



Techniques for visualizing 3-D FEM results using tetrahedral cells

Koji Koyamada

Abstract

Techniques for visualizing 3-D FEM results both globally and locally are presented. Many studies of these “volume visualization techniques” have recently made, most of them concerning the handling of regular volume datasets composed of orthogonal cubic cells. In contrast, the 3-D FEM results that we process are irregular volume datasets composed of arbitrarily shaped and irregularly arranged cells. There have been few studies of such irregular volume datasets, because it is difficult to calculate efficiently which cell contains a given point and to estimate efficiently a data value at an arbitrary point in a cell. These tasks become serious bottlenecks in interactive visualization. To solve the problems, we decompose the original 3-D FEM results into a set of tetrahedral cells and manage them in a tetrahedral model composed of a nodal data component, a cell topology component, a cell adjacency component, and an exterior face component.

The proposed techniques fall into two categories: those for overall viewers and those for detailed viewers. For overall viewers, isosurface display and cloudy representation (volume rendering) techniques have been developed. The isosurface display technique allows a series of isosurfaces to be generated interactively by searching for cells that intersect the isosurface. The searching employs an extrema-based marching tetrahedra algorithm. The volume rendering techniques were developed by using both a ray-casting approach and a splatting approach. A criterion is given for how they can be used properly on the basis of a benchmark test. For detailed viewers, data probing and stream line display techniques have been developed. These techniques are based on a new and efficient point location algorithm, which assumes the existence of a starting point in any cell, and they make it easy to specify a cell that contains a given point.

Finally, each of these techniques is integrated into a single package and applied to engineering problems. The isosurface display technique is used to visualize the stress and temperature distribution in the design of semiconductor chips. The volume rendering, stream line display, and data probing techniques

are used to visualize the simulated air flow field in a clean room. The volume rendering technique is also used to visualize the pressure distribution in the calculated in-cylinder flow of reciprocating engines.

Acknowledgements

I wish first of all to extend my sincere gratitude to the members of my dissertation committee for their encouragement, and to express my sincere appreciation to my advisor, Professor S. Eiho. He was a constant source of valuable advice, and his confidence and encouragement were truly inspiring. I also owe particular thanks to Assistant Professor K. Hachimura for carefully reading drafts of this paper and making many detailed comments and corrections.

Thanks for many enlightening discussions are due to T. Itoh, Dr. A. Doi, T. Miyazawa, A. Moriwaki, T. Murao, K. Sugimoto, S. Uno, and K. Shimizu of Tokyo Research Laboratory, IBM Research. I would also like to thank M. McDonald for his expert help in preparing this manuscript and for his faithful support.

Next, I wish to acknowledge the advice of the many anonymous reviewers of the refereed papers drawn from the material in this dissertation. I regret that I cannot thank them by name.

Finally, I am eternally grateful for the love and faithful support of my family.

Nomenclature

$\vec{u}$	Position vector of a set of local coordinates
$\vec{X}$	Position vector of a set of global coordinates
$S(\vec{X})$	Scalar data value at a position $\vec{X}$
$N_i(\vec{u})$	Interpolation function of the i th node point
C	Constant value for an isosurface
g_i	Difference between a scalar data value S_i and C at the i th node point
N_t	Number of generated triangles
N_v	Number of generated vertices
N	Number of tetrahedral cells
B	Brightness value
$\vec{D}$	Direction vector
$p(\vec{D}, \vec{D}')$	Phase function from $\vec{D}$ to $\vec{D}'$
ω	Solid angle
ρ	Number of particles per unit volume
ρ'	Number of particles per unit length along a tube whose radius is identical to the particle's
c	Luminosity per unit area of a particle projection
α	Opacity value

N_{re}	Number of times that viewing rays intersect front- or back-facing exterior faces
N_C	Number of times that viewing rays visit tetrahedral cells
N_S	Number of sampling points along viewing rays
N_e	Number of front- or back-facing exterior faces
N_r	Number of viewing rays that hit the volume
N_{pc}	Total number of times that a probing icon visits cells
N_m	Total number of times that a mouse-push takes place
N_{sc}	Total number of cells through which a stream line passes
N_i	Total number of iterations in the calculation of velocity vectors
$\vec{v}$	Velocity vector
$\vec{p}$	Position vector

Glossary

Numerical simulation	A method for solving a partial difference equation approximately by converting it into an algebraic form.
Finite element method (FEM)	A kind of numerical simulation method that builds algebraic equations by integrating the weighted error over the whole region, which is subdivided into volume cells.
Volume data	A set of data values that are usually defined at discrete points in 3-D space. The data values can be scalar, vector, or tensor.
Volume cell	A polyhedron at whose corners volume data values are defined. Usually, a “volume” consists of a set of volume cells, which may be tetrahedra, hexahedra, or prisms.
Cell topology	Information on the node points outlining a volume cell. Usually, it is described as a list of identifiers of node points.
Cell adjacency	Information on the cells neighboring a volume cell. Usually, it is described as a list of identifiers of volume cells.
Node point	A point at which a data value is defined in an FEM result. It is often a corner of a volume cell, and may be located on a cell edge.
Interpolation function	A weighting value that is defined at each node point, for interpolating volume data. A data value at an arbitrary point can be calculated by a linear combination of the weighting values and original data values at node points.

Local coordinate system	A normalized coordinate system that is defined for each volume cell and in which a node point is mapped to a predefined set of coordinates.
Exterior face	A volume cell face that is not connected to another volume cell.
Post-processor	A graphic software system for processing an output from a numerical simulation program, especially from a structural analysis program. Conventionally, it displays simple 2-D graphic outputs such as contour lines and arrow plots.
Volume visualization	A technique for constructing a meaningful picture from volume data.
Isosurface	A set of points that satisfy the equation, $S(\vec{X}) - C = 0$, where $S(\vec{X})$ is a scalar function in 3-D space.
Stream line	A set of points that are defined by the equation, $\frac{dx}{v_x} = \frac{dy}{v_y} = \frac{dz}{v_z}$, where $\vec{v} = (v_x, v_y, v_z)$ denotes a velocity vector field.
Volume rendering	A method for representing volume data as semitransparent density clouds, whose appearance can be easily modified by specifying a transfer function for mapping scalar data to colors and opacities.
Image order approach	A technique for implementing volume rendering. Also called ray-casting, it scans the display screen and, by casting a viewing ray, determines what volume cells affect each pixel.
Object order approach	A technique for implementing volume rendering. Also called splatting, it processes a list of volume cells in their visibility order and determines what pixels each volume cell affects.
Data probing	A technique for visualizing a data value at an arbitrary point in volume data.

Contents

Abstract	i
Acknowledgements	iii
Nomenclature	v
Glossary	vii
1 Introduction	1
1.1 Overview and Scope of This Study	1
1.2 Characteristics of volume data	3
1.2.1 Data type	3
1.2.2 Volume cell type	3
1.3 Previous Work	4
1.3.1 Scalar data visualization	5
1.3.2 Vector data visualization	13
1.3.3 Tensor data visualization	16
1.4 Objective of This Study	16
1.5 Outline of the Remaining Chapters	17
2 Interpolation technique for 3-D FEM results	19
2.1 Data structure	19
2.1.1 Node point	19
2.1.2 Volume cell topology	21
2.1.3 Data value	21

2.2	Data interpolation	21
2.2.1	Local coordinate system	22
2.2.2	Interpolation function	22
2.2.3	Interpolation algorithm	24
2.2.4	Use of tetrahedral cells as visualization primitives	26
2.3	Tetrahedronation	27
2.4	Tetrahedral model	30
2.5	Summary of this chapter	33
2.6	Appendix	33
2.6.1	Linear brick cell	33
2.6.2	Parabolic brick cell	35
2.6.3	Linear wedge cell	37
2.6.4	Parabolic wedge cell	38
2.6.5	Interpolation function	38
2.6.6	Linear tetrahedral cell	39
2.6.7	Parabolic tetrahedral cell	41
3	Isosurface display technique	45
3.1	Overview of marching tetrahedra	46
3.2	Triangular facet generation within a cell	47
3.2.1	Related work	47
3.2.2	Fully table-driven approach	48
3.3	Marching to the next cell	49
3.3.1	Related work	49
3.3.2	Extremum-based marching tetrahedra	50
3.4	Performance evaluation	54
3.4.1	Facet generation based on fully table-driven approach	54
3.4.2	Extremum-based marching tetrahedra	54
3.5	Summary of this chapter	56

3.6	Appendix	56
3.6.1	Proof	56
4	Cloudy representation technique	59
4.1	Light scattering equation	60
4.2	Brightness equation	61
4.3	Image order approach	63
4.3.1	Reducing the cost of ray-face intersection tests	65
4.3.2	Reducing the cost of interpolation of the sampled values	68
4.3.3	Integrating geometric data	71
4.3.4	Performance evaluation	75
4.4	Object order approach	85
4.4.1	Splatting with concentric spheres	86
4.4.2	Performance evaluation	94
4.5	Comparison of two techniques	102
4.6	Appendix	104
4.6.1	Solution of the brightness equation	104
4.6.2	Simplification based on integration by substitution	105
5	Data probing and stream line display techniques	107
5.1	Related work	107
5.2	Data probing technique	108
5.2.1	Spatial point location	108
5.2.2	Data interpolation	111
5.3	Stream line display technique	112
5.4	Performance evaluation	114
5.5	Summary of this chapter	116
6	System implementation and applications	119
6.1	Goal of IVS	119

6.1.1 Overall viewer	120
6.1.2 Detailed viewer	120
6.2 User interface of IVS	120
6.2.1 Overall viewer	120
6.2.2 Detailed viewer	122
6.3 Stereoscopic display capability	123
6.4 Application of IVS to engineering problems	124
6.4.1 Design of semiconductor chips	124
6.4.2 Control of air flow in a clean room	131
6.4.3 Design of reciprocating engines	135
7 Conclusion	137
7.1 Summary	137
7.2 Plan for extending this study	139

