell [55]. The last one is adaptive sampling [55]. Although this schema generates very high quality images, it is not practical to use because of the large amount of calculations it introduces. In the implementations of the algorithms discussed in this thesis, equi-distant and mid-point sampling schemes are employed. The Δt parameter in equi-distant sampling scheme is chosen empirically.

It is clear that linear sampling method is faster than the inverse distance interpolation method for equi-distant sampling schema especially when Δt is small. On the other hand, linear sampling method may be expected to perform worse than the inverse distance interpolation methods for mid-point sampling schema. However, in all algorithms presented in this work, values needed for linear sampling method are calculated at a very low cost by exploiting the results of the computations performed during the intersection tests. Therefore, linear sampling method performs better than the inverse distance interpolation method even for mid-point sampling schema.

3.1.4 Major Data Structures

The major data structures described in this section are common to all four algorithms implemented. For each algorithm, some modifications to the existing data structures may exist, and they will be described in the related sections.

The first of the data structures is the data structure that stores the tetrahedral cell data. This consists of two large arrays; the *NodeArray* ($\mathcal{NA}$) and the *CellArray* ($\mathcal{CA}$). $\mathcal{NA}$ keeps the position and the scalar values of original data points in the volumetric dataset, and its size is equal to the number of nodes in the original data. $\mathcal{CA}$ is an array, whose entries keep the information about a cell, therefore, its size is equal to the number of cells in the original data. For each cell, the following information is stored:

1. The identifiers of four vertices of the cell, which are indexes to corresponding nodes in $\mathcal{NA}$.

2. The identifiers of four neighbors of the cell through its four faces, which are indexes to corresponding cells in $\mathcal{CA}$. For external faces, -1 is used a sentinel value.

3. The identifiers of the faces of the neighbor cells that each face of this cell is connected to. Each of the four identifiers represents a value between 0-3, and is valid only if the corresponding face is an internal face. 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 information about each node, that is stored in the entries of $\mathcal{NA}$, is kept as *floating point* values. Each node has a *World Space Coordinate (WCS)* component that keeps the original x, y, z coordinates of the node in the 3D Cartesian space. In addition, each node has *Normalized Projection Coordinate (NPC)* component that keeps the transformed, and projected x, y, z values of the *WCS* coordinates. Each node has a *Scalar Value (S)* component that represents the value of the sample at this point. So totally, 7 floating point values are stored for each data point, which makes a total of 28 bytes in the environment that those algorithms are implemented.

Three components are stored for each cell in $\mathcal{CA}$. The first component which denotes the connectivity relation between this cell and its neighbors is an array of 4 entries, and each entry is an integer in the range -2^{31} to $2^{31} - 1$. The second component that stores the indexes to the 4 vertices of a cell similarly occupies 4 integers. The third component needs 2 bits for each face, so it totally occupies 8 bits, which is 1 byte effectively. So total number of bytes used for storing information of each cell is 33 bytes.

If the number of nodes is denoted by N, and number of cells is denoted by C, then the total space occupied by the $\mathcal{NA}$ and $\mathcal{CA}$ data structures can be found using the following formula

$$\mathcal{M}(\mathcal{NA}, \mathcal{CA}) = 28N + 33C.$$

The second major data structure common to all four algorithms is the *Screen* data structure. It is a 2D virtual array that corresponds to the image on the screen. In each entry of this 2D array, a linked list of rays is stored that symbolizes the rays to be shot from the pixels on the screen to the volume. The elements of the linked lists store the following information:

- A *Next* component which is a pointer to the next item in the linked list.
- A *ZDepth* component which denotes the z coordinate of the entry point of the ray, shot at this pixel, into the volume. The lists are maintained in sorted increasing order according to this component.
- A *SValue* component which denotes the *scalar* value at the intersection point of the ray with the face.
- An *EntryFace* component which identifies the external face by its cell index, and face index in the cell that generated this ray.

The size of the Screen data structure is dependent on the resolution of the image to be generated. For each pixel on the real screen, it stores a pointer (4 bytes) to the first item of the linked list. If no ray is generated for this pixel, then the pointer is set to *NULL*.

The lengths of the ray lists generated are completely dependent on the data to be visualized and the viewing parameters. If the rate of non-convexity is high for the chosen viewing parameters, then this means that a ray shot at a pixel will enter, and exit the volume many times, generating many ray-segments in this kind of an implementation, but our observations show that this case is rare for most of the datasets. Note that the length of each ray list in the *Screen* structure will be either 0 or 1 for convex datasets. The *Next* component in a list item is a pointer and occupies 4 bytes. Both *ZDepth* and *SValue* components are floating-point values and occupies 8 bytes in total. The *EntryFace* component stores a cell index which is an integer, a face index requiring 2 bits which can be stored in a byte totally occupying 5 bytes. Hence, 17 bytes are used for storing information for each ray generated. If the number of rays generated is denoted by R and the area of the screen in terms of pixels is

denoted by A, then the total number of bytes spend for the *Screen* component can be found using the formula below

$$\mathcal{M}(\mathcal{S}) = 17R + 4A.$$

All algorithms mentioned in this thesis involve large amounts of *dynamic memory management (DMM)* operations. In order to avoid the overheads to be introduced by operating system level DMM routines, we have implemented our own fast DMM modules.

3.2 Koyamada's Algorithm

Koyamada's algorithm is a ray-casting approach that makes use of the coherence in the image-space to generate rays, and follows those rays in the object-space. Therefore, the first step of his algorithm generates the rays to be traced. In his original algorithm, he sorts the 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 [4]. As the algorithms presented in this thesis aim at producing high quality, 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 and for each pixel covered by the projection area of an external face, we insert a list item into the appropriate ray list in the *Screen* data structure that represents a ray. The *ZDepth*, *SValue* and *EntryFace* components are set appropriately during the scan conversion process. Although Koyamada's approach of first sorting the external faces seems more space efficient, it becomes slower for data where the rate of non-convexity is low with respect to the approach presented here. Yet, an algorithm which is similar to the one proposed by Challinger [48, 49, 7] can guarantee the correct sorting of external faces which is expected to perform better if the rate of non-convexity is high, and this enhancement is discussed in Section 3.6.1. By this way, the *Screen* component will be free of linked lists, and it will be used like a z-buffer in conventional polygon rendering algorithms. As the speed of the rendering process is the main concern of this thesis, we rely on the assumption that rate of non-convexity is low in the data.

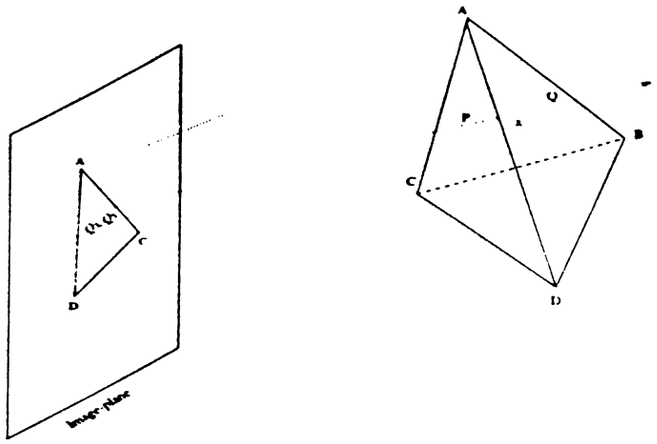


Figure 3.1: Shooting of a ray from image-plane, and following it in the volume using Koyamada's algorithm

After the rays are created, 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 intersected 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, the values at the point that the ray first penetrates into the volume is determined and stored during the scan conversion of front faces of external cells, all that is needed is a fast way to determine the exit face, and the values of the exit point from the cell. So the problem of tracing a ray inside the volume reduces to the problem of determining the exit point from a cell, when 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. Let (x_Q, y_Q) denote the position of point Q in NPC as in Fig. 3.1, which is, in fact, the coordinates of the pixel that the ray is shot from the image-plane. Let triangle CDA denote the projected triangular area of a face of a cell that will be subject to the ray-face intersection test. If the 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 must be tested. On the other hand, if the vectors are not parallel, then the vector AQ can be expressed as a linear combination of them:

$$\vec{AQ} = s_Q\vec{AC} + t_Q\vec{AD} \tag{3}$$

where s_Q and t_Q are weighting values, and can be found solving the following matrix system

$$\begin{bmatrix} x_C - x_A & x_D - x_A \\ y_C - y_A & y_D - y_A \end{bmatrix} \times \begin{bmatrix} s_Q \\ t_Q \end{bmatrix} = \begin{bmatrix} x_Q - x_A \\ y_Q - y_A \end{bmatrix} \tag{4}$$

where (x_C,y_C), (x_D,y_D), (x_A,y_A), and (x_Q,y_Q) are the coordinates of points C, D, A, and Q in NPC, respectively. If s_Q and t_Q satisfy the following conditions:

$$\begin{aligned} s_Q &\geq 0 \\ t_Q &\geq 0 \\ s_Q + t_Q &\leq 1 \end{aligned} \tag{5}$$

then the point Q is inside the triangle CDA, so no further tests need to be done on other candidate faces. Otherwise, the point Q is not inside the triangle, so another face must be checked. The conventional approach always checks 3 faces [36], but this method checks 2 faces on the average.

For the sake of simplicity and efficiency in the implementation, we have taken *View Reference Coordinate System (VRC)* and NPC System the same, so the z coordinate of the exit point Q, can be determined by using the Eq. (3) for the z components as

$$z_{AQ} = s_Q z_{AC} + t_Q z_{AD}. \tag{6}$$

As the exit face and the coordinates of the exit point have been determined correctly, the scalar value, F_Q, at the exit point Q can be estimated using the following inverse distance interpolation formula for 3 points:

$$F_Q = s_Q F_C + t_Q F_D + (1 - s_Q - t_Q)F_A \tag{7}$$

where s_Q is the ratio of area of triangle QDA to area of triangle CDA, t_Q is the ratio of area of triangle QAC to area of triangle CDA, and the value $(1 - s_Q - t_Q)$ is the ratio of area of triangle QCD to area of triangle CDA. The samples along the ray-segment PQ can be taken and composited using the linear interpolation method discussed previously.

3.3 The Span-Buffer Ray-Casting Algorithm

Koyamada's algorithm is an efficient ray-casting algorithm since it exploits object-space coherency by utilizing the connectivity relation between cells while following the ray. However, it does not exploit image-space coherency in ray-face intersection tests for internal cells. It is clear that a ray shot from a pixel on the screen is likely to pass through the same faces with the neighbor rays. So in fact, the algorithm performs operations that will give similar results over and over without considering this similarity. The proposed *Span-Buffer Ray-Casting (SBRC)* algorithm exploits image-space coherency for both internal and external cells without disturbing the use of object-space coherency.

The first version of the algorithm was called *Matrix-Buffer Ray-Casting (MBRC)* algorithm. The idea was simply to follow the rays just as in Koyamada's approach using the connectivity relation. The algorithm was inspired by the observation that a ray intersects a face if and only if the pixel, that the ray is shot from, is covered by the projection area of that face. So the problem is to find an efficient way to store the scan converted information of faces of a cell. The easiest solution is to create a 2D matrix whose sizes are equal to those of the bounding box of the projection area of the face. As we know to which pixel the starting point of the matrix corresponds to, the mapping of a ray shot at position (x, y) to the matrix-buffer is straight forward. The process of creating a temporary buffer to keep the scan converted information, and storing this information by scan converting the back faces of a cell is called *Activation* of a cell, and a cell that has a buffer in the memory is called an *active cell*. When a cell is activated, the bounding box of its projection area is calculated, and a matrix-buffer of that size is created. Then the back faces of the cell are scan converted to this matrix-buffer just like projecting them on the screen. For each position that a face is scan converted, three items are written to the corresponding entry of the matrix-buffer; the face identifier of the face that generates this pixel, the interpolated scalar value, and interpolated z coordinate value of that face. It is clear that not all the cells must be active during the rendering process. If the rays are shot, starting from the top scanline proceeding to the next one down, and inside each scanline from left to right, then the active life of a cell starts with the first ray that hits the cell, and

ends with the last ray that intersects the cell. The last ray that will intersect a cell is the ray shot from the rightmost bottom pixel covered by the projection area of a cell. So we can activate a cell the first time a ray intersects the cell by generating its matrix-buffer, and kill it when the last ray passes through the cell by freeing the memory occupied by its matrix-buffer.

Supporting early ray termination in MBRC algorithm needs special attention in matrix-buffer deallocation. We can never be sure that the last ray that will intersect the cell will travel enough in the volume to hit the cell, so that the cell is killed. This problem can be solved in three ways. The first one is to trace the ray until it exits the volume ignoring the sampling process after the ray reaches the threshold opacity. Using this method, the early ray termination is supported partially. The second approach is to let the matrix-buffers reside in the memory until the memory chunk allocated to store the matrix-buffers of active cells is full. When there is no more space to allocate a new matrix-buffer, one can traverse all the cells in $\mathcal{CA}$ to determine which cells can be killed. The third approach can be implemented by employing a y-bucket data structure. After the allocation of the matrix-buffer for a cell, its pointer is inserted to the y-bucket list corresponding to its bottom y coordinate. After processing a scanline, the matrix-buffers the pointers of which are in the respective y-bucket list can be deallocated.

The problem with the MBRC approach is that the matrix-buffers generated for each cell will occupy too much space, and as the projection areas of the cells will not fill the matrix-buffers completely, there will be some excess space spent unnecessarily. So for memory efficiency, a better and economical way to keep the same information must be developed. Another disadvantage of the MBRC algorithm is that scan conversions of the faces of cells incur multiple rasterizations of their shared edges. The SBRC algorithm is proposed to solve the problems of the MBRC algorithm in an efficient way. In this new version, the main objective is to use only the space necessary to keep the scan conversion information of cells without spending any excess memory, and scan convert each edge only once to exploit the image-space coherency at the maximum rate.

SBRC algorithm necessitates the addition of extra data structures and some modifications to the existing ones. First of all, a hash table is needed. This hash

table will be used for storing and retrieving the data necessary for active edges. An *edge* is defined as the line connecting two data points in the grid. It can be identified uniquely by the two indices of its endpoints to the corresponding entries in the $\mathcal{NA}$ structure. This means that the two indices of its endpoints can be used as a unique key to a hash table to determine if the edge is in the table, and if it is there, to retrieve it; and if it is not there, to insert it into the respective slot in the hash table. For each active edge, the following data should be stored in a structure called *Edge*:

- The bottom y coordinate of this edge. If the current scanline is below this value, then this edge will never be needed in the rendering process, so it can be removed from the hash table.
- The (x, z, s) tuple corresponding to the current x coordinate, z coordinate, and the *scalar value* of the edge. They are the values at the intersection point of the edge with the current scanline.
- $\Delta x, \Delta z$, and Δs values which correspond to the amount of incremental change, along the edge, in x, z, and s values, respectively, when proceeding from one scanline to the next.
- The unique key, that consists of the two indices of its endpoints to the corresponding entries in the $\mathcal{NA}$ structure, to identify the edge.

This means 6 floating-point, 2 integer and one short integer values that occupy 34 bytes in the environment that this software was developed. Hopefully, not all of the edges will be active at the same time. Moreover, the experiments show that at most 10% to 15% of the edges become active at the same time. This nice property also allows one to determine the size of the hash table empirically.

In addition to the hash table and the *Edge* structure, we have to insert a structure called *EdgeInfo*, which keeps the local information about the edges of a cell to each entry of $\mathcal{CA}$ structure. As not all of the cells will be active at the same time, placing the *EdgeInfo* structure directly in the entries of $\mathcal{CA}$ will be a waste of space, so a pointer called *EdgeInfoBuffer* is inserted instead

of the whole structure. It is set to *NULL* when the cell is inactive, and points to the appropriate *EdgeInfo* structure when the cell is active. This additional pointer introduces a 4 byte to each entry in the *CA* structure. The *EdgeInfo* structure is defined as below:

- *NumEdges* field which keeps the total number of edges of the cell that will be necessary for the scan conversion of the back faces of this cell. This number can be at most 6 for a tetrahedral cell, so 1 byte is enough for this field.
- An array of size 6 to keep the topological order of the indices of the edges, which is called *EdgeOrder*. Although only the first *NumEdges* elements of this structure will be valid, we have to allocate space for the maximum case. As each index is in the range of 0-5, 3 bits is enough for each entry. So this structure can be simulated using 18 bits.
- An array of size 6 to keep the local status of each edge, which is called *EdgeStatus*. There are three status; *Inactive*, *Active*, and *Killed*. So 2 bits for each edge is needed which makes a total of 12 bits.
- *LastScanline* field which keeps the y coordinate of the scanline that the last span-buffer was created for. During the execution of the algorithm, this value is used to decide whether the current span-buffer is valid for this scanline, or a new span-buffer must be created. It is a short integer taking 2 bytes.
- *SpanBuffer* field which is a pointer to the current span-buffer for this cell. It takes 4 bytes.

Packing the *EdgeOrder* and *EdgeStatus* components to one compact structure, 4 bytes is enough to store them. Four bytes come from the *SpanBuffer* field. We can pack the *LastScanline* and *NumEdges* fields into 2 bytes by using the leftmost 3 bits of the field for the *NumEdges* component, and using the other 13 bits to store the *LastScanline* component. This constitutes a total of additional 10 bytes for each active cell.

The next structure introduced by the SBRC algorithm is the *MainBuffer* structure which is a large piece of memory chunk that will be used to store the

span-buffers. It is manipulated by our code. There is a pointer, *EmptyBuffer*, which always shows to the first empty location in the *MainBuffer*. When a request for a span-buffer comes from a cell, it sends the address of this first empty location as the starting location for the span-buffer. The cell, after writing the information for a span starting from this location, returns the number of bytes written, and the *EmptyBuffer* pointer is advanced to the next empty location in *MainBuffer* by using this information. When a scanline is processed, the *EmptyBuffer* pointer is set to point to the beginning of the *MainBuffer* to overwrite the previous span-buffers with the new span-buffers to be created for this scanline. As there are no memory allocations, and free operations at the operating system level, this implementation is very efficient for the buffer of span-buffers. The size of this structure is defined by the user. It must be large enough to store all the span-buffers created for one scanline. The suitable sizes for different data will be experimentally investigated in the Chapter 4.

As in the MBRC algorithm, the problem of determining the time to deallocate the *EdgeInfo* structure for an active cell due to early ray termination exists in the SBRC algorithm. In our implementation, we have chosen the third of the three techniques presented in this section formerly to solve this problem. That is, we use a y-bucket structure which has a size equal to the height of the screen. The pointers to *EdgeInfo* structures for the activated cells are inserted into the y-bucket list stored at the position that corresponds to the bottom y coordinate of the bounding box of the projection area of the cell on the screen. When the respective scanline is processed, the pointers to *EdgeInfo* structures in the list are freed. This structure is referred to here as *YBucket*.

In SBRC algorithm, the rays must be shot starting from the topmost scanline proceeding to the next one down, and inside a scanline from left to right. When a ray hits a cell the first time, it is activated just like in the MBRC algorithm, but the process of activation is slightly different. While activating a cell, the first thing to do is to request an *EdgeInfo* structure and set its *EdgeInfoBuffer* component so that it points to the new *EdgeInfo* structure. Then, we need to identify the edges necessary for the scan conversion of back faces of the cell. A tetrahedral cell has 6 edges, and according to the view point at

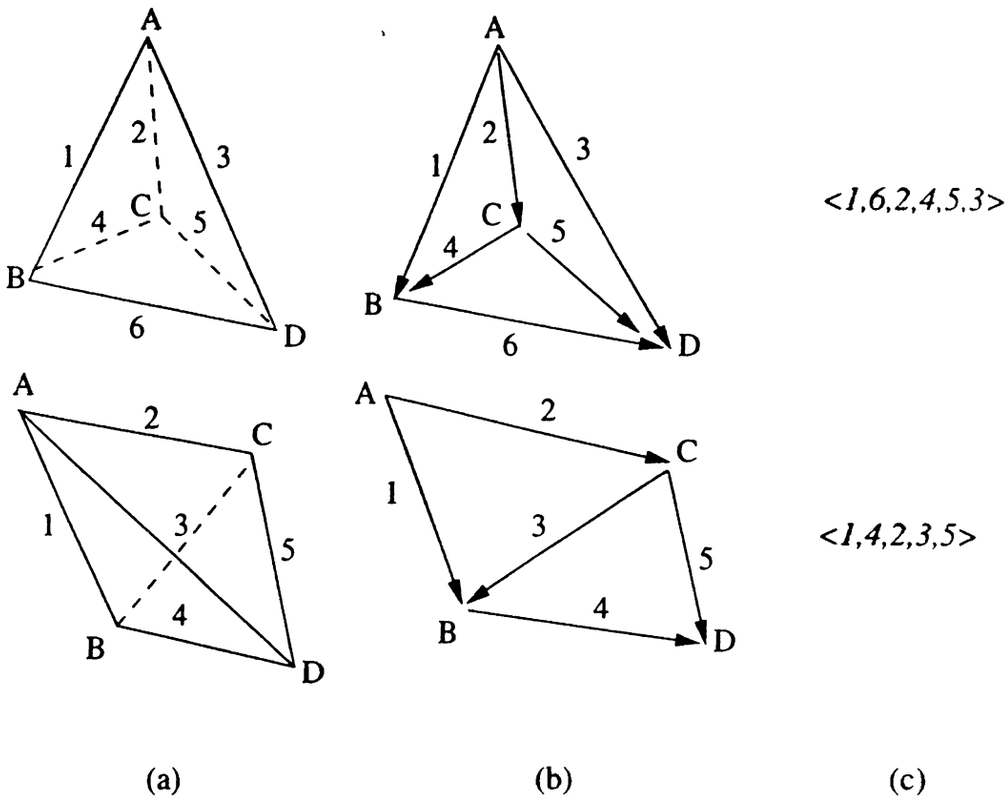


Figure 3.2: (a) Two sample cells projected onto the screen with edge numbering (Note that only back-face edges are numbered). (b) The corresponding directed graphs where depth-first search is performed starting from the topmost vertex A. (c) The correct back-face edge orderings.

least 2, and at most 6 edges might be necessary for the activation process. It is obvious that the edges that belong to the back faces, are the edges that we are looking for. Such edges are called *back-face edges*, and they can be defined as edges belonging to at least one back-face of the cell. The two vertices that form a back-face edge are called *back-face vertices*. As the necessary edges are identified, we set the *NumEdges* field of the corresponding cell. Then, we perform a kind of sorting operation on the back-face edges of the cell. The aim of this sorting is to find an order on the edges such that when they are cut by a virtual line parallel to a scanline, the intersection points will always appear sorted in increasing order with respect to their x coordinates. This is like a topological order on the edges that preserves the dependency of them when they are viewed from the left side of the screen. This sorting operation is performed only once when the cell is activated. This edge sorting operation

is performed by using a *depth-first search* based algorithm as follows: We construct a directed graph whose nodes correspond to the back-face vertices of the cell. For each back-face edge AB of the cell, if $y_A < y_B$ ($y_B > y_A$), then a directed edge from node A to B (B to A) is added to the graph. Here, y_A denotes the y coordinate of a back-face vertex A of the cell in NPC. Then, outgoing edges from each vertex in the graph are sorted according to their angles with respect to the x *axis* (parallel to a scanline) in NPC. Note that this sorting operation is necessary to ensure the traversal of outgoing edges of a vertex in the graph in left to right order. A depth-first search initiated from the node with the highest y coordinate in NPC which traverses the outgoing edges of a vertex in left to right order directly gives the desired topological order on the back-face edges. Figure 3.2 illustrates two cases for the sorting operation. After the sorting operation, *EdgeOrder* keeps the sorted order of the indices of the edges which implicitly identify the edges, and the status of the necessary edges are all set to *Inactive* in the *EdgeStatus* structure. After the activation, a span for this scanline must be created. The creation of span-buffers is exactly the same for each scanline. There are three possibilities for each edge to be used in the scan conversion of the back faces of a cell:

- Both the start, and end points of the edge are above the current scanline, so this edge will never be used by any cell including this cell, so the status of this edge must be set as *Killed*.
- Both the start, and end points of the edge are below the current scanline, so this edge will be used later by this and may be some other cell, so the status of this edge must be left as *Inactive*.
- The start point is above the current scanline, and the end point is below the current scanline, so this scanline intersects this edge, then the status of this edge must be set to *Active*.

The edges with status *Killed* are ignored directly. The edges with status *Inactive* may pass to either *Killed* or *Active* state. If it passes to *Active* state then it is tried to be inserted in the hash table. If it is already there (inserted by some other cell activated before which shares this edge), then the pointer to its *Edge* structure is returned; if it is not there, then an *Edge* structure is

allocated and it is filled by the current and incremental values of the subject edge. The key field of the new *Edge* structure is set by using the indices of the two vertices that form the edge and it is inserted into the hash table. The edges with status *Active* may pass to *Killed* state, or may be left in *Active* state. If an edge is left in *Active* state, then the pointer to its *Edge* structure is also retrieved from the hash table. After this update of edge status and retrieval of the pointers to *Edge* structures, we now have a sorted list of the intersection points of the current scanline with the active back-face edges of this cell. Then a span-buffer is requested, and the following information is written into the span-buffer:

- *StartX* field keeps the x coordinate of the leftmost pixel of the respective span generated. This is used in mapping the x position of a ray to the appropriate position in the span-buffer. It is a short integer that occupies 2 bytes.
- Following the*StartX* field, a burst of scan converted pixel information is written. This information includes the face identifier of the face that generates this pixel, the interpolated scalar value, and interpolated z coordinate value of that face. For each pixel, this necessitates 2 floating-point values, and 1 byte, making a total of 9 bytes. This structure is called the *CellBuffer*.

After the span-buffer is filled, the number of bytes written is returned to update the *EmptyBuffer* pointer so that it points to the first empty location in the *MainBuffer* structure. The *LastScanline* field of this cell is set to the y coordinate of the current scanline as an indication that the information in the span-buffer is valid only for this scanline. Figure 3.3 shows a sample case of following a ray.

As the span-buffer for the first scan-line that intersects this cell is created, we can read the values from the created span-buffer to determine the exit face, and exit point of the ray from this cell. For mapping the x coordinate of the ray to the correct position in the span-buffer, we just subtract *StartX* value from the x coordinate of the ray. The difference can be used as an index in the *CellBuffer* structure to the corresponding position of the original pixel on

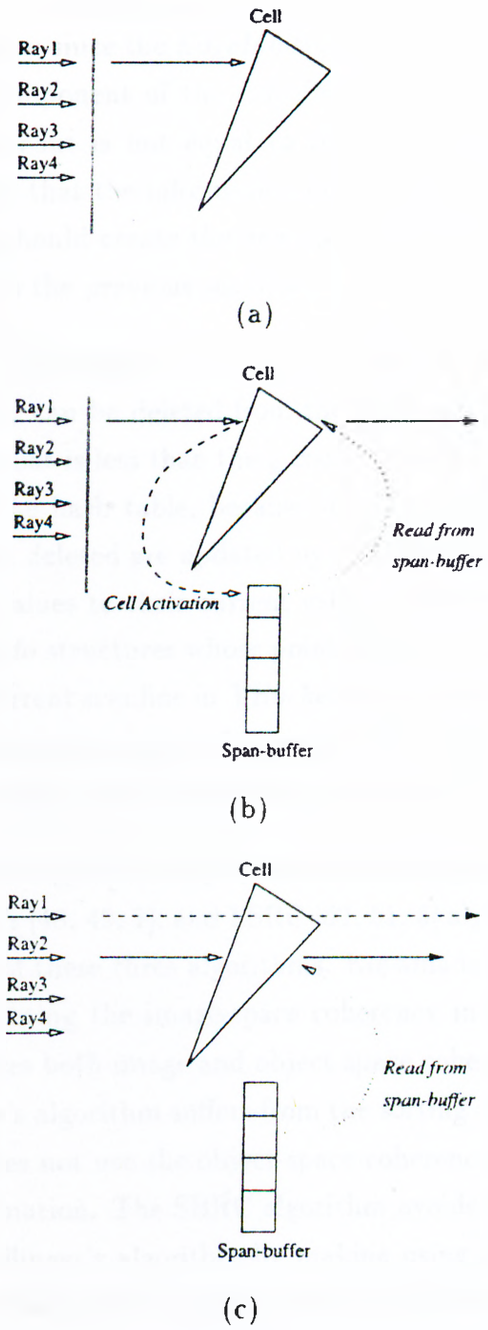


Figure 3.3: A sample case of following a ray in SBRC algorithm. (a) *Ray1* hits the *Cell*, and activates it. (b) *Ray1* reads the exit values from the span-buffer and continues in the volume. (c) *Ray2* hits the *Cell*, and directly reads the values from the span-buffer.

the screen. Then, the ray is followed just like in Koyamada's algorithm. In the next scanline, when another ray hits the same cell, we understand that the cell was activated before, since the *EdgeInfoBuffer* field is not *NULL* anymore. If the *LastScanline* component of the *EdgeInfo* structure pointed to by the *EdgeInfoBuffer* component is not equal to the y coordinate of the current scanline, then we know that the information in the span-buffer is not valid for this scanline, and we should create the new span-buffer for this scanline, and it is created just like in the previous scanline.

When a scanline is processed, all the valid entries in the hash table are traversed to see if they can be deleted from the hash table. Any edge, which has a bottom y value that is less than the y coordinate of the current scanline, can be deleted from the hash table, because it will never be accessed again. The edges that are not deleted are updated by adding the incremental change values of x, z, and s values to their current values. After the traversal of the hash table, the *EdgeInfo* structures whose pointers are stored in the respective y-bucket list for the current scanline in *YBucket* structure are deallocated, and the elements of the respective y-bucket list are deleted. The algorithm proceeds in this manner downwards until all the scanlines are processed.

The proposed SBRC algorithm may be viewed as being the hybrid of Koyamada's [4], Challinger's [48, 49, 7], and LSRC [52, 51, 5] algorithms. It tries to exploit the best sides of these three algorithms. Koyamada's algorithm suffers from the lack of exploiting the image-space coherency in the data, whereas SBRC algorithm utilizes both image and object space coherencies at the maximum rate. Challinger's algorithm suffers from the sorting phase of spans, and buckets, because it does not use the object-space coherency available through the connectivity information. The SBRC algorithm avoids the extensive sorting operations in Challinger's algorithm by making using of the connectivity relation in the data. The LSRC algorithm uses the coherence at a rate close to the rate used by this algorithm by exactly scan converting each edge once, but the sorting operations to be performed in each of the 3D and 2D sweeps, and overheads introduced by the management, and maintenance of priority heaps and events degrade the performance of the algorithm. The only disadvantage of the SBRC algorithm is that a tight bound on the size of the memory

used by *MainBuffer*, *Edge*, and *EdgeInfo* structures can not be determined a priori. However, this problem can easily be solved by using *dynamic table scheme* supporting *table expansions* [56], other than that we expect to reach the fastest rendering times for unstructured grids among pure ray-casting based algorithms which produce correct images, and can handle cyclic data sets. Especially for data where the cells are large and for high screen resolution, the SBRC algorithm is expected to perform much better than the existing DVR algorithms, because as the projection areas get larger in terms of the number of pixels covered, the benefit of using the image-space coherency by scan conversion increases, and overheads introduced by the hash table operations and activation process of a cell stay nearly constant.