List of Figures

1.1	Types of volume datasets	4
2.1	Data structure of a 3-D FEM result	20
2.2	Volume coordinate system	23
2.3	Face subdivision of a parabolic brick cell	28
2.4	Parabolic brick and wedge cell sharing a face	29
2.5	Face subdivision of various kinds of volume cell	30
2.6	Example of a tetrahedral model	34
2.7	Linear brick cell	42
2.8	Parabolic brick cell	42
2.9	Linear wedge brick cell	43
2.10	Parabolic wedge cell	43
2.11	Linear tetrahedral cell	44
2.12	Parabolic tetrahedral cell	44
3.1	Signs given at node points in a tetrahedral cell	46
3.2	Overview of extremum-based marching tetrahedra	50
3.3	Searching from a minimum extremum point	52
3.4	Traversal of the partial graph of $G(V,A)$	53
4.1	Difference in brightness B between the two ends of a tube	61
4.2	Scan-conversion of exterior faces	67
4.3	First step of linear sampling method	69
4.4	Second step of linear sampling method	70

4.5	Third step of linear sampling method	72
4.6	Overview of integrated rendering	73
4.7	Fast algorithm for the image order approach	76
4.8	Conventional algorithm for the image order approach	77
4.9	Sample image of dataset no. 1	80
4.10	Sample image of dataset no. 2	80
4.11	Sample image of dataset no. 3	81
4.12	Sample image of dataset no. 4	81
4.13	Improvement in the performance when dataset no. 4 is used.	82
4.14	Approximate calculation of weighting values	84
4.15	Accurate calculation of weighting values	84
4.16	Volume rendering based on ray-casting	87
4.17	Volume rendering based on splatting	87
4.18	Overview of the splatting algorithm using concentric spheres	88
4.19	Outline of the algorithm (preprocess)	89
4.20	Extract triangles on the m th sphere	90
4.21	Outline of the algorithm (iso-distance surface generation)	91
4.22	Outline of the algorithm (Method-1,2,3)	92
4.23	Scan-conversion of triangles	93
4.24	Method-1 (approximate opacities, approximate luminosities)	96
4.25	Method-2 (accurate opacity, approximate luminosities)	96
4.26	Method-3 (accurate opacity, accurate luminosities)	97
4.27	Method-4 (accurate opacity, accurate luminosities, raytracing)	97
5.1	Spatial point location based on a cell traverse	109
5.2	Re-traverse for non-convex volume	110
5.3	Calculation of an intersection at a cell face	111
5.4	Integration of the velocity field within a cell	113
5.5	Calculation of a velocity vector in a cell	114

6.1	User interface of IVS overall viewer	122
6.2	User interface of IVS detailed viewer	124
6.3	Two vertically compressed images for stereoscopic display	126
6.4	FEM model for thermal stress analysis in a semiconductor chip (41,264 cells)	126
6.5	Thermal stress contour on the exterior faces	127
6.6	Iso-stress surfaces within the chip	127
6.7	Isothermal surface in the fluid region (55,800 cells)	129
6.8	Thermal distribution outside the heat-generating modules	129
6.9	Isothermal surfaces in the heat sink (9,800 cells)	130
6.10	Volume rendering of particle density (61,680 cells)	130
6.11	Localized visualization above the workbench	134
6.12	Visualization of vortical flow near the ceiling	134
6.13	Cloudy representation of the calculated pressure fields (372,500 cells)	136
6.14	Identification of low-pressure region under the valve	136

List of Tables

2.1	Tetrahedral model	32
2.2	Interpolation function values for a linear brick cell	35
2.3	Interpolation function values for a parabolic brick cell	37
2.4	Interpolation function values for a linear wedge cell	38
2.5	Interpolation function values for a parabolic wedge cell	40
2.6	Interpolation function values for a parabolic tetrahedral cell	42
3.1	Pre-computed intersection table (P:+.N:-)	47
3.2	Intersection table based on three states (-:N.0:Z.+:P)	48
3.3	Results of performance evaluation of facet generation techniques	54
3.4	Results of performance evaluation of cell traverse techniques	55
3.5	Comparison with brute-force algorithm	56
4.1	Statistics on benchmark tests for the image order approach	78
4.2	Results of performance evaluation	98
4.3	Rendering performance	100
4.4	Time costs of generating iso-distance surfaces	101
4.5	Confirmation of the third assumption	103
4.6	Average numbers of viewing rays that intersect a single cell	103
5.1	Statistics on tests for the data probing technique and the stream line display technique	116

Chapter 1

Introduction

1.1 Overview and Scope of This Study

The recent “downsizing” trend has made computer hardware miniaturized and complex. As a result, mechanical engineers have had to understand physical phenomena that were not often considered previously.

In complex hardware, there can be little heat exchange. A poor mechanical design may cause fatal damage by destroying an electrical part as a result of excess heating. To avoid this possibility, proper analysis of the thermal air flow in the hardware is required. Since it is very difficult to measure the flow field and the temperature distribution, numerical simulation techniques play a significant role in the analysis.

A miniaturized hardware product is very sensitive to air-borne particles in the manufacturing environment. Semiconductor chips and magnetic hard disks must be manufactured in a clean room, where higher cleanliness is required for each new design. Uniform improvement of the cleanliness is very costly. To reduce the cost, it is necessary to control the air flow in order to maintain high cleanliness not by increasing the ventilation but by changing the layout of the room. It is possible to measure the air flow field and the cleanliness, but the measurement may require a temporary stoppage of manufacturing operations. In this case, numerical simulation techniques are very important.

Many engineers have come to employ numerical simulation techniques in order to solve their engineering problems, because there has been a significant increase in available computing power, as well as improvements in numerical simulation techniques. Of these techniques, the finite element method (FEM) is often selected, because a region to be modeled is often a complex three-dimensional space in engineering

problems. In general, 3-D FEM produces a very large volume dataset. Although a visualization tool is very important in this situation, the conventional tool, called a post-processor, is not good for viewing the resulting volume dataset. It projects the dataset onto a 2-D space, such as a plane or an exterior face, in a non-interactive manner. Without extensive use of such projections, engineers might miss important information contained in the dataset. Furthermore, the tool's low performance prevents them from using it extensively.

For mechanical engineers to explore 3-D FEM results, effective and efficient visualization techniques should be developed. The scope of this study ranges from cell decomposition of 3-D FEM data into tetrahedral cells, through isosurface display techniques, volume rendering techniques, and stream line display techniques, to implementation and application of a system.

To give some background on this study, a bibliography of recent publications on work in several areas of volume visualization techniques is presented. In Section 1.3.1, research in the area of scalar data visualization is described. This section includes three broad categories of data visualization: surface representation, in which geometric surface primitives are fitted to the volume data; binary voxel representation, in which the volume data are thresholded to yield a binary array, which is displayed by treating it as opaque cubes having six polygonal faces; and cloudy representation, sometimes called volume rendering, in which the continuous nature of the volume data is emphasized without the use of geometric primitives. In Section 1.3.2, research in the area of vector data visualization is described. There are three categories of visualization: arrow representation, in which vector data are mapped onto arrow geometries and icons for vectors; stream line representation, in which a trajectory of passive particles in the flow field is generated; and flow topology representation, in which the topological characteristics of a vector field are displayed at critical points where the velocity vector is zero. In Section 1.3.3, research in the area of tensor data visualization is described. The proposed visualization techniques employ scalar or vector data visualization, and the tensor data are often converted into scalar or vector data in advance.

This background information is followed by a section outlining the specific objectives of the research described in this dissertation. Chapter 1 closes with an explanation of the organization of this dissertation.

1.2 Characteristics of volume data

It is appropriate at this point to pause and to introduce some definitions of volume data.

1.2.1 Data type

Data can best be classified into scalar, vector, and tensor values:

- Scalar data:
 - Data are values of functions of the form $S(x,y,z)$. They have a magnitude and no direction. Examples are temperature, pressure, density, and energy.
- Vector data:
 - Data are values of functions of the form $S_i(x,y,z)$, where $i=1,2,3$ for three dimensions. They have both magnitude and direction. Examples are velocity, acceleration, force, and magnetic field.
- Tensor data:
 - Data are values of functions of the form $S_i(x,y,z)$, where $i=1,2,3,4,5,6$ or $i=1,2,3,4,5,6,7,8,9$. Examples are stress and strain. The name tensor data implicitly denotes second-order tensor data. Tensor data of order zero and tensor data of order one are scalar data and vector data, respectively.

1.2.2 Volume cell type

Volume data values are usually defined on a grid made up of volume cells. A volume cell is a polyhedron whose corners are called node points or grid points and whose value varies between the points. Sometimes, the volume cell is simply described as a cell. Volume data may be defined as an array of voxels. A voxel is a kind of a volume cell within which data values are constant.

Volume cells can be divided into three types:

- Regular volume cell
- Curvilinear volume cell
- Irregular volume cell,

which are widely used for numerical simulation techniques. A volume dataset defined on one of the above volume cells is called a regular volume dataset, a curvilinear volume dataset, or an irregular volume dataset, respectively (see Figure 1.1).

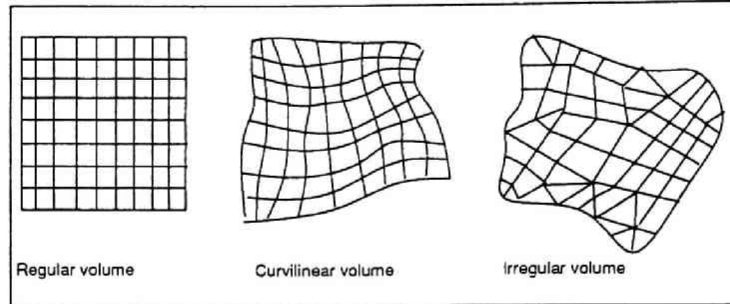


Figure 1.1: Types of volume datasets

Regular volume cells are rectangular bricks aligned with the axes, along which the distances between the node points are arbitrary. The locations of internal node points are defined implicitly by those on the axes and their topological information is also implicit. These volume cells are commonly used in the Finite Difference Method (FDM) and the Finite Volume Method (FVM).