3.4 Hybrid SBRC Algorithm

In the SBRC algorithm, for each cell that is intersected by at least one ray, the activation operation, which is not a cheap operation as it involves the identification of back-face edges and a topological sort on these edges, is performed. For cells whose projection areas occupy a small number of pixels, the benefit of trying to use the image-space coherency in the data by scan conversion might be suppressed by the overhead of the activation process, and the hash table operations such as retrieving, inserting, deleting or updating an edge. So one can suggest a hypothesis which states that Koyamada's algorithm can beat this algorithm for visualizations where there is a large number of cells with small projection ares. Hopefully, the similarity between Koyamada's and the SBRC algorithms allows the emerging of a new hybrid method called the *Hybrid Span-Buffer Ray-Casting (H-SBRC)* algorithm that blends the two approaches. The main idea is to use Koyamada's algorithm to render a cell when the cost of using the SBRC algorithm to render the cell is more costly. Also, this approach will reduce the space complexity of the SBRC algorithm by not activating each cell intersected by a ray; therefore, reducing the amount of space needed to store *MainBuffer*, *Edge* and *EdgeInfo* structures since number of active cells directly affects the size of the memory needed to store them. Another important benefit is that this algorithm can be used in cases where a

large memory is not available. For example, when the *MainBuffer* structure is full, and there are still some cells to be processed, Koyamada's algorithm can be used to process the rest of the cells in a scanline.

In order to decide which algorithm should be employed to process a cell, we should develop a *bias*. If we can estimate the amount of time to be spent to render a cell by each of the two algorithms with a small error, then the algorithm with the smaller rendering time should be used to render the cell. So we have to determine, somehow, the amount of time that each of the algorithms will spend to render a cell. The execution time of Koyamada's algorithm can be divided into four parts: *time spent for transforming nodes from WCS to NPC, time spent for generating ray-segments by scan converting front-facing external faces, time spent for sampling operations*, and *time spent for intersection tests of ray-segments with the faces of cells*. So we can write the following formula that approximates the time for rendering a dataset as:

$$T_{Koy} = \alpha R + \beta S + \gamma N + \delta I_T \tag{8}$$

where R is the number of ray-segments generated, S is the number of samples taken, N is the number of nodes in the data, I_T is the number of intersection tests performed, and α, β, γ and δ are respective costs associated with each of these operations. The main operations performed in the SBRC algorithm that affects the execution time can be listed as: *time spent for transforming nodes from WCS to NPC, time spent for generating ray-segments by scan converting front-facing external faces, time spent for sampling operations, time spent for activation of cells, and the time spent for creating spans*. The time for creating spans can be divided into two parts as: *time spent for scan conversion to create a span and time spent for initializing the scan conversion process that involves inserting, and retrieving edges from the hash table*. So the formula that approximates the rendering time of a dataset by SBRC algorithm can be written as follows:

$$T_{SBRC} = \alpha R + \beta S + \gamma N + \kappa C + \lambda I + \mu H \tag{9}$$

where R denotes the number of ray-segments generated, S denotes the number of samples taken, N denotes the number of nodes transformed, C denotes the number of cells activated, I denotes the number of ray-face intersections, H

denotes the number of spans created, and α, β, γ, κ, λ and μ denote the costs of the respective operations. We can ignore the costs for ray-generation, node transformation and sampling because the same routines are employed in both of the algorithms. Consider a cell c with a projection area of a_c, and height (number of spans) h_c. Then, the expected execution cost of each algorithm for cell c can be written as follows:

$$T_{Koy} = \delta i_c \approx 2\delta a_c \quad (10)$$

$$T_{SBRC} = \kappa + \lambda a_c + \mu h_c \quad (11)$$

where i_c denotes the expected number of intersection tests to be performed on cell c by Koyamada's algorithm. As also mentioned earlier, Koyamada's algorithm is expected to perform 2 intersection tests per intersection on the average. Therefore, i_c is approximated by $2a_c$ for cell c in Eq. (10). So if $2\delta a_c \leq \kappa + \lambda a_c + \mu h_c$, then we should use Koyamada's algorithm, otherwise we should use SBRC algorithm to process the cell.

As now the bias is defined, the only problem is to find a_c, and h_c in a fast way. Two heuristics have been developed to approximate a_c, and h_c:

- *Exact Area Heuristic (EA)*: This heuristic tries to estimate a very close approximation to the number of pixels occupied by the projection area of a cell on the screen. To do this, it calculates the areas of all faces that belong to the cell in NPC system, and sums them up. The value obtained should be divided into two because originally it is twice as the original coverage area, as both front and back faces are used.

- *Bounding Box Area Heuristic (BBA)*: This heuristic uses half of the area of the bounding box of the projection area of a cell as an approximation to a_c. As a tetrahedral cell either forms a triangle, or a four sided polygon when projected onto the screen, it is very easy to calculate the area of the bounding box of the projection in NPC.

Both of the heuristics can calculate h_c exactly by finding the height of the projection of the cell in NPC. It is clear that EA will estimate a_c more accurately, but it must perform more operations with respect to BBA. On the

other hand, *BBA* is expected to overestimate a_c, which in turn will result in favoring SBRC algorithm in cases where the difference between projection area and the bounding box area of a cell is large. The results obtained from both of the heuristics will be discussed in Chapter 4.

The H-SBRC algorithm introduces an overhead of taking a decision on the algorithm to be employed for each cell that is intersected by at least one ray-segment. So if the bias chooses Koyamada's algorithm for all cells intersected by at least one ray-segment, then this will result in a larger execution time than the execution time of Koyamada's original algorithm. On the other hand, if the bias chooses SBRC algorithm for all cells intersected by at least one ray-segment, then this will result in a larger execution time than the execution time of the original SBRC algorithm.

3.5 The Melting Cells Algorithm

The third algorithm proposed in this thesis is the *Melting Cells (MC)* algorithm. The MC algorithm falls into the *Projection* methods category of DVR methods. As the name implies, the algorithm runs as if the cells are *melting*, and falling onto the screen one at a time. If we imagine the volume data as a block of tetrahedral ice cells and the screen as a heater, then the cells that are closest to the screen will melt first. After a cell melts, then can melt the cells behind it. Although the idea is similar to other projection methods [25, 41, 42, 33, 30, 57, 10, 45], this method produces images of the same high quality with other ray-casting based methods.

The projection methods require a visibility ordering among the cells for rendering. A visibility ordering of a set of objects with respect to a view plane is an ordering such that if object a obstructs object b, object a precedes object b in the ordering. If we can find such an order among the cells, then we can process them one by one by projecting them on the screen in the visibility order, because we are sure that once a cell is projected onto the screen, no other cell which is closer to the screen whose projection area on the image-plane intersects with the projection area of this cell, will be projected after this cell. Many

algorithms to solve this problem exist in the literature [45]. For a convex set of connected convex polyhedron, this problem is the simplest to solve, because one can exploit the connectivity relation, as a partial information, among the cells to determine the correct order in linear time. The problem gets harder for a non-convex set of convex polyhedron, because in this case the cells need not be connected which brings an overhead of sorting the cells at the boundary of the volume. Furthermore, in some cases the sorting is impossible, especially in sets of convex or non-convex grids where the cells are non-convex. For example, if there is a cycle among the cells such that a obstructs b, b obstructs c, and c obstructs a then such an ordering does not exist. For the last case, one of the solutions is to regenerate the mesh by applying a *Delaunay Triangulation (DT)* algorithm. A DT of a set of points in E^3 is a triangulation such that a sphere circumscribed about the four vertices of any tetrahedron in the triangulation contains no other points in the set. DT can be computed in $O(n \log n + k)$ time [58]. The tetrahedron generated tend to be well behaved, and not skinny. Thus, DTs are conveniently generated from point sets and they are useful for generating unstructured grids. A nice property of DTs is that the tetrahedron generated are cycle free [59], therefore, the set of cells can always be ordered in front-to-back order [60]. So the sorting algorithm employed in the MC algorithm assumes the data to be a non-convex set of convex tetrahedra where there are no cycles. If there are cycles in the original data, then one can perform a DT over it, as a preprocessing step only once, to generate a similar, acyclic dataset.

The MC algorithm starts by scan converting the front facing external faces to generate ray segments in sorted order just like in the previously mentioned three algorithms. The rest of the algorithm consists of two phases; a preprocessing phase for *visibility ordering* and the *rendering* phase. In the first phase, the information to be used in constructing the visibility order among the cells in the volume data is constructed, and in the second phase the cells are processed, while the visibility order is constructed gradually, for resampling and composition of the samples.

The MC algorithm necessitates the addition of following data structures. For each entry of $\mathcal{CA}$, the following field is added:

- *InDegree* field which keeps a number that denotes the number of obstructions for this cell in the visibility order. If it is 0, there are no obstructions for this cell, so it can be processed safely.

The following component is embedded into the ray-segment data structure:

- *Sample* component which keeps a 4 tuple (R, G, B, ρ) as accumulated values of the 3-color channels red, green and blue, and the *opacity*, respectively.

The *InDegree* field is stored in a short integer occupying 2 bytes. The *Sample* component keeps 4 floating-point values occupying 16 bytes. So in the MC algorithm, 2 bytes of additional memory space is necessary for each cell, and 16 bytes of additional memory space is necessary for each ray-segment to be generated. The other data component to be added is a queue. It can be either *FIFO* or *LIFO*. If it is a *LIFO* queue, then the cells are processed in *depth-first* order, if it is a *FIFO* queue then the cells are processed in *breadth-first* manner. This component is called *CellQueue*, and is implemented as a circular queue on a linear array in our implementation for the sake of memory efficiency. Each item of the *CellQueue* is a 4 byte integer value, which is just the index of the cell into the CA structure. As the cells are inserted and deleted continuously, this structure need not be so large.

In the MC algorithm, a visibility order with respect to a view plane is found by using a topological sorting schema which minimizes both the additional memory requirement and the execution time. It uses the concept of *dependency* among the cells to construct a visibility order. A cell a is *dependent* on another cell b, if cell b obstructs cell a in the visibility order. In other words, if a is dependent on b, then b must be processed before a is processed. We define 2 kinds of dependencies referred to here as *Internal* and *External* dependencies. An internal dependency occurs between each pair of neighbor cells which share a face. An external dependency may only occur between a pair of external cells, when the projection areas of their front-facing external faces overlap. If an internal dependency exists between a pair of cells in the data, then this dependency will always exist, but its direction may change with varying viewing

parameters. However, in the case of external dependencies, both the dependencies and their directions are subject to change with the change in viewing parameters. Note that each internal face which is not orthogonal to the image plane always induces an internal dependency, whereas only external faces may be the source of external dependencies. These two types of dependencies are constructed in two distinct steps during the preprocessing phase of the MC algorithm. In the first step, the connectivity relation among the cells is exploited to find the internal dependencies. We traverse all the cells in CA structure. For each face of each cell, we estimate the z component of the normal vector of the face in NPC. The sign of the z component determines if the face is a front or a back face. If it is 0, then the face is *orthogonal* to the image plane, and it can be ignored because its projection area will not cover any pixels on the image plane. If the face is a front face, then the neighbor cell that shares this face obstructs this cell, so the *InDegree* field of this cell is incremented by one. If the data is known to be a convex set of convex tetrahedron, then the second step need not be performed and the algorithm can proceed directly to the rendering phase, since in such data sets no external dependencies exist. If there is no guarantee that the data is a convex set, then the second step must be performed. The aim of this step is to determine the external dependencies. We can use the sorted ray segments in the *Screen* data structure to determine the existing external dependencies. It is clear that each successive pair of ray segments stored in a ray list of the *Screen* data structure corresponds to an external dependency between the respective pair of cells that generated these two ray segments. Note that ray lists in nearby pixels may induce the same external dependencies. Hence, the total set of distinct external dependencies in the data is equal to the union of these external dependencies induced by the ray segment pairs. However, determining this union will necessitate search operations and maintenance of linked lists for storing these dependencies in cell basis. Instead, we propose a simple yet effective schema for this purpose. It is obvious that if a ray enters the volume at a cell, then it must exit at some other point from the volume through a back-facing external face. So instead of trying to generate a single external dependency between a pair of external cells, we implicitly generate the same external dependency multiple times by increasing the *InDegree* field of the dependent cell for each such successive pair of ray segments relying on the fact that we can adjust the respective *InDegree*

field by decreasing it by 1 for each ray segment when it leaves the volume in the rendering phase. Therefore, each internal dependency contributes by 1 to the *InDegree* field of the dependent cell, whereas each external dependency contributes by an amount equal to the number of pixels shared between the projection areas of the front-facing external faces of the respective external cell pair. The proposed schema is efficiently implemented by traversing each ray list in the *Screen* data structure, and incrementing the *InDegree* field of every cell that generated a ray segment except for the cell that generated the first ray segment in the list. As both types of dependencies have been found, the rendering phase can now be started.

The rendering phase begins by traversing the $\mathcal{CA}$ structure, and inserting the indices of the cells with *InDegree* equal to 0 into the *CellQueue*. This is due to the fact that the cells with *InDegree* equal to 0 are the cells that are not obstructed by any other cell, so they must be processed first. Figure 3.4 shows a sample case from the execution of MC algorithm. After the *CellQueue* is initialized, the algorithm proceeds in the following way. An item is dequeued from the queue which is the index of the cell to be processed. There are 2 tasks to be performed in processing of each cell: removing the dependencies induced by this cell onto the other cells, and scan converting its back faces. The internal dependencies are removed by decreasing the *InDegree* fields of the neighbor cells which are connected to this cell through an internal back face of this cell by 1. The process of removing external dependencies and scan conversion of back faces are performed in an interleaved manner for the sake of efficiency. As a back face is scan converted, the *ZDepth* and *SValue* fields stored in the first ray segment of the ray list in each covered pixel are used as the values at the entry points of the rays into the cell, and the values obtained by scan conversion of the face are used as the values at the exit points of the rays from the cell for resampling and composition operations just like in the previously presented three algorithms. The linear sampling method, which is a very efficient way of resampling, can be used in the MC algorithm. The (R, G, B, ρ) values obtained from the sampling are composited onto the (R, G, B, ρ) values in the *Sample* component of the subject ray segment, and its *ZDepth* and *SValue* fields are updated by the values obtained from the scan conversion of the back face. If the face is an external back face, then this

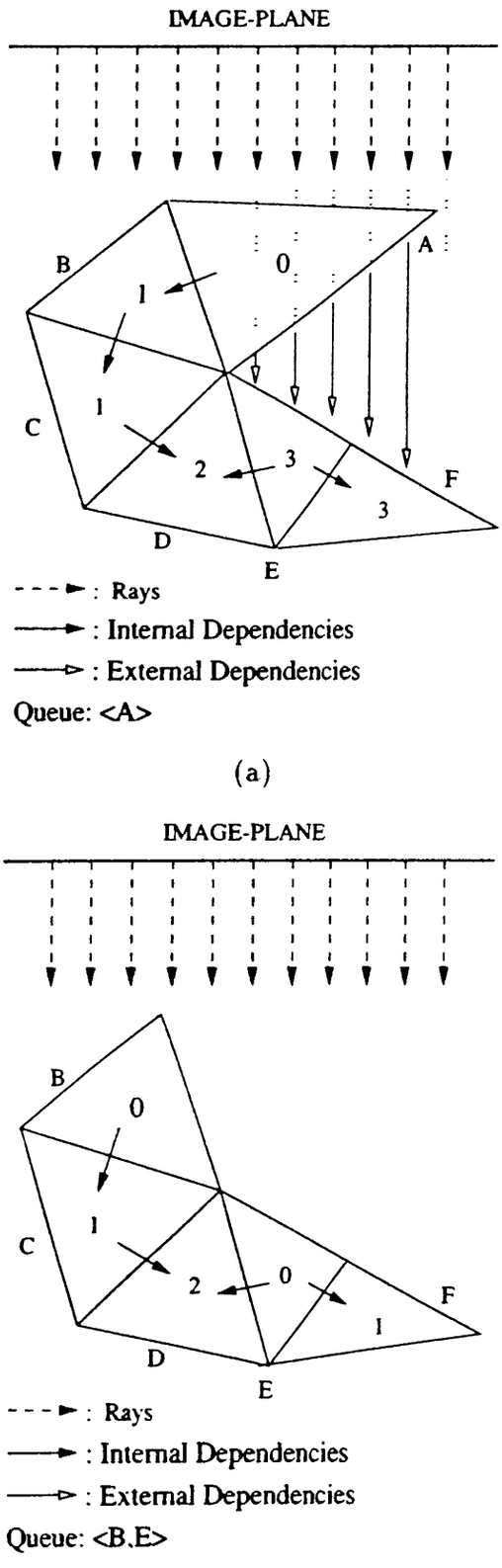


Figure 3.4: A sample case from the rendering phase of MC algorithm. The numbers inside the cells show their current *InDegrees* (a) *Queue* is initialized with cell A. (b) After cell A is processed, its dependents B and E, are placed in the *Queue*.

means that the subject ray segment is leaving the volume, so the *Sample* field holds the final values for this ray segment. So if there is a ray segment in the respective ray list that proceeds the subject ray segment, then we copy the *Sample* field of the subject ray segment to that of the next one, and we delete the subject ray segment. Meanwhile, if such a ray segment exists, then we also decrement the *InDegree* field of the cell that generated that ray segment removing one of the external dependencies induced by the cell being processed. On the other hand, if no such next ray segment exists in the ray list, then the values in the *Sample* component of the subject ray segment represent the final values of the respective pixel in the image being generated. This way, ray lists shrink every time an external back face is scan converted, and the *Sample* component of the first ray segment in each ray list always keep the composited samples by the processed cells so far.

After all the cells are processed, the *CellQueue* will become empty, and the ray lists in the *Screen* structure will either have 1 ray segment, or none at all. The pixels with a ray list that has only 1 ray segment are those which are covered by the projection area of the volumetric dataset, and the pixels with a ray list that has no items at all are those that are outside the coverage area of the projection area of the volumetric dataset. Moreover, the *Sample* components of these ray segments contain the final 3 channel color and opacity values that should be obtained by casting a ray shot from each respective pixel, so they can be directly used to construct the image to be displayed.

The MC algorithm is expected to perform very well in all datasets, especially where the cells are large, and/or when the image is large. The reason behind this is that all projection algorithms use coherence in the data by scan conversion speeding up the rendering process. To be able to do this, they perform some operations which are an overhead like visibility ordering of cells in this algorithm. So if the cells are large, or if the resolution is high which generates large projection areas even if the cells are small, then the effect of this overhead is suppressed by the gain from the exploiting of coherence. Also note that the visibility ordering process is partially independent of the screen resolution, and completely independent of the transfer function in the MC algorithm. This means that the algorithm is very suitable for applications where the transfer

function change is dominant over the change in viewing parameters, because as long as the viewing parameters are kept constant, then only the rendering phase can be executed to produce a correct image at a cost of duplicating the *InDegree* fields. The visibility ordering process is partially independent of the screen resolution, because when the resolution changes only the external dependencies may change. So if one keeps two distinct *InDegree* fields, one for the internal dependencies, and one for the external dependencies produced by external cells, then only the second step of the preprocessing phase must be executed as the internal dependencies do not change by the screen resolution. Another superiority of the MC algorithm over other projection algorithms is that, this algorithm generates the visibility order on the fly during the rendering phase by the help of the information gathered by a preprocessing phase. In general, other approaches execute a topological sort algorithm on the cells after determining the cell dependencies, but this algorithm performs the topological sort on the fly using the *InDegree* fields, and the *CellQueue* structure. Furthermore, the MC algorithm is capable of generating high quality images since it is a ray-casting based DVR algorithm, whereas most of the projection algorithms are only approximations. Beyond these advantages, the major disadvantage of this algorithm is that it has to sort all the cells in the volume, even if the projection area of a cell does not cover a pixel on the screen, thus having no effect in the image produced. Ray-casting algorithms consider only those cells that are intersected by a ray, so such small cells are ignored by the algorithm implicitly, but the MC algorithm can not avoid such cases.

3.6 Possible Extensions

In all algorithms discussed in this thesis, the ray lists in the *Screen* structure may introduce considerable memory overhead in the rendering of datasets with high rate of non-convexity especially to obtain high resolution images. In this section, we propose optimization schemes on the proposed algorithms especially to reduce their space complexities at a cost of slightly decreasing their speed. However, as explained below, the realization of these optimizations may even yield faster renderings. The proposed schemes totally avoids the ray lists in the

Screen data structure in all algorithms. However, this necessitates the addition of a new data structure for the MC algorithm.

3.6.1 Koyamada's, SBRC and H-SBRC Algorithms

We can exploit the face-by-face version of Challinger's rendering algorithm [48, 49, 7] with slight modifications to totally avoid the *Screen* structure as described as follows. In the first phase of the proposed algorithm, all front facing external faces are bucket sorted into a y-bucket according to their minimum y coordinate. The algorithm proceeds from one scanline to other scanline on the image plane, and from one pixel to other pixel in the same scanline. An active list of external faces are created for the current scanline using the y-bucket. The active external faces y-extend of those which cover the current scanline are intersected with the current scanline to determine the edge intersections. The spans created by the edge intersections are sorted into an x-bucket. As pixels are processed in the current scanline, an active span list is created using the x-bucket. The spans are rasterized to generate the intersection points of the ray shot at the current pixel with the front facing external faces. The distance of each ray-face intersection and the information to identify the external face in the $\mathcal{CA}$ structure are inserted into a sorted linked list, called *Intersection list* (referred to here as z-list), which is sorted in increasing distance values. Hence, each item in a z-list corresponds to a ray item that denotes a ray-segment in the respective ray list of our *Screen* structure. Projections of front-facing external faces cover consecutive scanlines on the image plane. Hence, the intersection of scanlines with active external faces can be carried out by incremental calculations. Each span in the current scanline covers consecutive pixel locations. Therefore, sorting of z-intersections with polygons are avoided as long as the list of external faces intersected by the ray does not change.

For each item in the z-list of the current pixel, the respective ray-segment is followed in the volume according to the underlying algorithm (Koyamada's, SBRC, H-SBRC) until it hits an external face which must be a back external face. It is clear that the next external face that the ray will intersect will

be identified by the next item in the z-list. This scheme for generating ray-segments can be utilized for all three algorithms, Koyamada's, SBRC and H-SBRC, without any change. Note that this scheme inherently yields the ray-segment processing order required by the SBRC and H-SBRC algorithms.

Beside reducing the space complexity by a reasonable amount, the proposed scheme is expected to yield run-time performance improvements in the rendering of datasets with high rate of non-convexity. Note that in the current implementation of Koyamada's, SBRC and H-SBRC algorithms, ray-generation phase involves sorting operations in ray-segments for each pixel from scratch. However, the proposed extension considerably reduces the amount of these sorting operations by exploiting intra-scanline coherency on front external faces as described above. On the other hand, it should be noted that the current implementation is expected to run faster than this enhanced version on the experimental datasets used in this work because of their low rate of non-convexity.

3.6.2 MC Algorithm

It is clear that the optimization schema suggested in the previous section can not be used for the MC algorithm, but we can achieve the same goal of avoiding ray lists in the *Screen* structure by slightly modifying the original MC algorithm. As the ray lists in the *Screen* structure are completely avoided, the following fields of the ray-segment data structure must be embedded into each entry of the *Screen* structure: *ZDepth*, *SValue* and *Sample*.

The initial ray generation phase is totally skipped. Instead, an algorithm that works in 3D to sort n arbitrarily placed convex polyhedra is employed to sort front external cells. The details of such an algorithm are described by Stein *et al.* [46]. Stein *et al.* [46] generalizes the *Newell, Newell and Sancha painter's* sort for polygons to 3D volume elements. It is an $O(n^2)$ algorithm which is stated to perform much better in most of the cases [46]. The input to this sorting subroutine will be the cells with front external faces, which are the front external cells indeed, and the output will be stored in a linked list referred to here as *SortedExternalCells (SEC)*. The first step of the preprocessing phase that determines the internal dependencies through computing the *InDegree*

fields of the cells is performed as in the original MC algorithm. The second step of this phase which generates the external dependencies is totally skipped.

The rendering phase is modified as follows. The *CellQueue* is initially empty. We scan the *SEC* list and those cells with an *InDegree* value of 0 are deleted from the *SEC* and they are enqueued to the *CellQueue*. The original MC algorithm works on the *CellQueue* until it becomes empty with the following modifications. For the front external cells, their front faces must also be scan converted and this should precede the scan conversion of their back faces. If the face is an external front face, then the z and *scalar* values obtained are written to the *ZDepth* and *SValue* fields of the corresponding entries of the *Screen* structure. During the scan conversion of a back face, the z and *scalar* values at the entry points of the rays are taken from the *ZDepth* and *SValue* fields of the corresponding entries of the *Screen* structure, and the resampling is performed in the same way, but the results are composited to the *Sample* field of the corresponding entry of the *Screen* structure. The internal dependencies are manipulated just like in the original algorithm, but no work is done for the manipulation of external dependencies. That is, during the scan conversion of a back external face, the decrement operations for the *InDegree* fields are skipped. The *InDegree* field of each processed cell is set to -1 to indicate that it is processed.

When the *CellQueue* becomes empty, we make a pass on the *SEC* list to select the cells with an *InDegree* value of 0. These cells are deleted and enqueued to the *CellQueue*. Meanwhile, during this pass any cell which has an *InDegree* of -1 is also deleted from the *SEC*. This process is repeated until the *SEC* becomes empty.

We have preferred the version in the original algorithm, because for types of non-convex grids whose rate of non-convexity is low, that version performs faster and most of the datasets are of this type. So as the main objective of this thesis is to develop fast algorithms, we preferred the fast version at a cost of higher space complexity. On the other hand, the enhanced version can be expected to be faster in cases where the projection areas of front external faces are large.

Chapter 4

Experimental Results

In this chapter, the experimental data used in this work and results obtained from the executions of our implementations will be presented.

4.1 Datasets and Environment

In this thesis, four datasets obtained from NASA-Ames Research Center are used. All datasets were originally *curvilinear* in structure, and they represent the results of *CFD* simulations. They were first obtained in Nasa Plot3D format. The raw datasets consisted of hexhedral cells, so we converted them into unstructured standard tetrahedral format by breaking each hexahedral cell into 5 tetrahedal cells. During this operation, we also constructed the *connectivity* relation as this has to be given explicitly for unstructured grids.

The first dataset is named as *Blunt Fin (BF)* dataset. It defines the airflow over a flat plate with a blunt fin rising from the plate. Free stream flow is parallel to the plate and to the flat part of the fin, entirely in the x component direction. As the flow is assumed to be symmetrical about a plane through the center of the fin, only one half of the real geometry is present in the dataset. In curvilinear format, the original dataset had $40 \times 32 \times 32$ grid points.

The second dataset is named as *Combustion Chamber (CC)*. It models the

Name	Dimensions	N	C
Blunt Fin (BF)	$40 \times 32 \times 32$	40,960	187,395
Combustion Chamber (CC)	$57 \times 33 \times 25$	47,025	215,040
Oxygen Post (OP)	$38 \times 76 \times 38$	109,744	513.375
Delta Wing (DW)	$56 \times 54 \times 70$	211,680	1,005.675

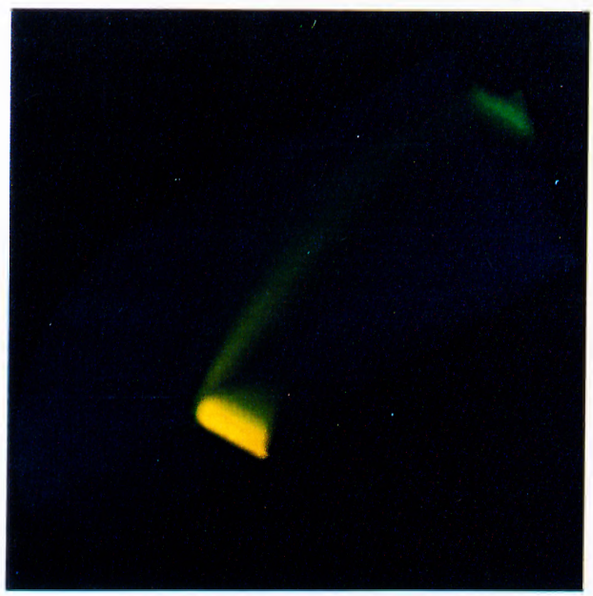
Table 4.1: List of datasets used for testing. *Dimensions* are the original NASA Plot3D sizes. N and C represent the actual sizes used by the algorithms.

simulation of flow of various gases obtained from combustion inside a chamber. The original dimensions is $57 \times 33 \times 25$ grid points.

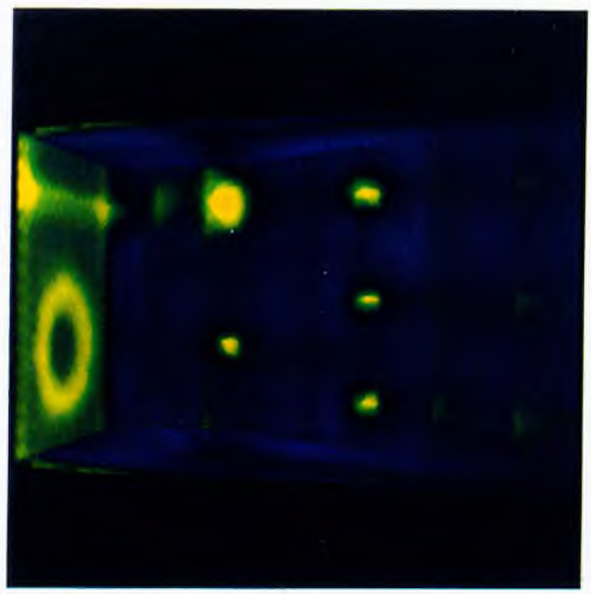
The third dataset is named as *Oxygen Post (OP)*. It represents the liquid oxygen flow across a flat plate with a cylindyrical post rising perpendicular to the plate and therefore, the flow. The simulation models a flow internal to a rocket engine. The original dimensions of this dataset is $38 \times 76 \times 38$ grid points.

The fourth dataset used in this work is named as *Delta Wing (DW)*. It defines the flow past over a very simplified geometry representing a delta wing aircraft, a moderately high angle of attack. The grid being a particularly twisted and scaled one is a good dataset to test of certain features and capabilities of visualization systems. The original dataset contained $56 \times 54 \times 70$ grid points. Table (4.1) summarizes the datasets used in this work. Note that data sets are ordered in increasing order of size, both in terms of number of cells and nodes. This order is maintained in all the following tables for a better and simpler understanding of performance analysis. Figure 4.1 shows the rendered images of the data sets for different views.