Curvilinear volume cells allow non-boxy volumes to be gridded. Logically, they are Cartesian volume cells that are subjected to non-linear transformations so as to fill a volume or warp around a flow obstacle. The locations of node points must be specified explicitly but their topological information is implicitly defined. These volume cells are commonly used in FDM or FVM on a Boundary Fitting Coordinate system (BFC).

Unlike the previous types, where topological information is implicit, irregular volume cells have no geometric information implied by the lists of node points. Topological information must be supplied in some form. The volume cells may be tetrahedra, hexahedra, prisms, and so on, and they may be linear or higher-order. These volume cells are standard in the Finite Element Method (FEM).

1.3 Previous Work

In this section, a literature survey of pertinent research in the area of volume visualization will be presented. In keeping with the scope of the current research, emphasis is placed on the following visualization techniques:

- Isosurface display techniques
- Volume rendering techniques
- Stream line display techniques.

1.3.1 Scalar data visualization

Surface representation

Early methods for visualizing volume data concentrated on the generation of isosurfaces from the data. An isosurface is a three-dimensional surface within the volume on which the values of the data are the same. It is used to show the entire data distribution for numerical simulation, and the boundaries between structures, such as bone and soft tissue, for medical applications.

Contour-connecting: In applications involving a biological specimen that is cut with a microtome, photographs that are digitized, or medical imaging in which a CT scan is obtained, the end result is a stack of digital “slices.” The first step is to identify the contours in each slice. Initially manual tracing was used, but this has been replaced by automatic methods that use edge detection and region growing algorithms from image processing [Eiho84].

Once the contours have been identified, the problem becomes one of finding an optimal tessellation, usually of triangles, by connecting the curves in each two adjacent slices. Keppel reduced this problem to finding a path in a directed graph by using heuristics [Kepp75], and Fuchs et al. further specified the problem as that of finding a minimum cost path in a directed toroidal graph [Fuch77]. Both of those solutions find an approximation to the surface passing through high-gradient threshold cells in the data.

None of the early methods catered for bifurcations, where a contour in one slice divides into two or more contours in an adjacent slice. Initially, manual intervention was required to deal with bifurcations. In recent years, however, research has been directed towards the production of an algorithm that generates the surface completely in all cases. Christiansen and Sederberg [Chri78] manually introduced a node midway between the contours in the slice containing two contours and moved it halfway between the two slices. Today, very complex, multi-contour slices with multiple bifurcations are being routinely processed to yield three-dimensional surfaces [Ekou91, Chan91, Shin91a]. Most recently, attention has focussed on initially extracting the topological “skeleton” of a three-dimensional object by using a Reeb graph. This then determines between which contours the surface patches should be generated [Gier90, Shin91b].

A number of interrelated methods provide an alternative to the above methods for representing the three-dimensional surface. The TROTS program [Veen77] was used with biological specimens and displayed a stack of contours as solid slices after hidden-surface removal. The ISOSRF program [Wrig79] was designed for calculation of computational fluid dynamics data within a 3-D region. It generated contours in the three orthogonal xy , xz , and yz planes, which were combined to generate a display of an isosurface. In a related technique called Basket Weave [Sewe88], a surface is represented by a set of interleaving ribbon contours. This method is used to display surfaces described by implicit mathematical equations.

Marching Cubes: As alternatives to the above contour-connecting algorithms, table-based surface-fitting algorithms that yield large sets of cell-sized polygons have been presented. "Marching Cubes" is probably the algorithm that has been most widely implemented for generating isosurfaces within a regular volume dataset [Lore87].

Before the algorithms are applied, the user specifies a threshold value. The algorithm then loops on each successive group of four adjacent data slices. The slices are read into memory, gradients are calculated, and each cell between the middle two slices is scanned to determine whether its corner values straddle the threshold value. Non-straddling cells are discarded.

Cells that do straddle the threshold are more closely examined. The eight corners of the cube are numbered 1 through 8 and valued "1" if they are above the threshold and "0" if they are below the threshold. The eight corners of the cube are then put in eight consecutive bit locations (0-7) to form an eight-bit byte. This byte is treated as an index into a precomputed edge intersection table. The edge intersection lookup function returns 12 booleans, indicating which of the 12 edges of the cell are intersected by the isosurface. Interpolation is used to locate the edge intersected by the isosurface. If it is assumed that each edge can be intersected only once, then four triangles are sufficient to show the path of the isosurface through the cell. There are exactly 256 ways that four or less triangles can be fitted to a cell, and the number of cases can be reduced to 15 by reflection and rotation. Groups of three cell-edge intersection points are grouped to form triangles. The gradients at the intersection points are found by interpolating between the gradients at the corners of the cell. The interpolated gradients are stored with the triangles and later used for shading.

The Marching Cubes algorithm as originally described by Lorensen and Cline has ambiguous cases.

which were discussed by Durst [Durs88]. This problem has been resolved by Wilhelms and van Gelder [Wilh90b], who also reviewed a number of other approaches. They devised two implicit mathematical functions for testing the ability of various algorithms to determine the correct topology. The central cell in a $4 \times 4 \times 4$ array looks exactly the same for both functions; however, one represents a single connected surface and the other represents two lobes of a hyperboloid. This led Wilhelms and van Gelder to their proposition that in general it is impossible to determine the correct surface topologically in a cell solely by examining the data values at the vertices of that cell.

Wyvill, McPheeters, and Wyvill [Wyvi86] calculate the value at the center of the ambiguous face by using bilinear interpolation of the four values at the corners of that face. The obtained value is compared with the isosurface value and, depending on whether it is greater or less, one or other of the possibilities is chosen.

Wilhelms and van Gelder [Wilh90b] propose two techniques that use information in adjacent cells. Their more precise “quadratic fit” assumes that the underlying function can be locally represented as a quadratic function, and fits such a function by using a least-squares technique. This function is then used to compute a gradient which is added to the vertex values, giving an improved estimate of the function at the center of the face. Comparison with the isosurface value proceeds as above, but successfully copes with the implicit mathematical test functions.

A recent method proposed by Wallin [Wall91] uses local volume cells’ values to determine the topology of the surface. It is related to Marching Cubes but differs in that it examines the faces of grid cells independently and classifies them into one of 16 possible intersection cases. The surface polygons are then generated from faces with common edges.

Much earlier, Dayhoff [Dayh63] described a computer program for automatically generating contours in electron-density maps from the field of chemical crystallography. This treated each square in the grid in turn and was a two-dimensional version of the Marching Cubes algorithm, which might be called “Marching Squares.” It was refined by Heap and Pink [Heap69], whose method was a two-dimensional forerunner of that of Wyvill and McPheeters [Wyvi86]. Sutcliffe [Sutc76] discusses these techniques, as well as the two-dimensional equivalent of the ambiguity problem that occurs in Marching Cubes.

Dividing Cubes: Lorensen and Cline soon realized that the size of the generated triangles, when they are rendered and projected, is often smaller than that of a pixel. A new algorithm was invented to

take advantage of this observation. Dividing Cubes [Clin88] begins by traversing each cell in the volume. When a cell is encountered with corner-values that straddle the threshold, the cell is projected into screen space to determine whether it projects into an area larger than a pixel. If it does, the cell is divided into subcells, each of which is rendered as a surface point; otherwise, the entire cell is rendered as a surface point. Surface points are composed of a value, a location in object space, and a calculated gradient for shading. No intermediate surface primitives are used in the Dividing Cubes algorithm. Surface points are rendered into the image buffer by using a standard computer graphics hidden-surface algorithm such as painter's or z-buffer. Rendering the surface points in a pseudo-random order allows the user to see the quality of the image gradually improving. Rendering surface points instead of surface primitives saves a great deal of time, especially when surface-rendering hardware is not available. A hardware implementation of the Dividing Cubes algorithm is described by Cline et al. [Clin90].

Marching Tetrahedra: Another way of tessellating a regular grid is to use a tetrahedral cell. This resolves the Marching Cubes ambiguity problem by subdividing each cube into a number of tetrahedral cells, although it generates more triangles than the Marching Cubes algorithm. Koide, Doi, and Kajioka [Koid86] use a division into five tetrahedra so that each one is intersected unambiguously by the isosurface. Similar approaches with different tetrahedral subdivisions of a cube are adopted by Purvis and Culberson [Purv86], Upson [Upso86], and Koyamada [Koya88].

Binary voxel representation

In addition to the surface representation described previously, there is a straightforward scalar data visualization procedure called the Cuberille algorithm, which was devised by Herman and Liu [Herm79]. The Cuberille algorithm has been almost exclusively confined to medical imaging, where a binary segmentation is appropriate and the data sampling is averaged. The motivation was to reduce the long computation time that earlier techniques took to generate surfaces. This algorithm involves three main stages: resampling, boundary detection, and surface shading.

Resampling: The voxels in a three-dimensional CT scan volume are not generally cube-shaped, because the inter-slice distance usually exceeds the width of a pixel in the slice. The first stage in the Cuberille technique is to create cubic voxels from the regular grid. This is achieved by using trilinear interpolation to generate new slices such that the inter-slice distance for the new set of slices equals the

width of a pixel. However, resampling can introduce errors. Different classes of objects are characterized in CT scans by different ranges of data values. Fat has lower values than muscle, which has lower values than bone. If fat and bone were adjacent then an interpolation at the boundary would give an average value that would be in the range for muscle, and would not be correct.