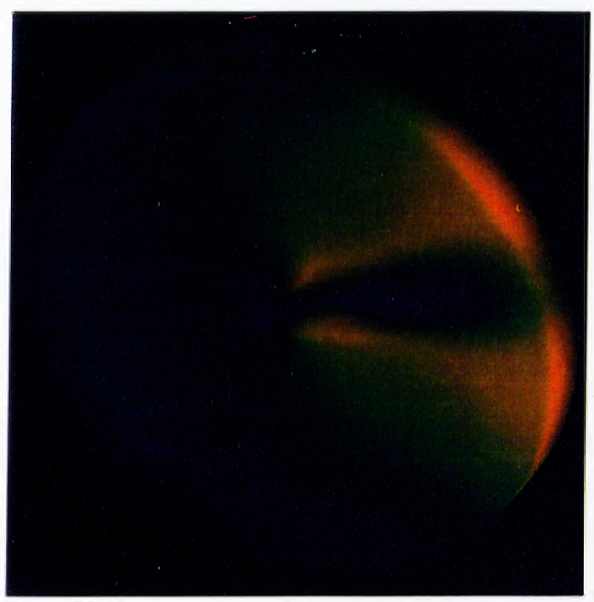
All algorithms presented in this thesis are implemented using the *C language*, and they are compiled with the native *SunSoft C compiler* using *O3* optimization level. Implementations done for benchmarking results are made on an *Sun Ultra Enterprise 4000* computer equipped with 512 Mbytes of memory and 8 Ultra Sparc II (250 Mhz) processors each with a 256 Kbyte level 2 cache. The architecture is a *shared memory* parallel architecture, but we have run our programs on a single processor in sequential mode. Therefore, all times



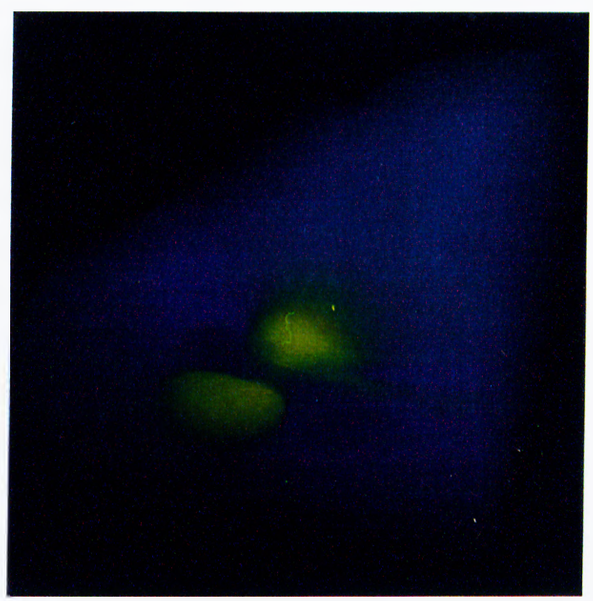
(a) Blunt Fin



(b) Combustion Chamber



(c) Oxygen Post



(d) Delta Wing

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

presented in this work are user times.

The presented algorithms have been experimented using different viewing parameters, therefore the transformation model is briefly explained. Our transformation model is quite simple. The coordinates of the volume read from the file is assumed to be in *Modeling Coordinate System (MCS)*. We find the extents of the volume in MCS and create a translation transformation matrix that will transform the volume to *World Coordinate System (WCS)* such that the center of the volume will be at the origin of WCS. Then, we supply a 3-tuple in the form of (θ, ϕ, ω) in which θ denotes the angle of rotation around x-axis, ϕ denotes the angle of rotation around y-axis and ω denotes the angle of rotation around z-axis. Using this 3-tuple a rotation transformation matrix is generated and this matrix is used to rotate the volume in WCS to the desired orientation. The rotation matrix is constructed such that volume is firstly rotated around x-axis, then around y-axis and finally, around z-axis. The view direction vector $\vec{v}$ and view up vector $\vec{u}$ are kept constant for all visualizations and are chosen to be $(0, 0, -1)$ and $(0, 1, 0)$, respectively. The view point v_p is selected such that it is on positive z-axis, and is outside the volume for each data set. As parallel projection is used, any such point will produce the same image. The view direction vector $\vec{v}$, view up vector $\vec{u}$ and view point v_p defines our *View Reference Coordinate System (VRC)*. The view plane is selected considering the constraints for the selection of the view point. The window on the 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. Finally, we have chosen *Normalized Projection Coordinate System (NPC)* to be the same as VRC again for the sake of simplicity and especially the efficiency of linear sampling method used in the algorithms. So throughout the rest of the thesis different views are are identified by the 3-tuple (θ, ϕ, ω) defining the angles of rotation around x, y and z axis, respectively.

4.2 Memory Requirements

Table 4.2 illustrates the memory requirements of the presented algorithms for each data set and three image resolutions from the standard view which can

Data set	Image res.	Core	Screen	Memory Overhead				
						H-SBRC		
				Koy.	SBRC	EA	BBA	MC
Blunt Fin	300×300	7.2	1.0	0.0	1.5	1.2	1.2	0.7
	600×600		4.1	0.0	1.8	1.6	1.6	1.5
	900×900		9.2	0.0	2.1	1.9	1.8	2.9
Comb. Cham.	300×300	8.2	1.3	0.0	1.6	1.5	1.3	0.8
	600×600		5.1	0.0	1.7	1.7	1.7	1.9
	900×900		11.4	0.0	1.9	1.9	1.9	3.8
Oxygen Post	300×300	19.6	1.0	0.0	3.1	2.6	2.4	1.3
	600×600		4.0	0.0	3.5	2.9	2.8	2.1
	900×900		9.0	0.0	3.8	3.3	3.1	3.4
Delta Wing	300×300	38.3	1.3	0.0	6.3	4.8	4.7	2.4
	600×600		5.3	0.0	7.0	5.6	5.6	3.6
	900×900		11.8	0.0	7.6	6.4	6.3	5.6

Table 4.2: Memory consumptions of the algorithms in *MBytes*.

be defined by the 3-tuple $(0,0,0)$. In the table, *Core* column denotes the memory occupied by the data set itself. *Screen* column represents the memory usage for the *Screen* data structure containing ray lists. The following five columns illustrates the additional memory overhead introduced by the respective algorithms. As seen in the table, Koyamada's algorithm introduces no overheads to the existing data structures. The memory overheads introduced by SBRC and H-SBRC algorithms are due to the data structures necessary for keeping the information about the active edges, active cells, and the span-buffers created. The sources of the memory overhead in the MC algorithm are *InDegree* field added for each cell, *Sample* component added to each ray-segment, and the *CellQueue* structure. The major overhead for the MC algorithm comes from the *Sample* component, and this fact can be clearly observed from the higher rate of increase in the memory overhead with increasing resolution relative to the other algorithms.

As seen in Table 4.2, the percent memory overhead of SBRC algorithm with respect to the core data size is always below 30%, and it decreases with the increasing data size for a fixed image resolution. As expected, H-SBRC algorithm behaves even slightly better than SBRC algorithm due to less number of active edges, active cells and span-buffers. As seen in the table, MC algorithm introduces considerable amount of memory overheads of 40% and 46%

for smaller data sets BF and CC, respectively, at highest resolution (900×900). However, percent memory overheads drastically reduces below 20% for larger data sets OP and DW.

Table 4.2 shows that MC algorithm tends to perform better than SBRC algorithm, interms of memory consumption, for the larger data sets. However, SBRC algorithm is expected to behave much better in visualization applications involving early ray termination since this nice feature is not supported by MC algorithm. That is, in such applications the memory overhead of SBRC algorithm is expected to decrease substantially because of less number of active edges, active cells and span-buffers created. In fact, we have experimentally verified this expectation by changing the transfer function and opacity threshold.

The *Screen* structure occupies more space than the core data for the smaller data sets at highest resolution, but for the larger data sets the percentage of the *Screen* structure to the core data reduces below 50%. However, the schemes proposed in Section 3.6 can be used to totally avoid this structure thereby reducing the overall memory requirements considerably.

4.3 Performance Analysis and Comparisons

The relative performances of the presented algorithms are evaluated by the visualization of each data set using four different views for three distinct image resolutions. As mentioned earlier, different views will be identified by a 3-tuple (θ, ϕ, ω) which denotes the angle of rotation in each of the axis. Four different views are used in our experiments. The 3-tuples $(0,0,0)$, $(0,90,0)$, $(90,0,0)$ and $(45,45,45)$ identify the four views V_1, V_2, V_3 and V_4, respectively. Note that, V_1 corresponds to the viewing parameters for the standard view.

Tables 4.3–4.6 display the dissection of execution times of the algorithms for different view. The visualization statistics for each view are summarized at the bottom sections of these tables. As mid-point sampling shema is used the number of intersections and number of samplings are exactly equal. In

these tables, T_{Tot} denotes the total execution time, T_R denotes the time for generating ray-segments, T_S shows the time spent for sampling and composition operations, and T_I denotes the time spent for calculating the ray-face intersections. In SBRC, T_I is further dissected into three components as $T_{I'}$, T_{CA} and T_{SC}. Here, $T_{I'}$ denotes the time spent for incremental ray-face intersections using the span-buffers, T_{CA} and T_{SC} shows the time spent for cell activation and span creation, respectively. T_{CA} and T_{SC} should be considered as overheads for the sake of determining ray-face intersections more efficiently. In MC algorithm, T_{Pre} denotes the time spent for the preprocessing phase of the algorithm which constructs the internal and external dependencies.

T_R and T_{Pre} have been accurately measured as they can be considered as distinct phases. However, T_S and T_I can not be measured directly because of their highly interleaved manner of execution. To determine T_S and T_I, the programs have been executed twice, one with sampling and the other without sampling. The difference between these two runs yields T_S, thus, enabling the determination of T_I. In SBRC, T_I computed in this manner consists of three components namely, $T_{I'}$, T_{CA} and T_{SC}. However, this dissection can not be computed directly as in the case of intersection and sampling time. Instead, we have estimated the unit costs for each of these operations statistically using the *Least-Squares Line-Fitting* method, and used these costs to determine them approximately. Our experimental analysis show that the maximum error in estimating T_I using these costs do not exceed 6%. Note that these unit costs are actually the coefficients κ, λ, and μ used in Eq. 11 for approximating the execution cost of SBRC algorithm for a cell. The same method has been employed to determine the coefficient β for Koyamada's algorithm in Eq. 10. The same dissection for T_I is not given for H-SBRC because of both increased number of parameters and the extra cost introduced by the bias computations.

As seen in Tables 4.3–4.6, T_R and T_S components are almost the same in all algorithms for all visualization instances, thereby verifying the accuracy of our method for dissectioning into T_R, T_S and T_I (and T_{Pre} in MC). In these tables, it is observed that T_R does not vary linearly, because the cost per pixel for ray-segment generation can not be determined correctly due to the sorting operations involved. Although T_R increases with the image resolution, it always

remains negligible compared to total rendering time. As T_{Pre} in MC can be considered to be negligible because it is very small compared to T_{Tot}, and T_R and T_S are almost equal in all algorithms for all visualization instances, T_I turns out to be the determining factor in relative comparison of the algorithms.

For Koyamada's algorithm T_I increases almost linearly with I (number of intersections) as expected. This tendency can be explained by the fact that it does not use image-space coherence effectively. Recall that number of intersection tests performed by the algorithm is approximately twice as large as I on the average.

In SBRC, cell activation time, T_{CA}, is directly proportional to number of cells touched during rendering, instead of total number of cells in the data. Therefore, in Tables 4.3 - 4.6, T_{CA} gently increases with increasing resolution for data sets BF, OP and DW. However, it remains constant for CC, and furthermore, it is interesting to note that T_{CA} spent for CC compared to OP in Tables 4.5 and 4.4 for 300×300 resolution, although OP is considerably larger than CC in terms of number of cells. This behaviour can be attributed to the fact that all cells of CC are sufficiently large so that all of them are hit by a ray even in the smallest resolution. In the tables, it is also observed that the rate of increase in $T_{I'}$ with increasing resolution for a fixed data set is much more pronounced than that of T_{SC}. This is due to the fact that number of spans created is related to the height dimension of the resolution, whereas number of intersections is associated to both width and height dimensions. These tables also show that T_{CA} becomes negligible with increasing resolution, and $T_{I'}$ and T_{SC} become the dominating components in T_I.

H-SBRC shows the same characteristics as SBRC for different data sets and resolutions, but it is interesting to note that, in general, H-SBRC performs better than SBRC in small resolutions, and when the resolution increases it becomes slightly worse. This can be mainly explained by the fact that when the resolution gets larger, the projection areas of cells get larger resulting most of the cells to be processed by SBRC, and therefore, it performs worse than SBRC due to the overhead introduced by the bias computations for selecting the algorithm. By nature, H-SBRC is expected to give the best payoff in situations where the variation in cell size is large. The instances of DW and

OP for 300×300 resolution verifies this expectation by always reducing total rendering time of SBRC.

Views V_2 and V_3 result in cyclic meshes for DW due to the internal dependencies and as MC can not handle cyclic meshes, the execution times for these instances are not available (N/A) in Tables 4.4 and 4.5. The dissection tables illustrates that T_{Pre} is directly affected by two factors, namely, number of faces (number of cells) and number of ray-segments generated. Former factor is independent of both viewing parameters and resolution, whereas the latter one depends on both, therefore the time spent for generating internal dependencies is constant for a fixed data set. It is also seen that T_{Pre} can always be considered to be negligible for almost all instances, especially for higher resolutions of MC.

Table 4.7 illustrates the relative run-time performances of the presented algorithms for standard view V_1. In the table, all the proposed algorithms are observed to perform better than Koyamada's algorithm in all visualization instances except the visualization of DW in 300×300 resolution by MC. The poor performance of MC in this specific instance is because of the large gap between the number of cells processed by MC and the other algorithms. The performance gap between proposed algorithms and Koyamada's algorithm increases in favour of the proposed ones with increasing resolution, as expected. It is seen that, the proposed algorithms are 2 to 3 times faster than Koyamada's algorithm, in general. The H-SBRC approach is observed not to increase the performance with respect to SBRC algorithm, and it is seen that MC achieves the best performance, in general, for the standard view.

Table 4.8 shows the average relative run-time performances for four views. Each value in Table 4.8 is computed by averaging the normalized execution times of the visualizations of the same data set with a fixed resolution from four different views. Table 4.8 shows that H-SBRC performs considerably better than SBRC in both 300×300 and 600×600 resolutions of OP and DW. This experimental finding is because of the large variation in cell sizes in these data sets. On the contrary, H-SBRC performs worse than SBRC for CC which has almost equally sized large cells. It is clearly seen from the table that BBA heuristic always performs better than EA heuristic in low resolutions.

and this performance difference decreases with increasing resolution so that EA begins to perform better than BBA in high resolutions. This is mainly due to the fact that the favour of BBA heuristic towards SBRC in overestimating the cell projection areas is not sufficient enough to misfavour SBRC in low resolutions. In other words, for low resolutions the errors introduced by BBA does not exaggerate the cell projection areas enough to cause bias misselect SBRC approach instead of Koyamada's approach. As resolution is increased, these errors cause bias to select SBRC for cells which should be processed by Koyamada's approach indeed, thus resulting in worse performance.

Average values in Tables 4.8 and 4.7 are graphically displayed in Fig. 4.3 as the variation of relative performances of the proposed algorithms with respect to Koyamada's algorithm with increasing resolution for each data set. Relative performances of all proposed algorithms with respect to Koyamada's algorithm increases with increasing resolution for the same data set. This rate of performance increase is much more pronounced in the MC algorithm in all data sets except CC. In fact, a careful analysis of Fig. 4.3(b) reveals that SBRC and H-SBRC achieve slightly larger rate of performance increase with respect to MC, because the number of cells processed does not increase with increasing resolution thereby stressing that the benefit of using image-space coherency is more in high resolutions.

As seen in Table 4.8 and Fig. 4.3, H-SBRC$_{BBA}$ achieves the best run-time performance in 300×300 resolution in all data sets except CC in which MC achieves the best. In 600×600 resolution, MC achieves the best performance for BF and CC, but H-SBRC$_{BBA}$ performs still better than MC in OP and DW by taking the advantage of large variation in cell projection areas. As expected in 900×900 resolution, MC outperforms SBRC and H-SBRC for BF and CC, however it can not beat H-SBRC$_{EA}$ in OP, and in DW, it still remains below both SBRC and H-SBRC$_{EA}$. When we consider the averages over data sets for different resolutions given at the bottom of Table 4.8, we see that H-SBRC$_{BBA}$ achieves the best performance both in 300×300 and 600×600 resolutions, and all the proposed algorithms perform nearly the same in 900×900 resolution. Finally, on the overall average, H-SBRC turns out to yield the best performance among the proposed algorithms.

Table 4.9 illustrates the comparison of execution times of the presented algorithms with LSRC algorithm [5], since it is one of the latest works in the literature and it is reported to be two orders of magnitude faster than other algorithms implemented in software. As mentioned earlier, all algorithms listed in the table are capable of producing images of the same high quality. Koyamada's and the proposed SBRC and H-SBRC algorithms are superior to LSRC algorithm in terms of algorithmic quality since they can handle cyclic meshes at no cost and support early ray termination, whereas the LSRC algorithm can not support early ray termination. However, MC can be considered to be inferior to LSRC in terms of algorithmic quality since MC can not handle cyclic meshes. As seen in the table, all of the proposed algorithms are considerably faster than LSRC algorithm on all visualization instances despite the fact that our benchmark machine *(Sun Ultra Enterprise 4000)* is estimated to be approximately 1.18 times slower than their benchmark machine *(SGI Power Challenge)*. This ratio is computed using the *specfp95* figures[1] of these two machines. Considering the run-time estimates of LSRC algorithm on our benchmark machine, it can easily be seen that MC algorithm is 1.8 to 2.4, SBRC and H-SBRC algorithms are 1.5 to 3.2 times faster than LSRC algorithm. H-SBRC$_{BBA}$ achieves the highest relative performance ratios of 3.2 and 2.4 with respect to LSRC in larger data sets OP and DW at low resolution of 300×300. Although the proposed SBRC and H-SBRC algorithms are substantially faster than LSRC algorithm, restricted reported results of LSRC algorithm on larger resolutions leads us to the idea that LSRC can scale slightly better than SBRC and H-SBRC with increasing resolution. In OP, relative performance ratio of SBRC and H-SBRC algorithms with respect to LSRC algorithm reduces from 2.4 at 300×300 to 1.8 at 600×600 and finally, to 1.5 at 900×900. However, the similarity of LSRC and MC algorithms, in the sense that they both have to process all the object-space, encourages us that SBRC and especially H-SBRC will achieve a better performance in a wider analysis for different data sets at different resolutions from different views, and even for varying transfer functions. Finally, Table 4.9 illustrates that SBRC and H-SBRC algorithms scale better than LSRC with increasing data set size for fixed resolution. For

[1] http://www.specbench.org/osg/cpu95/results/cfp95.html

example, relative performance ratio of H-SBRC$_{BBA}$ with respect to LSRC increases from 1.4 to 2.4 and finally to 3.2 in the rendering of CC, OP and DW in increasing order of data set size at similar resolutions of 300×300, respectively.

Alg. Name	T_{exec} (sec.)	Image Resolution											
		300×300				600×600				900×900			
		Data set											
		BF	CC	OP	DW	BF	CC	OP	DW	BF	CC	OP	DW
Koy.	T_R	0.2	0.4	0.5	0.9	0.4	0.8	1.0	1.4	0.7	1.4	1.6	2.0
	T_S	1.2	1.7	4.0	3.3	4.9	7.1	15.1	13.4	10.6	15.6	32.9	29.9
	T_I	9.1	14.0	26.6	25.4	36.6	55.6	105.9	100.7	82.8	125.1	238.4	225.5
	T_{Tot}	10.6	16.2	31.3	30.0	42.0	63.6	122.2	115.9	94.1	142.2	273.1	257.8
SBRC	T_R	0.2	0.5	0.6	1.0	0.5	1.0	1.2	1.6	1.0	1.9	2.3	2.7
	T_S	1.3	1.7	3.5	3.6	5.0	6.9	14.2	14.3	11.3	15.6	32.0	32.1
	$T_{I'}$	1.5	2.2	4.5	4.5	5.3	9.5	18.0	18.1	14.3	21.5	40.6	40.7
	T_{CA}	1.6	3.5	3.6	8.1	2.2	3.5	4.7	11.9	2.4	3.5	5.3	13.5
	T_{SC}	2.0	6.1	6.1	10.3	5.3	14.2	14.6	26.7	7.0	21.5	24.4	46.0
	T_{Tot}	6.6	14.1	18.6	27.9	18.4	35.4	52.9	73.0	36.2	64.2	104.9	135.5
Hyb_{EA}	T_R	0.2	0.4	0.5	1.0	0.4	0.8	1.0	1.4	0.7	1.4	1.7	2.1
	T_S	1.4	2.0	3.7	4.5	5.6	7.8	14.9	14.3	11.9	16.1	34.3	32.1
	T_I	5.0	12.8	13.9	19.5	14.2	29.3	38.7	55.4	26.0	50.5	73.8	104.2
	T_{Tot}	6.6	15.3	18.3	25.4	20.3	38.0	54.8	71.6	38.7	68.1	109.9	138.8
Hyb_{BBA}	T_R	0.2	0.4	0.5	0.9	0.4	0.8	1.0	1.4	0.7	1.4	1.7	2.4
	T_S	1.4	1.9	3.6	3.6	5.4	8.0	15.1	14.5	11.1	15.6	33.3	28.7
	T_I	5.0	13.4	13.6	18.8	14.1	28.7	40.0	54.2	26.9	50.4	78.7	113.6
	T_{Tot}	6.6	15.7	17.9	23.8	20.0	37.6	56.3	70.4	38.8	67.5	113.9	145.0
MC	T_{Pre}	0.6	0.9	1.7	3.1	0.9	1.4	2.2	3.6	1.3	2.1	3.1	4.5
	T_R	0.2	0.4	0.5	0.9	0.4	0.8	0.9	1.4	0.7	1.4	1.6	2.0
	T_S	1.3	1.8	3.5	3.3	5.1	6.7	14.0	13.1	11.6	14.6	32.0	30.8
	T_I	4.6	8.7	13.4	25.5	10.5	18.9	30.3	46.5	19.4	34.1	55.7	75.6
	T_{Tot}	6.6	11.4	18.9	32.3	16.6	27.1	46.8	63.6	32.3	50.9	91.0	111.2
Visualization Statistics													
P: #pixels×10^3		28.4	54.7	67.4	54.9	114.1	219.2	270.4	220.1	257.1	493.8	609.0	495.7
R: #ray seg.×10^3		28.4	54.7	67.4	60.5	114.1	219.2	270.4	243.0	257.1	493.8	609.0	547.2
I: #int.×10^3		2645	3655	7481	7508	10615	14671	30010	30146	23907	33047	67599	67880

Table 4.3: Execution-time dissection and visualization statistics of algorithms for view V_1.

Alg. Name	T_{exec} (sec.)	Image Resolution											
		300×300				600×600				900×900			
		Data set											
		BF	CC	OP	DW	BF	CC	OP	DW	BF	CC	OP	DW
Koy.	T_R	0.3	0.4	0.5	0.9	0.7	0.9	0.7	1.2	1.3	1.7	0.9	1.8
	T_S	2.6	3.1	0.6	1.4	10.3	10.5	2.2	5.9	23.2	26.4	4.8	13.4
	T_I	18.5	20.4	4.3	11.5	74.1	82.5	17.0	45.2	166.1	182.6	38.1	101.0
	T_{Tot}	21.4	24.0	5.6	14.2	85.2	94.0	20.1	52.7	190.8	210.7	44.0	116.6
SBRC	T_R	0.3	0.4	0.5	0.9	0.8	0.9	0.6	1.2	1.6	1.7	0.9	1.8
	T_S	2.6	2.9	0.6	1.6	10.6	11.7	2.3	6.3	23.9	26.4	5.1	14.2
	$T_{I'}$	3.1	3.7	0.7	2.0	11.2	14.8	2.9	8.0	30.3	33.4	6.4	18.0
	T_{CA}	2.0	3.5	3.0	4.3	2.5	3.5	4.9	6.1	2.7	3.5	6.0	7.2
	T_{SC}	3.9	8.0	3.7	7.3	10.4	18.6	10.4	18.1	14.7	29.0	17.9	29.4
	T_{Tot}	12.1	18.7	8.7	16.5	35.5	49.7	21.2	40.1	73.3	94.2	36.4	70.9
Hyb_{EA}	T_R	0.3	0.4	0.5	0.9	0.7	0.9	0.7	1.2	1.4	1.7	0.9	1.8
	T_S	3.0	3.4	0.6	1.7	11.2	12.2	2.4	6.6	23.4	27.4	5.5	15.0
	T_I	8.7	16.4	5.1	10.9	26.9	41.0	14.6	30.4	50.9	72.8	27.2	54.9
	T_{Tot}	12.2	20.3	6.4	13.9	38.9	54.2	18.0	38.6	75.8	101.9	33.8	72.1
Hyb_{BBA}	T_R	0.3	0.4	0.5	0.9	0.7	0.9	0.7	1.2	1.4	1.7	0.9	2.0
	T_S	2.6	3.0	0.6	1.5	10.3	13.3	2.2	5.9	23.5	26.6	5.3	13.4
	T_I	8.8	16.3	4.8	10.3	26.1	39.9	14.2	29.3	51.5	72.4	27.7	61.2
	T_{Tot}	11.8	19.8	6.0	13.1	37.3	54.2	17.3	36.9	76.5	100.8	34.0	77.1
MC	T_{Pre}	0.7	0.9	1.7	N/A	1.2	1.5	1.9	N/A	2.0	2.4	2.2	N/A
	T_R	0.3	0.4	0.5	N/A	0.7	0.9	0.7	N/A	1.3	1.6	0.9	N/A
	T_S	2.6	2.9	0.6	N/A	10.5	11.5	2.0	N/A	23.8	26.3	4.6	N/A
	T_I	9.3	12.0	12.6	N/A	23.4	30.6	19.5	N/A	44.1	57.2	27.7	N/A
	T_{Tot}	12.7	15.9	15.0	N/A	35.1	43.8	23.7	N/A	70.0	86.0	34.7	N/A
Visualization Statistics													
P:#pixels×10^3		57.7	60.9	13.7	46.5	231.5	244.4	55.2	186.7	521.5	550.5	124.2	420.7
R:#ray seg.×10^3		57.7	66.4	14.2	46.7	231.5	266.2	57.0	187.3	521.5	599.7	128.3	422.0
I:#int.×10^3		5584	6165	1187	3312	22408	24746	4766	13294	50476	55741	10734	29944

Table 4.4: Execution-time dissection and visualization statistics of algorithms for view V_2.

Alg. Name	T_{exec} (sec.)	Image Resolution											
		300×300				600×600				900×900			
		Data set											
		BF	CC	OP	DW	BF	CC	OP	DW	BF	CC	OP	DW
Koy.	T_R	0.2	0.3	0.5	0.9	0.4	0.7	0.7	1.3	0.6	1.2	0.9	1.9
	T_S	0.8	1.7	0.6	1.4	3.3	6.9	2.2	5.7	7.2	15.4	5.1	12.8
	T_I	5.1	11.5	4.2	10.4	20.2	46.0	16.6	40.8	45.2	101.4	37.5	91.3
	T_{Tot}	6.1	13.6	5.5	13.1	23.9	53.6	19.8	48.2	53.1	118.0	43.7	106.5
SBRC	T_R	0.2	0.3	0.5	0.8	0.3	0.6	0.6	1.2	0.6	1.1	0.9	1.9
	T_S	0.8	1.7	0.6	1.5	3.2	6.6	2.3	6.1	7.2	15.0	5.1	13.7
	$T_{I'}$	0.9	2.1	0.7	1.9	3.4	8.4	2.9	7.7	8.4	19.0	6.4	17.4
	T_{CA}	1.5	3.5	3.0	5.5	2.1	3.5	4.9	7.6	2.4	3.5	6.0	8.9
	T_{SC}	1.3	6.7	1.3	3.9	3.5	15.6	4.4	10.2	6.0	24.9	8.1	17.4
	T_{Tot}	4.8	14.4	6.3	14.1	12.6	34.9	15.2	33.3	24.7	63.6	26.7	59.6
Hyb_{EA}	T_R	0.2	0.3	0.5	0.9	0.4	0.7	0.7	1.3	0.6	1.2	0.9	2.0
	T_S	0.9	1.9	0.6	1.6	3.6	7.1	2.4	6.1	7.4	15.8	5.3	13.6
	T_I	3.1	13.0	3.5	8.9	9.4	30.2	9.9	24.5	17.3	51.9	18.4	45.0
	T_{Tot}	4.3	15.4	4.8	11.8	13.4	38.1	13.2	32.4	25.4	68.9	24.8	61.0
Hyb_{BBA}	T_R	0.2	0.3	0.5	0.9	0.4	0.7	0.7	1.3	0.6	1.2	0.9	2.2
	T_S	0.8	1.7	0.6	1.5	3.2	7.6	2.2	5.8	7.3	15.4	5.3	12.9
	T_I	3.0	11.9	3.4	8.5	9.0	29.3	9.5	23.9	17.3	51.3	18.9	50.0
	T_{Tot}	4.1	14.1	4.6	11.3	12.6	37.6	12.6	31.3	25.3	68.0	25.3	65.6
MC	T_{Pre}	0.6	0.8	1.7	N/A	0.8	1.3	1.9	N/A	1.2	1.9	2.3	N/A
	T_R	0.2	0.3	0.5	N/A	0.3	0.7	0.7	N/A	0.6	1.1	0.9	N/A
	T_S	0.8	1.8	0.6	N/A	3.3	6.8	2.4	N/A	7.4	15.3	5.4	N/A
	T_I	4.8	9.5	10.6	N/A	9.5	22.2	15.2	N/A	16.2	39.5	21.5	N/A
	T_{Tot}	6.3	12.2	13.1	N/A	13.7	30.3	19.7	N/A	24.8	56.9	29.4	N/A
Visualization Statistics													
P:#pixels×10^3		22.0	37.0	13.7	56.9	88.3	148.5	55.2	228.3	198.8	334.6	124.2	514.2
R:#ray seg.×10^3		22.0	38.8	20.8	56.9	88.3	155.6	83.7	228.3	198.8	350.6	188.4	514.2
I:#int.×10^3		1687	3499	1188	3204	6771	14044	4768	12861	15251	31633	10740	28970

Table 4.5: Execution-time dissection and visualization statistics of algorithms for view V_3.