Boundary detection: The second stage starts with choice of appropriate values for characterizing the object under investigation. This is a binary segmentation, as it results in voxels being divided into those that represent the object and those that do not. If the object is simple, it is possible to identify voxels in a 2-D slice so as to generate a boundary. These 2-D boundaries can be connected with triangular ribbons to form the isosurface. If the object is complex, as is often the case with CT scans, then surface detection must be fully three-dimensional.

In the volume dataset, a connected subset Q of the voxels represents the object and, in the method of Artzy, Frieder, and Herman [Artz81], the problem of detecting the surface of the object is treated as a problem of traversing a directed graph G . The nodes of the graph correspond to faces separating voxels in the subset Q from voxels not in Q , and connected subgraphs of G correspond to the surfaces.

In the Cuberille technique, the surface of the object is approximated by the faces of cubic voxels, which are square and of equal size. All light rays are assumed to be parallel; this means that there are only six possible face orientations with respect to the light source. Finally, an orthographic projection is used, which means that from a given viewpoint a maximum of three face orientations are visible. This reduces the computation required for hidden surface removal and allows simple surface shading methods to be used.

Surface shading: In medical imaging, accurately depicting an object is more important than generating a pleasing image. However, the surface of a medical object tends to be complex and minute details of the original surface have to be retained in its presentation. As a result, the number of faces in a Cuberille representation is usually very large; more than 500,000 faces are not uncommon. Consequently, the more accurate methods such as Phong shading, which need surface normals, are very expensive to compute. Therefore, the final stage in the Cuberille technique, that of surface shading, tends to be a compromise between preservation of detail and accurate shading.

The actual object surface is only approximated by the boundary surface, which is made up of faces of voxels. The role of surface shading in the Cuberille technique is to make the boundary surface appear as

similar as possible to the object surface under given lighting conditions. This can be reduced to estimating the normal to the object surface at a given point and applying it to the voxel face in the boundary surface corresponding to that point. Different methods estimate the surface normal in different ways, most of which have their origin in the work of Chen [Chen85].

Cloudy representation

Cloudy representation, which is generally called volume rendering, describes a given volume dataset as semi-transparent density clouds, whose appearance can be easily modified by specifying a transfer function for mapping scalar data to color (brightness) and opacity (light attenuation). The specification is performed by the user so that data values are related to meaningful colors, the part of the volume data most interesting to him/her is exposed, and the part that is not interesting to him/her is transparent. Images are formed from the resulting colored semi-transparent volume by blending together volume cells projected onto the same pixel on the picture plane. This projection can be performed in either image order or object order.

Image order approach: The image order approach is generally called “ray-casting,” because it scans the display screen and, by casting a viewing ray, determines for each pixel what volume cells affect it. The opacities and shaded colors encountered along the ray are summed to find the opacity and color of the pixel. Whereas in ray-tracing, viewing rays bounce off when they hit reflective objects, in ray-casting, they continue in a straight line until the opacity encountered by the ray sums to unity or the ray reaches the exterior of the volume data. No shadows or reflections are generated. Descriptions of ray-casting approaches appear in various studies [Levo88, Upso88, Sabe88, Drebb88].

When a viewing ray intersects a volume cell between node points, an interpolation may be performed to find an in-between value for the intersection point. Alternatively, the value stored at the nearest neighboring node point may be used, as in voxel-based approaches. A transfer function is then invoked to find the color and opacity that the user specified for this data value. The color is gradient shaded, and the opacity is attenuated by the magnitude of the gradient. Because the magnitude is usually a good indication of the strength of an isosurface within a cell, the result is a large color contribution to the pixel when the ray encounters a large difference of data values in the volume dataset.

Although the ray-casting approach provides a mechanism for displaying small or poorly defined fea-

tures, it suffers from a number of problems. High on this list is its computational expense. Since all volume cells participate in the generation of each image, the rendering time grows linearly with the size of the dataset. Fortunately, most volume data contain a significant amount of spatial coherence. This coherence can be used to speed up rendering by means of an octree decomposition of the volume data [Levo90a] or a quadtree decomposition of the image plane [Levo90b].

Another problem with ray-casting is its lack of versatility. Many applications require that polygonal data and volume data be integrated in a single image. Examples include the superimposition of flow obstacles over a volume-rendered flow field. One solution is to convert the polygon and the volume into a common format. Kaufman proposes a technique for 3-D scan-conversion of polygon data to yield a binary voxel array [Kauf87a]. An alternative approach is to directly display both types of data by using a hybrid rendering algorithm. Preservation of the original representations eliminates conversion artifacts. Examples of this approach include separate rendering of geometrically defined objects and volume data followed by a depth-sorted compositing post-process [Good89], and extension of a conventional ray tracer to handle volume data [John89, Levo90c, Miya92].

Initially, attention in volume rendering was focused on medical imaging. This promoted the development of techniques for regular volume datasets. Recently, efforts have been made to support the rendition of data stored in curvilinear volume datasets [Wilh90a] and in irregular volume datasets [Koya90, Garr90].

Object order approach: The object order approach processes a list of volume cells in their visibility order and determines for each volume cell what pixels it affects. Westover [West89, West90] introduced a splatting technique in which each voxel is transformed into screen space and then shaded. Each voxel's contribution to the image is calculated and composited by using a series of table lookups. The procedure is called splatting because it can be likened to throwing a snowball (voxel) at a transparent plate. The snow contribution at the center of impact is high and the contribution drops off further away from the center of impact.

The first step in the splatting algorithm is to determine in what order the volume should be traversed. The face of the volume and the corner of the face closest to the image plane can be found by passing the coordinates of the corners of the volume through the viewing matrix. Voxels are splatted according to their distance from the image plane, with the closest voxel being splatted first. The voxels in one slice are all splatted before the next slice is started. The value of each voxel is classified according to user-specified

color and opacity transfer functions. The resulting color is shaded by using gradient shading, and the opacity is attenuated by the normalized strength of the gradient.

The next step finds the contribution of the color to an image buffer by projecting the voxel into image space. A round filter called a reconstruction kernel is used to find the pixel extent of this contribution. For orthogonal viewing, a circular kernel can be calculated once and used for all of the voxels. However, for perspective viewing, a new oblique kernel must be calculated for every voxel. The projection of the kernel into the image buffer is called a footprint. The size of the footprint is made proportional to the size of the volume dataset and the size of the image to be generated, so that a small volume dataset can fill a large image. Once the footprint has been calculated, its center is placed at the center of the voxel's projection in the image buffer.

The shaded color and opacity values are blended into the image buffer at every pixel that falls within the area covered by the circular Gaussian footprint. Before a color value is blended with a pixel value, it is attenuated at the value of the Gaussian footprint at that particular pixel center. This has the effect of making voxel contributions higher when they are near the center of the projection and lower when they are far from the center. After all of the voxels have been splatted into the image buffer, the image is complete. When the opacity at a pixel in the image buffer reaches unity, then further splats will have no effect on it.

Splatting seems in many ways preferable to ray casting. It can take advantage of coherence when a voxel projects into many neighboring pixels. It can avoid some of the aliasing inherent in point-sampling approaches. It can process the image plane by plane, so that if the total rendering time is considerable, the viewer can gain useful information during the rendering process by watching the image being created. This is particularly useful if the image is drawn back to front, because regions of the image that might be obscured later are all visible at some point during rendering. These features attract many researchers. More recently, methods have been developed for rendering the projection of volume cells by using Gouraud-shaded and partially transparent polygons. Wilhelms breaks the projection of cubic cell into topologically uniform regions, and then computes the opacity at the vertices according to various projection formulae [Wilh91]. A similar approach can be applied to tetrahedra, which have the advantage that there are only two topologically distinct cases to consider [Shir90, Will92a]. Laur and Hanrahan have created a generalized Gaussian splatting algorithm based on approximating footprints

with a collection of Gouraud-shaded polygons [Laur91].

Another group of object order approaches is based on projecting a stack of volume slices onto a screen from front to back, blending each slice with the projection formed by previous slices. Drebin [Dreb88] transforms each slice from voxel space to pixel space by using 3-D affine transformation (shearing) [Hanr90, Schr91]. Koyamada extracts iso-distance surfaces by polygonal approximation and projects these surfaces onto the screen from back to front [Koya92b].

1.3.2 Vector data visualization

Arrow representation

One simple and common technique for visualizing three-dimensional vector fields is to choose a set of points in the field and draw arrows indicating the magnitude and direction of the vector at each point; this is known as hedgehogging. It has been used by Forrest [Forr69] and others as a means of displaying normal vector information at the vertices of a surface, and more recently it has been used in visualization software to display slices of three-dimensional flow fields. Hedgehogging, which is typically done with line segments originating at a surface and directed parallel to the vectors they express, is a three-dimensional analog of an old technique. Edmund Halley used pen strokes to show the prevailing wind directions on a map of the world's oceans [Hall86]. Short lines from each grid point have been used as well [Robe70]: line length was tied to field strength with results that were reasonable in regions of low field strength, but less so in regions of high field strength, where the picture became cluttered. Lavin and Cherveney have used the density of line segments, rather than their length, to display the field intensity [Lavi87]. Displaying slices of a field by using hedgehogging is most effective on a fast graphics workstation, so that the entire slice can be rotated and translated in real time. Klassen and Harrington describe a technique for rapidly obtaining hedgehogging images with shadows [Klas91].

Streamline display

By definition, the lines of a vector field have the property that their direction at any point coincides with the direction of the field at that point. The lines of the velocity vector field are called streamlines and the underlying equations are called streamline equations. The construction of streamlines sometimes provides a clearer representation of the flow field and can lead to a better understanding of the underly-

ing fluid motion than arrow representation. The main difficulty with their construction is the numerous calculations that must be performed and the development of the user interface for the placement of seed points (the starting points of streamlines).