Alg. Name	T_{exec} (sec.)	Image Resolution: 300×300 Data set BF	CC	OP	DW	600×600 BF	CC	OP	DW	900×900 BF	CC	OP	DW
Koy.	T_R	0.3	0.4	0.6	0.9	0.7	0.8	1.0	1.3	1.3	1.5	1.5	2.1
	T_S	1.8	1.6	2.0	2.3	7.5	6.5	8.1	9.2	16.4	14.5	18.2	20.8
	T_I	13.9	12.1	15.6	17.6	55.6	48.6	61.9	69.1	125.2	107.3	138.7	154.6
	T_{Tot}	16.1	14.2	18.3	21.2	63.9	56.1	71.1	80.1	142.9	123.5	158.6	177.8
SBRC	T_R	0.3	0.4	0.6	0.9	0.7	0.8	0.9	1.3	1.2	1.5	1.5	1.9
	T_S	1.9	1.6	1.9	2.5	7.7	6.5	7.7	9.9	16.8	14.6	17.3	22.3
	$T_{I'}$	2.2	2.0	2.4	3.1	11.0	8.9	11.4	14.6	23.1	20.1	25.6	30.6
	T_{CA}	2.3	3.5	4.2	7.2	2.9	3.5	7.3	11.1	3.7	3.5	8.8	15.4
	T_{SC}	5.1	6.7	5.4	7.8	12.0	15.1	11.7	19.2	19.5	22.9	19.6	33.4
	T_{Tot}	11.8	14.4	14.8	21.8	34.5	34.9	39.2	56.5	64.3	62.6	73.0	104.1
Hyb_{EA}	T_R	0.3	0.4	0.6	0.9	0.7	0.9	1.0	1.3	1.3	1.5	1.5	2.0
	T_S	2.1	2.2	2.1	2.5	8.2	6.9	8.4	9.8	14.8	15.2	17.9	21.8
	T_I	8.9	13.0	10.1	13.8	25.9	29.4	28.0	39.5	50.7	49.7	51.9	76.4
	T_{Tot}	11.4	15.6	13.0	17.6	35.0	37.3	37.6	51.1	66.9	66.6	71.6	100.6
Hyb_{BBA}	T_R	0.3	0.4	0.6	0.9	0.7	0.8	1.0	1.3	1.4	1.5	1.6	2.3
	T_S	1.9	1.7	2.0	2.4	7.5	7.8	7.8	9.3	17.2	14.6	18.7	21.0
	T_I	9.1	12.3	9.7	14.7	26.0	28.5	27.1	42.1	49.8	49.3	54.0	87.9
	T_{Tot}	11.4	14.5	12.5	18.4	34.3	37.1	36.2	53.2	68.5	65.5	74.5	111.6
MC	T_{Pre}	0.8	0.9	1.8	3.1	1.2	1.4	2.2	3.6	1.9	2.3	3.0	4.5
	T_R	0.3	0.4	0.6	0.9	0.7	0.8	1.0	1.3	1.2	1.5	1.5	2.0
	T_S	1.8	1.9	1.9	2.7	7.3	7.5	7.8	10.6	16.7	16.5	17.6	23.5
	T_I	8.6	9.5	14.3	25.3	20.9	22.3	27.1	44.0	38.1	40.3	45.1	70.6
	T_{Tot}	11.3	12.4	18.2	31.5	29.5	31.2	37.3	58.6	56.8	59.2	65.9	99.0
Visualization Statistics													
P:#pixels×10^3		47.9	52.1	43.5	58.5	192.4	208.9	174.7	234.5	433.5	470.5	393.6	528.3
R:#ray seg.×10^3		48.3	52.4	47.1	59.3	193.7	210.3	189.2	238.1	436.3	473.7	426.3	536.3
I:#int.×10^3		3924	3412	4039	5209	15751	13696	16214	20908	35481	30850	36524	47097

Table 4.6: Execution-time dissection and visualization statistics of algorithms for view V_4.

Data set Name	Image res.	Algorithm: Koy.	SBRC	H-SBRC EA	H-SBRC BBA	MC
Blunt Fin	300×300	1.00	0.62	0.63	0.62	0.62
	600×600	1.00	0.44	0.48	0.48	0.39
	900×900	1.00	0.38	0.41	0.41	0.34
Comb. Cham.	300×300	1.00	0.87	0.94	0.97	0.70
	600×600	1.00	0.56	0.60	0.59	0.43
	900×900	1.00	0.45	0.48	0.48	0.36
Oxygen Post	300×300	1.00	0.59	0.59	0.57	0.60
	600×600	1.00	0.43	0.45	0.46	0.38
	900×900	1.00	0.38	0.40	0.42	0.33
Delta Wing	300×300	1.00	0.93	0.85	0.79	1.08
	600×600	1.00	0.63	0.62	0.61	0.55
	900×900	1.00	0.53	0.54	0.56	0.43
Averages						
avg. of 300×300		1.00	0.75	0.75	0.74	0.75
avg. of 600×600		1.00	0.51	0.54	0.53	0.44
avg. of 900×900		1.00	0.44	0.46	0.47	0.37
overall averages		1.00	0.57	0.58	0.58	0.52

Table 4.7: Execution times of algorithms normalized with respect to those of Koyamada's (for standard view V_1).

Data set Name	Image res.	Algorithm				
				H-SBRC		
		Koy.	SBRC	EA	BBA	MC
Blunt Fin	300x300	1.00	0.67	0.65	0.64	0.73
	600x600	1.00	0.48	0.51	0.49	0.46
	900x900	1.00	0.42	0.44	0.44	0.39
Comb. Cham.	300x300	1.00	0.93	1.00	0.96	0.78
	600x600	1.00	0.59	0.64	0.63	0.50
	900x900	1.00	0.49	0.52	0.52	0.43
Oxygen Post	300x300	1.00	1.03	0.83	0.79	1.67
	600x600	1.00	0.70	0.64	0.62	0.77
	900x900	1.00	0.57	0.55	0.56	0.55
Delta Wing	300x300	1.00	1.05	0.89	0.86	1.85*
	600x600	1.00	0.70	0.67	0.66	0.83*
	900x900	1.00	0.57	0.57	0.62	0.59*
Averages						
avg. of 300x300		1.00	0.92	0.84	0.81	1.26
avg. of 600x600		1.00	0.62	0.61	0.60	0.64
avg. of 900x900		1.00	0.51	0.52	0.53	0.49
overall averages		1.00	0.68	0.66	0.65	0.80

Table 4.8: Execution times of the proposed algorithms normalized with respect to those of Koyamada's (averages of four views). Values with * are averages of views V_1 and V_4 because of the cyclic meshes obtained in other views, V_2 and V_3.

Algorithm	Data set and Image resolution					
	Blunt Fin	Comb. Cham.	Oxygen Post			Delta Wing
	530x230	300x200	300x300	600x600	900x900	300x300
Koyamada	32.55	15.53	31.32	122.15	273.12	30.01
SBRC	15.25	13.75	18.58	52.93	104.90	27.87
H-SBRC$_{EA}$	16.04	14.87	18.33	54.81	109.91	25.36
H-SBRC$_{BBA}$	15.75	15.54	17.93	56.29	113.91	23.76
MC	13.68	11.17	18.86	46.76	90.97	32.32
LSRC (R10000)	22.00	19.00	37.00	82.00	136.00	64.00
LSRC (Sun)	*25.96*	*22.42*	*43.66*	*96.76*	*160.48*	*75.52*

Table 4.9: Performance comparison of presented algorithms with Lazy Sweep Ray-Casting (LSRC) algorithm for standard view V_1. LSRC (Sun) values represent the run-time estimates of LSRC on Sun Ultra Enterprise 4000 system computed using *specfp95* ratio of the Sun system to the SGI Power Challenge R10000 obtained from *http://www.specbench.org/osg/cpu95/results/cfp95.html.*

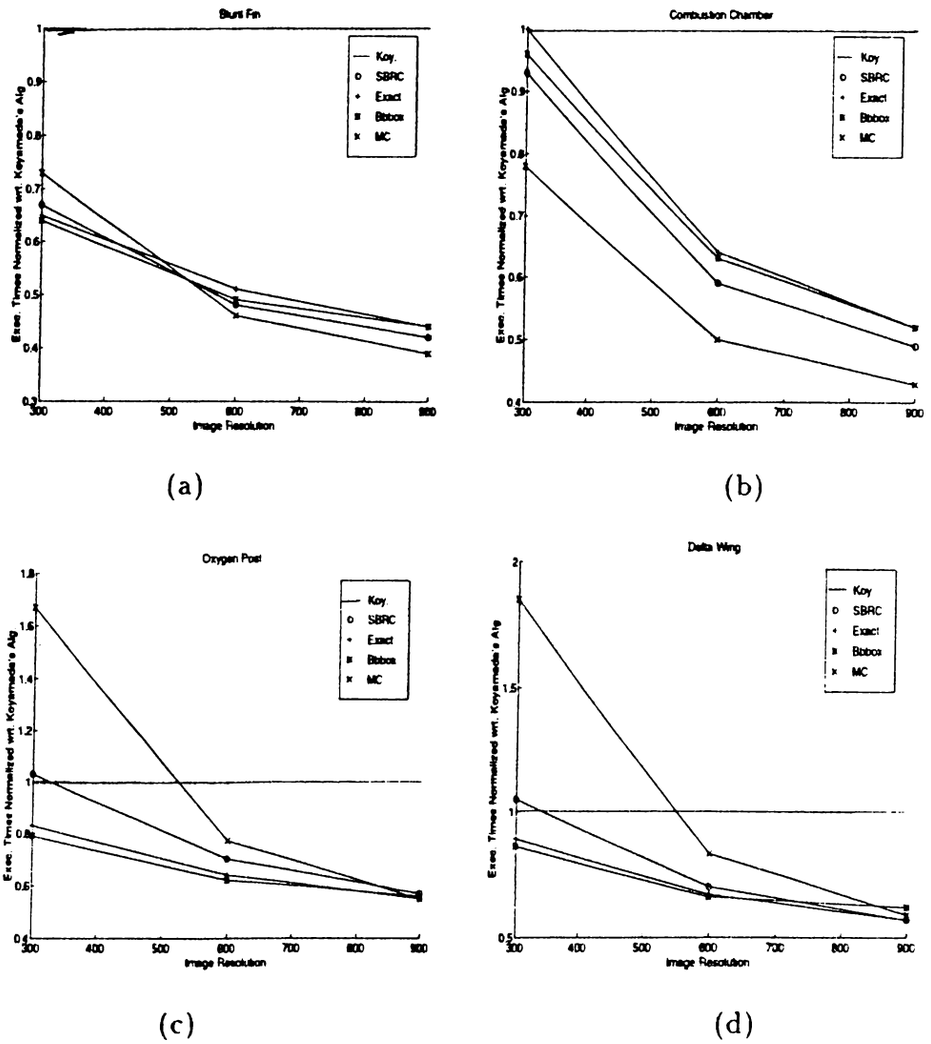


Figure 4.2: Variation of relative performances of the proposed algorithms with respect to Koyamada's algorithm with increasing resolution for (a) Blunt Fin, (b) Combustion Chamber, (c) Oxygen Post and (d) Delta Wing.

Chapter 5

Conclusion

In this thesis, three distinct categories, namely *image-space* methods, *object-space* methods and *hybrid* methods, have been investigated for fast direct volume rendering of unstructured grids. The main objective of the thesis was to identify the relative superiorities and inferiorities of the algorithms in these three categories both theoritically and practically. At least one algorithm from each category has been implemented and experimented in a common framework for software methods. All of the algorithms discussed and implemented in this framework have the same power to render unstructured grids including potentially disconnected and non-convex grids at a high accuracy.

We have defined the following quality measures for a fair comparison of the algorithms:

- Early Ray Termination: It is an optimization method used by many algorithms. The aim is to stop following the ray, when opacity reaches a user defined threshold.
- Generality: The capability of handling various types of meshes such as non-convex, cyclic, etc.
- Utilization of Coherencies:
  - Image-space Coherency (ISC): Ray-segments shot from nearby pixels are likely to pass through the same cells involving similar calculations

therefore, we can use this observation to speed up ray-face intersections.

- Object-space coherency (OSC): When a ray enters a cell, it must exit it through a back face of it. So using the connectivity relation, one can only check the neighbor cells to determine the next cell.

- Redundancy: Some algorithms perform redundant operations. Some of these operations have no impact on the image produced slowing down the process. We have identified two types of redundancies.
  - Redundancy in cell processing: It is clear that for the lighting model proposed here, only the cells that are intersected by at least one ray contribute to the final image. Therefore, cells that have no impact on the image should not be processed, but object-then-image (O-I) traversal compels the algorithm perform these redundant operations since this order enforces the processing of each cell.
  - Redundancy in ISC utilization: Although image-space coherency is very important and have an important impact on the speed of the algorithm, utilization of it for cells with small projection areas may be more costly then employing a naive approach. Therefore, algorithms should try not utilize ISC for cells where the cost exceeds the gains.

- Image Accuracy: The algorithms should produce correct images with no artifacts resulting from the algorithm itself.

Considering these quality measures and existing algorithms in the literature, we can say that image-space (I-O traversal) approaches support the following quality measures; early ray termination, generality, utilization of OSC, redundancies and image accuracy. Object-space (O-I traversal) methods support only the utilization of both object and image space coherencies, failing to support other quality measures. Hybrid approaches support generality, ISC and OSC utilization and image accuracy. The algorithms proposed in this thesis try to enlarge the number of quality measures supported by each category.

A well known image-space method, namely Koyamada's algorithm, has been implemented as efficiently as possible. One object-space method called Melting

Cells (MC) algorithm and one hybrid approach named as Span-Buffer Ray-Casting (SBRC) have been proposed and implemented. A fourth algorithm, namely Hybrid-SBRC (H-SBRC), stemmed from the idea of refining image-space and hybrid methods to extract the best features of each have been realized by blending Koyamada's and SBRC approaches.

Koyamada's algorithm, being a pure ray-casting approach, inherently exploits the object-space coherency available in the volume through connectivity relation. Furthermore, the algorithm handles the first ray-cell intersections very efficiently. 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. Therefore, Koyamada's algorithm, being one of the outstanding algorithms of image-space methods, has been selected as the representative of image-space approaches in this framework.

Melting Cells algorithm is similar to other object-space methods which utilize the image-space coherency and therefore, are faster than image-space methods. Although MC is slower with respect to other object-space methods, it differs from existing methods in its capability of producing high quality images which is a result of its different interpolation schema. Unlike other object-space methods, the algorithm handles the interpolations in face basis rather than cell basis thereby providing the superiority to yield high quality images. Thus, MC algorithm extends the set of quality measures supported by object-space methods by adding the image accuracy item. The sorting schema based on exploiting both image and object-space coherencies is another vital factor that contributes to its high performance.

Despite the high performance of object-space methods, their shortcoming in handling cyclic meshes have constituted the major motivation towards hybrid methods. Hybrid methods, in which the volume is traversed in object-space order and the contribution of each cell is computed in image-space order overcome this deficiency of the pure object-space methods, and in general, perform better than pure image-space-methods. However, first object-space then image-space computational order compels the existing schemes to process all

Category		Algorithm	Cyclic Meshes	Early Ray Term.	Coherence Utilization		Non-Redundancy in		Image Accuracy
					IS	OS	Cell Proc.	IS Coh. Util.	
Image-space (IS)		Koy.	√	√		√	√		√
Object-space (OS)		MC			√	√			√
Hybrid	I-O	SBRC	√	√	√	√	√		√
	I-O	H-SBRC	√	√	√	√	√	√	√
	O-I	LSRC	√		√	√			√

Table 5.1: Quality measure classification of software DVR methods for unstructured grids.

the volume data, thus preventing them to support early ray termination and furthermore making them suffer from redundant computations especially in low resolutions as the portion of data contributing to the image is relatively small. SBRC algorithm has been developed to overcome this deficiency of existing algorithms by changing the computational traversal order to first image-space and then object-space. This way the proposed SBRC algorithm gains the ability to support early ray termination, and avoids the redundant computations by processing only the contributing cells. Furthermore, SBRC algorithm fully utilizes both image and object space coherencies exploited by Koyamada's and MC algorithm without comprimising these nice features. Thus, it extends the set of quality measures supported by hybrid methods to include early ray termination, radundancy in cell processing.