Streamline tracking algorithms: The most widely used algorithms are those that depend on numerical integration techniques based on streamline equations. These techniques allow a streamline to be grown from a seed point by means of a sequence of small time steps. Various streamline calculation algorithms have been presented [Hols70, Mall76, Volp87, Yeun88, Buni88, Elia89]. The differences between the various methods stem from whether they deal with 2-D or 3-D flows, the form of velocity interpolation, the numerical integration method, the use of schemes for adapting step size, and the type of grid mapping used.

Errors can arise from several sources in calculating the path of a streamline by using numerical techniques. Some sources are beyond the control of the tracking algorithm; these include poor numerical convergence, truncation errors, and the effects of highly distorted grids. Those associated with the tracking process can be attributed to either the interpolation of the velocity field or the numerical integration of streamline equations.

Issues associated with numerical integration include the order of the method and the selection of a time step. The combination of high-order integration and small step size can lead to very high computational loads that detract from interactive visualization. Murman and Powell [Murm88] considered the effects of step size and the order of accuracy of the integration method.

Placement of seed points: Streamline display systems involve the placement of their seed points. There are several approaches to this task. One is to have the computer automatically place the seed points on the basis of analysis of the flow field. For example, it can be made to place seed points near critical points of the flow topology, which will be described later, or near local maxima of interesting scalars such as helicity [Yate91]. A second approach is to have the user specify in advance the positions of seed points, usually as textual arguments of some command. Although this may sound primitive, it is often the most useful approach. A third approach is simply to give the user flexible, rapid, and interactive control over the placement of seed points. Bryson and Levit developed a virtual environment to support the last two approaches by using a boom-mounted CRT headset and a glove controller [Brys92]. Haimes and Darmofal proposed a simple procedure whereby a streamline is spawned off from a cutting plane

constructed in the computational domain by a user [Haim91b].

Topology representation

The analysis and visualization of flow topology has been intensively investigated. In a number of papers, Helman and Hesselink [Hess87, Hess88, Helm89, Helm90, Helm91] describe the development of a system for visualizing flow topology. A general classification of flow fields is described by Chong [Chon90]. More recently, techniques similar to those of Helman and Hesselink has been described. Dickinson [Dick91a] addresses the interactive aspect of flow topology visualization, and Globus [Glob91] gives detailed information on how to implement a flow topology visualization program.

Flow topology analysis is based on critical point theory, which has been used widely to examine solution trajectories of ordinary differential equations. The topology of a vector field consists of critical points, where the velocity vector is zero, and integral curves and surfaces connecting these critical points. Images of a vector field topology display the topological characteristics of a vector field without displaying too much redundant information.

The positions of the critical points can be found by searching all cells in the flow field. Critical points can occur only in cells where all three components of the vector pass through zero. The exact position of a critical point can be calculated by recursively subdividing the cell, or by a numerical method such as Newton iteration [Glob91].

Once the critical points have been found, they can be classified. This is done by approximating the velocity field in the neighborhood of the critical point with a first-order Taylor expansion. Because the velocity at a critical point is zero, the velocity field in the neighborhood of a critical point is fully determined by the partial derivatives. The critical points can be classified according to the eigenvalues and eigenvectors of the derivatives into (1) a repelling focus, also repelling in the third dimension, (2) a repelling node, (3) a saddle, repelling in the third dimension, (4) an attracting focus, repelling in the third dimension (5) an attracting node, and (6) a center, repelling in the third dimension.

The classified critical points can be used as starting points for integral curves ("streamlines"), and the eigenvectors can be used as starting directions. This means that the starting point is a point on an eigenvector, very close to the critical point. The end points of the integral curves are also critical points, or points on the boundary of the flow domain. Because of numerical errors, some integral curves may

miss a critical point, or may enter an object in the flow. The first problem can be solved by attaching an integral curve to a critical point when it comes very close to the critical point. To solve the second problem, integral curves that enter an object can be restricted so that they follow the surface of the object.

1.3.3 Tensor data visualization

3-D tensor fields, which implicitly mean second-order tensor fields, often appear as stress tensor fields in the results of structural analysis. The tensor data consist of six or nine scalar functions whose independent visualization is possible but meaningless. To visualize the tensor data, discrete iconic techniques have been commonly used. Kerlick employs ellipsoidal icons to display real and symmetric tensor fields [Kerl90]. Haber uses tensor glyphs to visualize the time-varying stresses at a given location in problems of fracture dynamics [Habe90]. Dickinson recognizes the importance of recovering the directional information contained in the tensor data, and introduces the concept of tensor field lines, which are everywhere tangent to one of the eigenvectors [Dick91b]. Delmarcelle generalizes this idea in order to represent all the information present in the tensor data [Delm92]. They first reduce an unsymmetric real and complex Hermitian second-order tensor to a real and symmetric second-order tensor and a real vector field. Then, they focus on the specific problem of visualizing this kind of tensor data by introducing the concept of a hyperstreamline, which owes its name to an analogy with a conventional streamline.

1.4 Objective of This Study

The overall objective of this study is to provide a practical method for visualizing in an understandable way the volume dataset generated by 3-D FEM analysis programs. This goal can be accomplished by processing complex and large datasets efficiently and effectively. Each of the techniques developed here has the following specific goals:

1. It should use a flexible data model in which an irregular volume dataset can be handled efficiently. In 3-D FEM results, the cell may be a tetrahedral, wedge, or brick cell, whose faces need not be planar surfaces.
2. It should include a capability for giving overall features of the results. An isosurface or a volume slice can be a good abstraction of volume data, but it confines three-dimensional information to

a surface. To realize a capability for overall viewing, it is desirable that all data values should contribute to the final data representation. When an isosurface is used for the realization, the user must move it interactively in the direction of its gradient; when a volume slice is used, it must move in the direction of its normal. A volume rendering tool can meet these requirements naturally, because it produces an image of density clouds that emit light according to data values in three-dimensional space. An efficient algorithm is desired, because it generally takes a considerable time to generate a volume rendering image.

3. It should include a capability for interactively displaying the value of the data at an arbitrary point by interpolating a simulation result. Such a capability is called data probing. Efficient realization of a probing capability for irregular volume datasets is an open problem. With this capability, engineers can virtually measure data values in the same way as in their laboratory.
4. It should be applicable to real engineering problems that are solved by using 3-D FEM techniques, and its effectiveness should be confirmed.

Techniques for accomplishing these goals will be presented in the remaining chapters.

1.5 Outline of the Remaining Chapters

A data structure and interpolation techniques for 3-D FEM results are presented in Chapter 2. It is necessary to estimate the data value and the gradient at a position between raw data points. A particular algorithm should be employed according to the data characteristic of irregular volume cells. This chapter also contains a description of a tetrahedral model in which the visualization techniques described in the following chapters are implemented.

In Chapter 3, a table-driven technique for displaying isosurfaces according to the tetrahedral model is derived. This chapter includes a high-performance algorithm for generating series of isosurfaces in an interactive environment.

Chapter 4 presents techniques of volume rendering based on the tetrahedral model, following two approaches: image order and object order. The results of performance evaluation of four test cases from computational fluid dynamics (CFD) calculations are used in a discussion of how these two approaches can properly be implemented.

Chapter 5 proposes algorithms for probing and streamline display techniques based on cell traverse in a tetrahedral model. Performance evaluation tests show that this efficient traverse makes it possible to move freely around a large irregular volume dataset and to spawn off streamlines.

Chapter 6 discusses a system implementation based on the visualization techniques described in the previous chapters. A prototype called the Integrated Volume visualization System (IVS) is applied to the following engineering problems: design optimization for semiconductor chips, air flow control in a clean room, and design optimization for reciprocating engines.

Finally, a summary of this research appears in Chapter 7, and further work is discussed.

Chapter 2

Interpolation technique for 3-D FEM results

A technique for interpolating data values is essential to volume visualization techniques because volume data are defined on a set of discrete points (node points). For regular volume datasets, it is simple and efficient. For 3-D FEM results (irregular volume datasets), however, it is complex and inefficient in general. In this section, we propose an efficient interpolation technique for 3-D FEM results. First, we describe the characteristics of the data structure of 3-D FEM results. We then describe how 3-D FEM results are interpolated, and explain the effectiveness of using tetrahedral cells in volume visualization. Finally, we propose a tetrahedral model on which our visualization techniques are based.

2.1 Data structure

In general, a 3-D FEM result has three components (see Fig 2.1):

- Node point
- Volume cell topology
- Data value.

2.1.1 Node point

The node point component is an array object specifying a set of 3-D positions that are corners of volume cells. “Node” is a term commonly used in the FEM community, and is equivalent to “grid point.” Typically, each node point belongs to more than one volume cell, so the node points are stored in a separate array, allowing the volume cells to refer to the node points by identifier number.

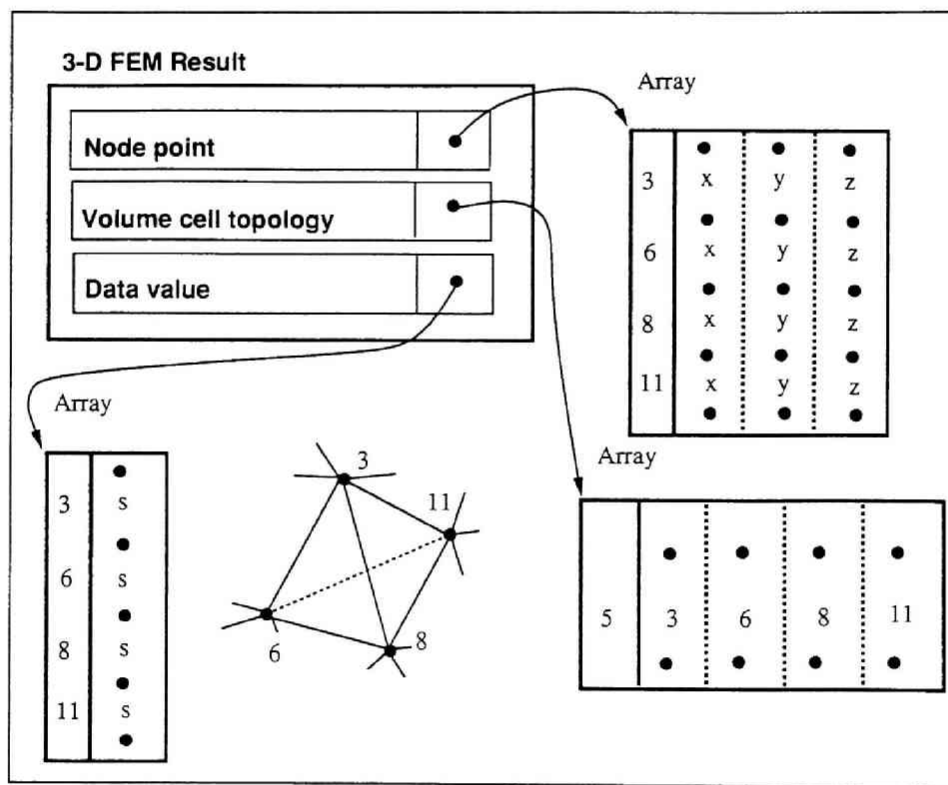


Figure 2.1: Data structure of a 3-D FEM result

2.1.2 Volume cell topology

The volume cell topology component describes the topological information contained in a volume cell by using its node points. Another term commonly used in the FEM community is “element,” which denotes a volume cell. Each item of the volume cell array describes a tetrahedron, a hexahedron, or some such object. The vertices of each volume cell are specified in the node point array by one array item consisting of a list of identifiers, with one identifier for each node point of the volume cell. In a 3-D FEM result, the corresponding region is filled with volume cells without gaps and overlaps, and a volume cell shares the node points with the neighboring volume cells. Therefore, an adjacency graph can be constructed from a given volume dataset.

2.1.3 Data value

The data value component is an array object specifying a set of the FEM result values. The data value may be defined at each node point or at each volume cell¹. The data type of the FEM results can be scalar, vector, or tensor. Our visualization techniques are intended only for scalar and vector data, because tensor data can be converted into these types of data. A real and symmetric tensor data value has three real eigenvalues and three real and orthogonal eigenvectors. A real asymmetric tensor data value and a complex Hermitian tensor data value can be reduced to a real and symmetric tensor data [Delm92]. For example, structural analysis generates stress tensor data. Principal stress values, which are eigenvalues for the tensor data, are employed for evaluation rather than the individual components of the tensor data, because many stress criterion theories such as the Tresca criterion theory and the Mises criterion theory are based on eigenvalues.

2.2 Data interpolation

In FEM analysis, a data value $S(\vec{X})$ at a point $\vec{X}$ in a volume cell is generally interpolated from nodal data S_i by using interpolation functions $N_i(\vec{u})$, which are expressed not in a global but in a local coordinate system.

$$S(\vec{X}) = \sum_{i=0}^{n-1} S_i \times N_i(\vec{u}), \quad (2.1)$$

¹Our visualization techniques assume the former case; in the latter case, the data are converted into the result at each node point by extrapolation, using the interpolation function of the volume cell.

where n is the number of node points forming the volume cell, $\vec{X}$ is the position vector of the global coordinates in the irregular cell, and $\vec{u}$ is the position vector of the local coordinates (u, v, w) .

2.2.1 Local coordinate system

The local coordinate system is a normalized coordinate system in which a node point is mapped to a pre-defined set of coordinates. In FEM calculation, two types of local coordinate system are used: a normalized orthogonal coordinate system and a volume coordinate system. In the former, a volume cell is mapped onto a regular cube whose primary diagonal points are $(1,1,1)$ and $(-1,-1,-1)$. The appendix shows how the following four kinds of volume cell are mapped:

- Linear brick cells, which are hexahedra composed of eight node points. Note that "linear" means that there are no node points on an edge except for the corners of the volume cell.
- Parabolic brick cells, which are hexahedra composed of twenty node points. Note that "parabolic" means that there is one node point on an edge apart from the corners of the volume cell.
- Linear wedge cells, which are triangular columns composed of six node points.
- Parabolic wedge cells, which are triangular columns composed of fifteen node points.

The second type of system is the volume coordinate system. This coordinate system is often used for FEM analysis based on tetrahedral cells, because the integral calculation can be simplified during the analysis. Each value of the volume coordinates (u, v, w, z) denotes the ratio to the entire volume of the volume of the partial tetrahedron composed of the point and the opposite triangular face of a node point (see Figure. 2.2). Therefore, one of the four coordinates can be automatically determined from the other three coordinates, because the equation $u+v+w+z=1$ holds. The appendix also shows how these volume cells are mapped both for the linear and parabolic cases.

2.2.2 Interpolation function

The value of the interpolation function in Eq. 2.1 should be 1.0 or 0.0 at the corresponding node point or at the other node points, respectively.

$$N_i(\vec{u}_j) = \begin{cases} 1.0, & \text{for } i = j \\ 0.0, & \text{for } i \neq j \end{cases} \quad (2.2)$$

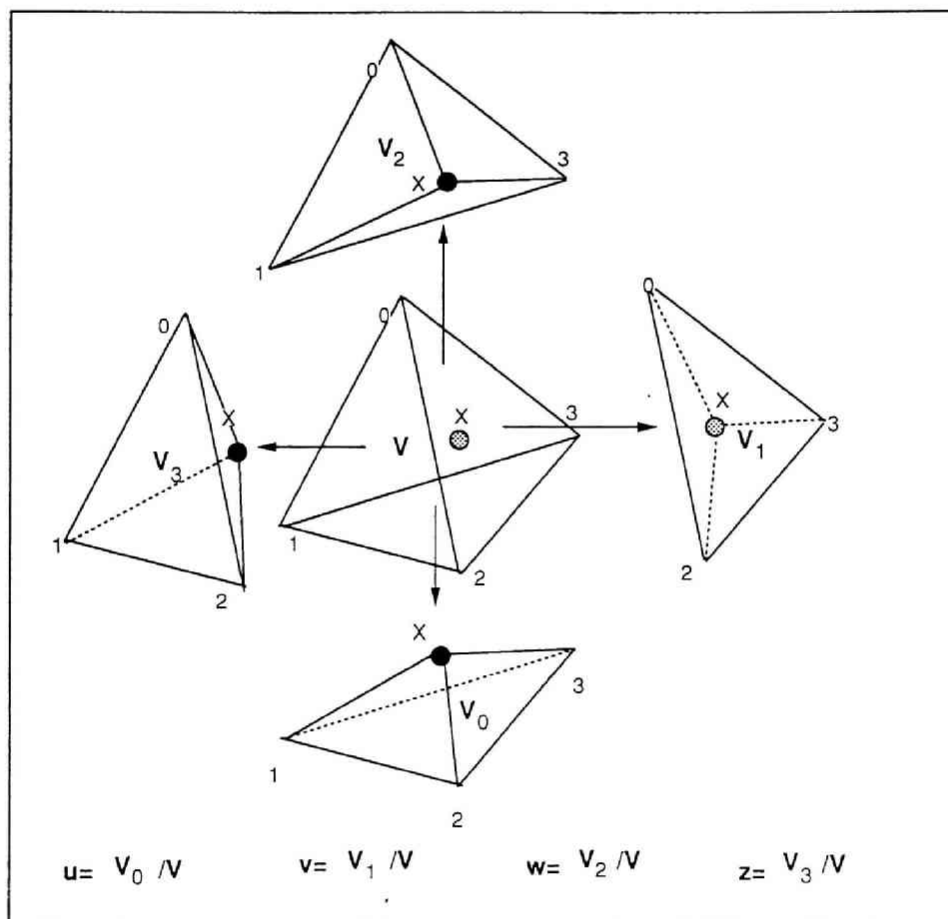


Figure 2.2: Volume coordinate system

$N_i(\vec{u})$ can be defined by using a linear combination of as many terms as there are node points in the volume cell. In the case of a linear brick cell, the terms are usually $(uvw, vw, wu, uv, u, v, w, 1)$. The coefficients for the terms can be determined by solving simultaneous equations based on the above relation Eq. 2.2.

Alternatively, the interpolation function $N_i(\vec{u})$ can often be determined by inspection if we assume that it is composed of a product of linear functions of the local coordinates.

1. Take the function

$$F_i(\vec{u}) = f_0(\vec{u}) \times \cdots \times f_j(\vec{u}) \times \cdots \times f_{k-1}(\vec{u}), \quad (2.3)$$

where $f_j(\vec{u}) = 0$ is an equation for a plane through three node points other than the i th member of all node points, and each node point other than the i th node point must be passed through by at least one of the k planes included in the above expression Eq. 2.3.

2. Divide the function by a value $S(\vec{u}_i)$ to obtain the interpolation function $N_i(\vec{u})$.

$$N_i(\vec{u}) = \frac{F_i(\vec{u})}{S(\vec{u}_i)} \quad (2.4)$$

In the appendix, a set of interpolation functions is shown for a brick, wedge, or tetrahedral cell (linear or parabolic).

2.2.3 Interpolation algorithm

In the FEM analysis procedure, integration is performed for each volume cell in the local coordinate system, as previously described. The calculation is relatively simple, because the interpolation function $N_i(\vec{u})$ is defined in the same coordinate system. On the other hand, in the FEM visualization procedure, the calculation is more complex. A coordinate transformation is required, because a position for interpolation is often given in a world coordinate system. For this transformation, Newton iteration is employed as follows:

1. Describe the coordinates of the given point by using the same interpolation functions on the assumption that the volume cell is isoparametric.

$$\vec{X} = \sum_{k=0}^{n-1} \vec{X}_k \times N_k(\vec{u}), \quad (2.5)$$

where $\vec{X}$ and $\vec{X}_k$ denote the position vectors for the given point and the k th node point in the volume cell, respectively.