Another disadvantage of object-space and hybrid methods is that they try to exploit image-space coherency even for small cells where the overhead of using this coherence does not afford the gains expected. The computational traversal order proposed by SBRC algorithm can be exploited to process such cells in a more cost effective way by employing a ray-casting schema which ignores image-space coherency, but still performs better. H-SBRC algorithm has been developed by this motivation blending SBRC and Koyamada's algorithms. Two heuristics, namely Exact Area (EA) and Bounding Box approximation (BBA), have been proposed which are used in determining the cell-processing schema to be employed. So H-SBRC algorithm supporting all quality measures supported by SBRC algorithm also supports the redundancy in ISC utilization item fullfillin all the quality measures. Table 5.1 summarizes the methods and the quality measures supported by each.

The current implementations of the presented algorithms have a potential to introduce a redundant memory overhead due to the way they handle the first ray-cell intersections. This potential may only be important in cases where resolution and the rate of non-convexity is high. Another contribution of the thesis is the proposal of optimization schemes to totally avoid this potential memory overhead without comprimising the run-time efficiency of the overall algorithms.

The algorithms discussed above have been implemented and experimented using tetrahedralized versions of curvilinear NASA data sets widely used for benchmarking rendering algorithms. The relative performances of the algorithms have been evaluated by the visualization of each of these data sets using four distinct views for three different resolutions for a better average case analysis.

Experimental results have shown that Koyamada's algorithm, despite its lack of using image-space coherency, performs considerably better than MC algorithm and slightly better than SBRC algorithm in large data sets and low resolutions. However, in the same experimental instances H-SBRC performs better than Koyamada's algorithm and is twice as fast as MC algorithm verifying our observations and expectations. Relative performances of all proposed algorithms increase with increasing resolution for the same data set, since the higher is the resolution the higher is the gain of exploiting image-space coherency. This rate of performance increase is more pronounced for MC. On the overall average performance analysis, in low and medium resolutions SBRC and H-SBRC yields the best performances whereas in high resolutions MC performs slightly better. The performances of SBRC and H-SBRC algorithms scales better than MC with increasing data set size. H-SBRC performs better than SBRC in low resolutions, but SBRC gradually and slightly overtakes H-SBRC in high resolutions. This behaviour can be attributed to the fact that in high resolutions, the projection areas of majority of cells get larger, thus resulting in the preferrence of SBRC approach for most of the cells and the performance of H-SBRC therefore reduces as it can not get the expected payoff. BBA heuristic always performs better than EA heuristic in low resolutions, and this performance difference decreases with increasing resolution

so that EA begins to perform better than BBA in high resolutions. This is mainly due to the fact that for low resolutions the errors introduced by BBA does not exaggerate the cell projection areas enough to cause bias misselect SBRC approach instead of Koyamada's approach. As resolution is increased, these errors cause bias to select SBRC for cells which should be processed by Koyamada's approach indeed, thus resulting in worse performance.

The respective performances of the presented algorithms have been compared with one of state-of-the-art algorithms called Lazy Sweep Ray-Casting algorithm. LSRC is also a hybrid approach and is reported to be two orders of magnitude faster than existing software algorithms. Considering the run-time estimates of LSRC algorithm on our benchmark machine, it has been seen that MC algorithm is 1.8 to 2.4, SBRC and H-SBRC algorithms are 1.5 to 3.2 times faster than LSRC algorithm on the overall average. Restricted reported results of LSRC algorithm reveals that it scales slightly better than SBRC and H-SBRC algorithms, on the other contrary SBRC and H-SBRC algorithms scale much better than LSRC algorithm with increasing data set size.

As a conclusion, we can say that:

- Image-space methods are slow in general, but their relative performance is better in low resolutions.
- Object-space methods are fast and especially for high resolutions their relative performance is very good, but they may perform unexpectedly bad at low resolutions, especially, for large data sets.
- Hybrid methods constitue the most promising category, because they may perform as fast as object-space methods at high resolutions, much faster than them in low and medium resolutions, and are much more flexible than them, in general.
- Image-then-object (I-O) approach performs better than object-than-image (O-I).
- On the overall average, H-SBRC algorithm is the fastest algorithm.
- H-SBRC algorithm satisfies all quality measures.

Chapter 6

Appendix A

In this appendix, the pseudo codes for the algorithms presented in this thesis are given. The pseudo codes are just to provide a better view for the algorithms. because for the sake of efficiency the actual algorithms may be implemented in different ways.

6.1 Pseudo code for Koyamada's Algorithm

for each front external face **f** *do*
  scan convert **f** *generating ray-segments in sorted order according to the z coor.*

for each scanline **y** *do*
  for each pixel **x** *in scanline* **y** *do*
    for each ray-segment **r** *shot at* **(x,y)** *do*
      read entry values (z_{in}, s_{in}) *from* **r**
    while **r** *does not exit the volume*
      find exit face **ef** *for* **r** *from cell* **c**
      calculate exit values (z_{out}, s_{out})
      resample from (z_{in}, s_{in}) *to* (z_{out}, s_{out})
      take compositon onto **p**
      $(z_{in}, s_{in}) := (z_{out}, s_{out})$

6.2 Pseudo code for Melting Cells Algorithm

```
for each front external face f do
   scan convert f generating ray-segments in sorted order according to the z coor.

//Generate Internal deoendencies
for each cell c do
   for each face f of c do
      if (IsFrontFace(f) and not(IsExternalFace(f))) then
         increment InDegree(c)

//Generate External deoendencies
for each scanline y do
   for each pixel x in scanline y do
      for each ray-segment r shot at (x,y) do
         if (not(FirstRaySegment(r))) then
            increment InDegree(c)

//Initialize Queue
for each cell c do
   if (InDegree(c)== 0) then
      Enqueue(Q,c)

//Process cells
while (not(IsEmpty(Q)))
   Dequeue(Q,c)
   for each face f of c do
      if (IsBackFace(f)) then
         decrement InDegree(NeighborCell(c,f))
         if (InDegree(NeighborCell(c,f))== 0) then
            Enqueue(Q,NeighborCell(c,f))
         scan convert f
         for each pixel p covered do
            read entry values (z_in, s_in) from CurRaySegment(p)
```

calculate exit values (z_{out}, s_{out})
resample from (z_{in}, s_{in}) *to* (z_{out}, s_{out})
take compositon onto **p**
CurRaySegment(p):= (z_{out}, s_{out})
*if (***IsExternalFace(f)** *and*
NextRaySegmentExists(CurRaySegment(p))*) then*
AdvanceToNextRaySegment(CurRaySegment(p))
decrement **InDegree(Cell(CurRaySegment(p)))**
*if (***InDegree(Cell(CurRaySegment(p)))**== 0*) then*
Enqueue(Q,Cell(CurRaySegment(p)))

6.3 Pseudo code for Span-Buffer Ray-Casting Algorithm

for each front external face **f** *do*
scan convert **f** *generating ray-segments in sorted order according to the z coor.*

for each scanline **y** *do*
for each pixel **x** *in scanline* **y** *do*
for each ray-segment **r** *shot at* **(x,y)** *do*
read entry values (z_{in}, s_{in}) *from* **r**
while **r** *does not exit the volume*
*if (not(***IsActive(CurrentCell(r))***))) then*
Activate(CurrentCell(r))
*if (not(***IsValidSpan(Span(CurrentCell(r)))***))) then*
CreateSpan(CurrentCell(r))
read exit values (z_{out}, s_{out}) *from span*
resample from (z_{in}, s_{in}) *to* (z_{out}, s_{out})
take compositon onto **p**
$(z_{in}, s_{in}) := (z_{out}, s_{out})$
kill active cells to be killed at this scanline

6.4 Pseudo code for Hybrid Span-Buffer Ray-Casting Algorithm

```
for each front external face f do
   scan convert f generating ray-segments in sorted order according to the z coor.

for each scanline y do
   for each pixel x in scanline y do
      for each ray-segment r shot at (x,y) do
         read entry values (z_in, s_in) from r
         while r does not exit the volume
            if (not(IsSchemaSelected(CurrentCell(r)))) then
               SelectSchema(CurrentCell(r))
            if (Schema(CurrentCell(r))==Koyamada) then
               employ Koyamada's alg. for current ray and cell
            else if (Schema(CurrentCell(r))==SBRC) then
               employ SBRC alg. for current ray and cell
   kill active cells to be killed at this scanline
```

Bibliography

[1] T. T. Elvins, "A survey of algorithms for volume visualization," *Computer Graphics*, 1992.

[2] A. Kaufman, W. E. Lorensen, and R. Yagel, "Volume visualization algorithms and applications," in *Volume Visualization.*

[3] N. Max, "Optical models for direct volume rendering," *IEEE Transactions on Visualization and Computer Graphics*, 1995.

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

[5] C. T. Silva and J. S. B. Mitchell, "The lazy sweep ray casting algorithm for rendering irregular grids," *IEEE Transactions on Visualization and Computer Graphics*, 1997.

[6] A. C. Pickover and K. S. Tewksburry, *Frontiers of Scientific Visualization.* 1994.

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

[8] T. Frühauf, "Ray casting of nonregularly structured volume data," in *EUROGRAPHICS '94. Eurographics Association.*

[9] 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.

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

[11] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes, *Computer Graphics, Principles and Practice, Second Edition.* Addison-Wesley, Reading. Massachussets, 1990.

[12] R. Wolff and L. Yaeger, *Visualization of Natural Phenomena.* Telos, 1993.

[13] S. M. Pizer, H. Fuchs, and C. Mosher, "Three dimensional shaded graphics in radiotherapy and diagnostic imaging," in *NCGA '86 Conf. Proc.*

[14] M. L. Rhodes and Y. Kuo, "Simple three dimensional image synthesis techniques for serial planes," *SPIE Medical Imaging II*, 1988.

[15] B. Wyvill, C. McPheeters, and G. Wyvill, "Data structures for soft object," *The Visual Computer.*

[16] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3d surface reconstruction algorithm," *Computer Graphics*, 1987.

[17] P. Cignoni, C. Montani, E. Puppo, and R. Scopigno, "Optimal isosurface extraction from irregular volume data," in *Symposium on Volume Visualization.*

[18] A. Watt and M. Watt. *Advanced Animation and Rendering Techniques, theory and practice.* Addison-Wesley, 1992.

[19] E. Tanin, "Comparison of image space subdivision algorithms for parallel volume rendering," Master's thesis, Dept. of Computer Engineering and Information Sci., Bilkent University, July 1995.

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

[21] J. F. Blinn, "Light reflection functions for simulations of clouds and dusty surfaces," in *Proceedings of SIGGRAPH82 in Computer Graphics.*

[22] N. Max, "Panel on the simulation of natural phenomena," in *Proceedings of SIGGRAPH83 in Computer Graphics.*

[23] R. Voss, "Fourier synthesis of gaussian fractals, 1/f noises, landscapes and flakes," in *Tutorial on State of the Art Image Synthesis v.10 , SIGGRAPH83.*

[24] J. T. Kajiya and B. P. V. Herzen, "Ray tracing volume densities," *Computer Graphics*, 1984.

[25] R. A. Drebin, L. Carpenter, and P. Hanrahan, "Volume rendering," *Computer Graphics*, vol. 22, no. 4, pp. 65–74, 1988.

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

[27] 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.

[28] R. Avila, L. Sobierajski, and A. Kaufman, "Towards a comprehensive volume visualization system," in *IEEE Visualization '92.*

[29] L. Sobierajski and R. Avila, "A hardware acceleration method for volume ray tracing," in *IEEE Visualization '95.*

[30] L. Westover, "Footprint evaluation for volume rendering," *Computer Graphics*, vol. 24, no. 4, pp. 367–376, 1990.

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

[32] T. J. Cullip and U. Neumann, "Accelerating volume reconstruction with 3d texture hardware,"

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

[34] A. Oppenheim and R. Schafer, *Digital Signal Processing.* Prentice Hall, 1975.

[35] H. Feichtinger and K. Gröheng, *Wavelets: Mathematics and Applications*, ch. Theory and Practice of Irregular Sampling. CRC Press, 1993.

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

[37] D. F. Rogers, *Procedural Elements for Computer Graphics*. McGraw-Hill. 1985.

[38] B. Tabatabai, E. A. Sessarego, and H. F. Mayer, "Volume rendering on non-regular grids," in *Proceedings of EUROGRAPHICS '94*, pp. 248–258. Eurographics Association, 1994.

[39] R. Yagel, "Volume rendering polyhedral grids by incremental slicing," tech. rep., 1993.

[40] R. Yagel, D. Reed, A. Law, P.-W. Shih, and N. Shareef, "Hardware assisted volume rendering of unstructured grids by incremental slicing," in *IEEE-ACM Volume Visualization Symposium.*

[41] N. Max, P. Hanrahan, and R. Crawfis, "Area and volume coherence for efficient visualization of 3d scalar functions," in *Computer Graphics (San Diego Workshop on Volume Visualization.*

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

[43] P. L. Williams and P. Shirley, "An a priori depth ordering algorithm for meshed polyhedra," tech. rep., Center for Supercomputing Research and Development, University of Illinois at Urbana-Champaign, 1990.

[44] A. V. Gelder and J. Wilhelms, "Rapid exploration of curvilinear grids using direct volume rendering," Tech. Rep. UCSC-CRL-93-02, Computer and Info. Sciences, University of California, Santa Cruz, USA, 1993.

[45] P. L. Williams, "Visibility ordering meshed polyhedra," *ACM Transactions on Graphics*, vol. 11, no. 2, pp. 103–126, 1992.

[46] C. Stein, B. Becker, and N. Max, "Sorting and hardware assisted rendering for volume visualization," in *Symposium on Volume Visualization.*

[47] P. Cignoni, C. Montani, D. Santi, and R. Scopigno, "On the optimization of projective volume rendering,"

[48] 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.

[49] 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.

[50] C. Giertsen, "Volume visualization of sparse irregular meshes," *IEEE Computer Graphics and Applications*, pp. 40–48, March 1992.

[51] C. Silva, *Parallel Volume Rendering Of Irregular Grids*. PhD thesis, State University of New York at Stony Brook, 1996.

[52] C. Silva, J. Mitchell, and A. Kaufman, "Fast rendering of irregular grids," in *IEEE-ACM Volume Visualization Symposium*.

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

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

[55] 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.

[56] T. H. Cormen, C. H. Leiserson, and R. L. Rivest, *Introduction To Algorithms*. MIT Press, 1994.

[57] J. Wilhelms and A. V. Gelder, "A coherent projection approach for direct volume rendering," *Computer Graphics*, vol. 25, no. 4, pp. 275–284, 1991.

[58] H. Edelsbrunner, *Algorithms in Combinatorial Geometry*. Springer-Verlag, Heidelberg, 1987.

[59] H. Edelsbrunner, "An acyclicity theorem for cell complexes in d dimensions," *Combinatorica*, 1990.

[60] H. Edelsbrunner, "An acyclicity theorem for cell complexes in d dimensions," in *Proceedings of ACM Symposium on Computational Geometry*.