2. Determine the initial value for the local coordinates, $\vec{u}_0$, and set the value of the index i to 0.
3. Calculate the world coordinate values $\vec{X}_i$ for the local coordinate values $\vec{u}_i$:

$$\vec{X}_i = \sum_{k=0}^{n-1} \vec{X}_k \times N_k(\vec{u}_i) \quad (2.6)$$

4. Describe the difference between the position vectors of the given point and the estimated point, $\Delta\vec{X}_i = \vec{X} - \vec{X}_i$, as the product of the Jacobian matrix $[J]$ and the displacement vector $\Delta\vec{u}_i$ to the solution:

$$\Delta\vec{X}_i = [J]^t \times \Delta\vec{u}_i \quad (2.7)$$

where

$$[J] = \begin{pmatrix} \frac{\partial x}{\partial u} & \frac{\partial y}{\partial u} & \frac{\partial z}{\partial u} \\ \frac{\partial x}{\partial v} & \frac{\partial y}{\partial v} & \frac{\partial z}{\partial v} \\ \frac{\partial x}{\partial w} & \frac{\partial y}{\partial w} & \frac{\partial z}{\partial w} \end{pmatrix}$$

and $[J]^t$ denotes a transpose of a matrix $[J]$. Each partial derivative is evaluated on the basis of Eq. 2.5. For example,

$$\frac{\partial x}{\partial u} = \frac{\partial}{\partial u} \sum_{k=0}^{n-1} x_k \times N_k(\vec{u}) = \sum_{k=0}^{n-1} x_k \times \frac{\partial N_k(\vec{u})}{\partial u}. \quad (2.8)$$

In the case of a linear brick cell (hexahedron), the partial derivative is expressed as follows:

$$\frac{\partial x}{\partial u} = \sum_{k=0}^7 x_k \times 0.125 \times u_k (1 + v_k v) (1 + w_k w). \quad (2.9)$$

5. Regard the above equations as simultaneous equations of each component in the displacement vector and solve them:

$$\Delta\vec{u}_i = [J]^{-t} \times \Delta\vec{X}_i \quad (2.10)$$

6. Add the calculated displacement vector $\Delta\vec{u}_i$ to the current local coordinates $\vec{u}_i$ and obtain the new local coordinates $\vec{u}_{i+1}$:

$$\vec{u}_{i+1} = \vec{u}_i + \Delta\vec{u}_i \quad (2.11)$$

7. Terminate the iterative process if the magnitude of the displacement vector $|\Delta\vec{u}_i|$ can be regarded as zero. If not, return to step 3 after incrementing i by 1.

The data interpolation is completed by substituting the local coordinate values calculated in the iterative process into Eq. 2.1. The scalar gradient, $\nabla S = \text{grad}(S)$, which is often required for shading calculation, can be estimated by using the calculated local coordinate values and the final Jacobian matrix, because it is described as follows:

$$\nabla S = \begin{pmatrix} \frac{\partial S}{\partial x} \\ \frac{\partial S}{\partial y} \\ \frac{\partial S}{\partial z} \end{pmatrix} = \sum_{i=0}^{n-1} S_i \times \begin{pmatrix} \frac{\partial N_i}{\partial x} \\ \frac{\partial N_i}{\partial y} \\ \frac{\partial N_i}{\partial z} \end{pmatrix} = \sum_{i=0}^{n-1} S_i \times [J]^{-1} \times \begin{pmatrix} \frac{\partial N_i}{\partial x} \\ \frac{\partial N_i}{\partial y} \\ \frac{\partial N_i}{\partial z} \end{pmatrix} \quad (2.12)$$

2.2.4 Use of tetrahedral cells as visualization primitives

The Newton iteration procedure requires a good guess for the initial values. If they are not close enough, the procedure may not converge. The convergence is also influenced by the degree of the interpolation function. The lower the degree, the more easily the procedure converges. For this reason, a linear tetrahedral cell is suitable. Since each component of the Jacobian matrix becomes a constant value, no iterative procedure is required and the data interpolation is straightforward. From Eq. 2.5, the coordinates of the given point can be described as

$$\begin{aligned} \vec{X} &= \begin{pmatrix} x \\ y \\ z \end{pmatrix} \\ &= u \times \begin{pmatrix} x_0 \\ y_0 \\ z_0 \end{pmatrix} + v \times \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix} + w \times \begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix} + (1 - u - v - w) \times \begin{pmatrix} x_3 \\ y_3 \\ z_3 \end{pmatrix}, \end{aligned}$$

where $\begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix}$ is the i th node point of the tetrahedral cell. Rearrangement based on the matrix formulation gives

$$\begin{pmatrix} x - x_3 \\ y - y_3 \\ z - z_3 \end{pmatrix} = \begin{bmatrix} x_0 - x_3 & x_1 - x_3 & x_2 - x_3 \\ y_0 - y_3 & y_1 - y_3 & y_2 - y_3 \\ z_0 - z_3 & z_1 - z_3 & z_2 - z_3 \end{bmatrix} \times \begin{pmatrix} u \\ v \\ w \end{pmatrix} \quad (2.13)$$

In 3-D FEM results that are successfully generated, a tetrahedral cell is never degenerated. Because the above matrix is not singular in this case, the local coordinates (u, v, w) can be obtained by multiplying both members of the above equation by the inverse matrix:

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{bmatrix} x_0 - x_3 & x_1 - x_3 & x_2 - x_3 \\ y_0 - y_3 & y_1 - y_3 & y_2 - y_3 \\ z_0 - z_3 & z_1 - z_3 & z_2 - z_3 \end{bmatrix}^{-1} \times \begin{pmatrix} x - x_3 \\ y - y_3 \\ z - z_3 \end{pmatrix} \quad (2.14)$$

From this result, the local coordinate values can be calculated directly for a given point, and from Eq. 2.1, a data value can be described as a linear function of x , y , and z :

$$S(\vec{X}) = P \times x + Q \times y + R \times z + C, \quad (2.15)$$

where P, Q, R, and C are determined by using coordinate values and data values at four node points.

From Eq. 2.12, the gradient vector becomes a constant vector:

$$\nabla S(\vec{X}) = \begin{pmatrix} P \\ Q \\ R \end{pmatrix}. \quad (2.16)$$

Note that in a tetrahedral cell, the data distribution is linear in any direction, which makes data interpolation along a straight line very efficient. Let us take a straight line parallel to a vector $\vec{a} = (a_x, a_y, a_z)$ through a point $\vec{b} = (b_x, b_y, b_z)$ in order to confirm this characteristic. By using a parameter t, a point (x,y,z) on the line can be expressed as

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = t \times \begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix} + \begin{pmatrix} b_x \\ b_y \\ b_z \end{pmatrix} \quad (2.17)$$

Substitution of the expression into Eq. 2.15 gives a linear function with respect to the parameter t:

$$S(t) = (P \times a_x + Q \times a_y + R \times a_z) + t \times (P \times b_x + Q \times b_y + R \times b_z + C) \quad (2.18)$$

Because the interpolation along a straight line appears often in volume visualization techniques, an efficient interpolation technique is indispensable. For this reason, we develop volume visualization techniques on the basis of tetrahedral cells². Various kinds of volume cell may be mixed in a 3-D FEM result. In this case, we subdivide these volume cells into a number of tetrahedral cells in a pre-process that we call tetrahedronation and describe it in the next subsection. In the visualization of 2-D FEM results, 2-D cells are often subdivided into triangles for contour plotting [Stel84, Yeo84]. Our approach is a natural extension of this subdivision to 3-D FEM results. The subdivision has side effects: the degradation of the accuracy in data interpolation and the increase of the required memory. Because our intention is to provide an interactive environment to visualize 3-D FEM results to engineers, we think that the stable and efficient data interpolation on the basis of tetrahedral cells is prior to the side effects.

2.3 Tetrahedronation

Koide and Doi developed a cell-based tetrahedronation technique for efficient extraction of an isosurface as a set of triangular facets [Koid86]. They subdivide each regular volume cell independently into five tetrahedral cells because, for a regular volume cell, there are some rules for subdivision that warrant alignment with the face of the adjacent cell. Since volume cells are not arranged regularly in a 3-D

²The tetrahedral cell has been used in many mesh generation techniques because of its geometric flexibility. Therefore most 3-D FEM results consist of tetrahedral cells.

FEM result, independent tetrahedronation can cause misalignment with the face of an adjacent cell. Our technique first subdivides a volume cell's face into triangles after information has been obtained about the face of the adjacent volume cell. Next, additional node points are generated within the volume cell. Finally, the resulting tetrahedral cells are constructed by using the triangles and node points.

Let us take a parabolic brick cell as an example for tetrahedronation (see Figure 2.3). First, each face of this volume cell is subdivided into six triangles. There can be two alignments in this subdivision. We store the information about which alignment is generated on the face so that the adjacent volume cell is subdivided in the same alignment. The identifier of one of the two nodes to which five edges are attached is selected to represent the alignment. We also store node identifiers representing the face. The identifiers of the first three out of four main nodes are selected in descending order of the values. Hashing [Ullm76] is employed to efficiently retrieve the stored information. In the hashing, the sum of the identifiers representing the face is used for the key value.

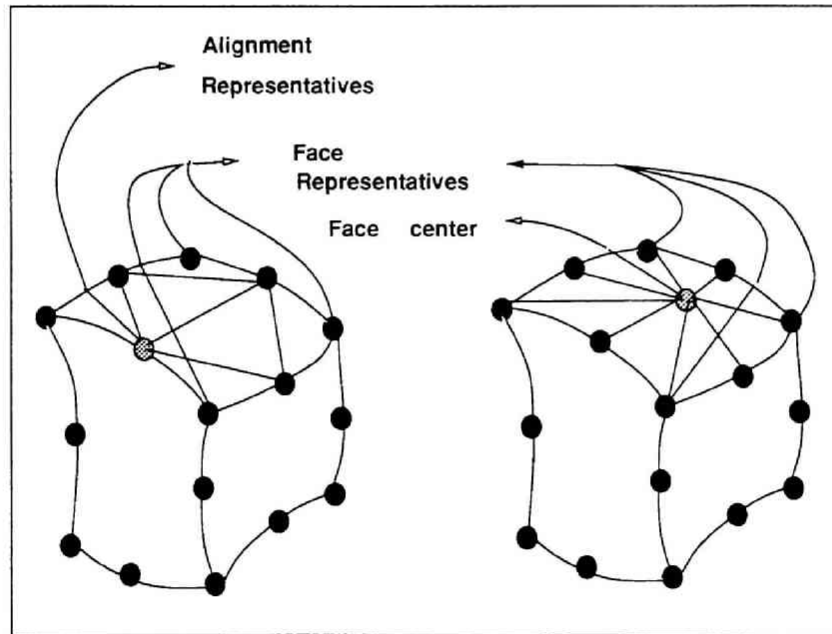


Figure 2.3: Face subdivision of a parabolic brick cell

Besides six-triangle subdivision, we can consider an eight-triangle subdivision accompanying an additional node generation at the center of a face by using the volume cell's interpolation function. The values of interpolation functions at the centers of various kinds of volume cell are shown in the appendix. By comparing a set of interpolation function values at the face centers, it can be confirmed that the node

points and the data values generated at the interface are coincident. Let us assume that a parabolic brick cell and a parabolic wedge cell are connected (see Figure 2.4). The face delimited by the nodes labeled 0, 1, 2, 9, 14, 13, 12, 8 of a parabolic brick cell is connected to the face delimited by the nodes labeled 4', 3', 2', 7', 11', 12', 13', 8' of a parabolic wedge cell. In both faces, the function values become

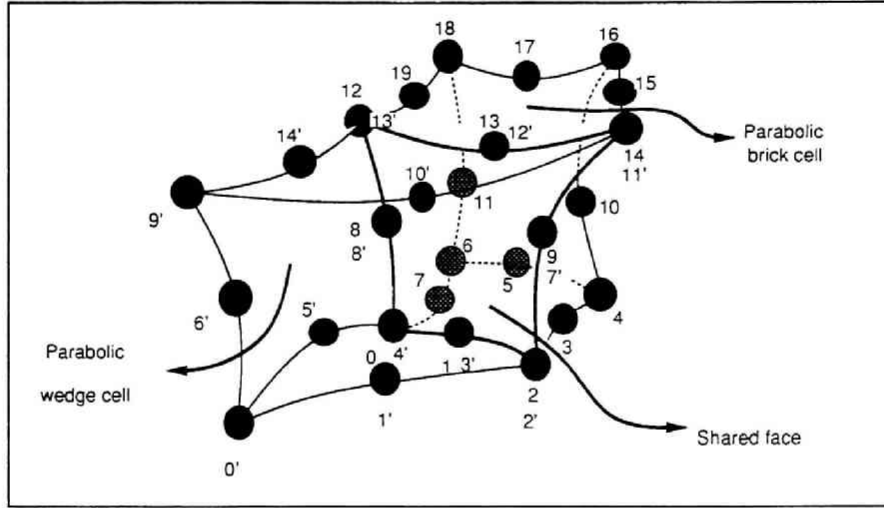


Figure 2.4: Parabolic brick and wedge cell sharing a face

$(-\frac{1}{4}, \frac{1}{2}, -\frac{1}{4}, \frac{1}{2}, -\frac{1}{4}, \frac{1}{2}, -\frac{1}{4}, \frac{1}{2})$. This subdivision is axi-symmetrical around the face center at every ninety degrees, so no misalignment of the subdivision occurs. In this subdivision, we store the identifier of the newly generated node point in order to avoid duplicate generation of a node.

Next, we independently generate an additional node at the center of a volume cell by using the volume cell's interpolation function. The values of interpolation functions at the volume centers of various kinds of volume cells are also shown in the appendix. Note that there are negative values in the case of parabolic cells. This shows that a function value may be negative at the volume center even if the function values at the node points are all positive.

Finally, we generate tetrahedral cells by connecting the generated node and the vertices of thirty-six or forty-eight triangles. For other types of volume cells, we also generate tetrahedral cells in a similar manner. In Figure 2.5, we show the face subdivision related to the tetrahedronation of various kinds of volume cells.

Recently, p-type FEM solvers have become increasingly popular, mainly for structural analysis aimed at obtaining a well-converged solution. They use higher-order polynomials as interpolation functions.

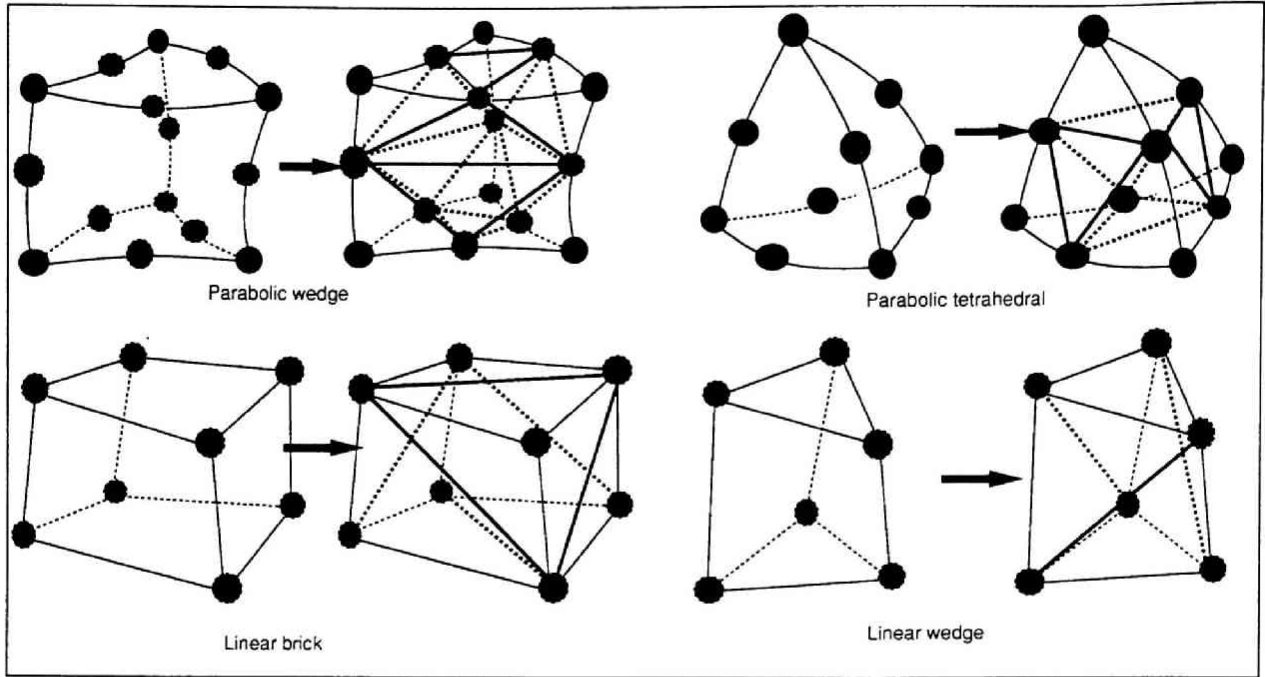


Figure 2.5: Face subdivision of various kinds of volume cell

rather than parabolic cells. In this case, two approaches to tetrahedronation can be considered. One approach is the same as that described previously, where each face is first subdivided into a number of triangles and the resulting tetrahedral cells are then generated by using additional node points. Although this process will be more complex than before, only one tetrahedronation rule for each order need be prepared in advance, because tetrahedronation can be performed in the local coordinate system. In the other approach, a high-order volume cell is regarded as a linear cell, and a simple subdivision process is first introduced. Each tetrahedral cell is then incrementally subdivided into eight tetrahedral cells according to a user-specified criterion. In this approach, new node points and data values need to be generated at the mid-points of the edges. The interpolation function of the original high-order volume cell must be used; it is meaningless if the generation is performed by using the interpolation function of the resulting tetrahedral cell.

2.4 Tetrahedral model

For the visualization of an irregular volume dataset, our algorithms assume a set of tetrahedral cells and their exterior faces, which we call the tetrahedral model [Koya91b]. In the tetrahedral model, each face

is a triangle, whose normal we define to be oriented outward from the parent cell. An exterior face is a face of a cell that is not shared by any other cell in a volume. Exterior faces do not overlap, and their union forms the boundary of the volume. An irregular volume cell is not necessarily a tetrahedral cell. If a visualized volume contains cells other than tetrahedral cells, we subdivide those cells into a number of tetrahedral cells by using the tetrahedronation technique previously described. The tetrahedral model has four components (see Figure 2.1):

- A nodal data component, whose entry contains position coordinates, scalar data values, and vector data values.
- A cell topology component, whose entry contains the identifiers of its four nodal data components.
- A cell adjacency component, whose entry contains the identifiers of the four cells connected to a cell through its four faces. If no cell is connected to a face, an exterior face identifier with a minus sign is placed in the table instead. In the irregular volumes formed by three-dimensional FEM results, cell adjacency information is not explicitly expressed, because procedures for numerical simulation do not require this relationship. In this case, the cell adjacency is assumed to be generated from information on the cell's topology.
- An exterior face component, whose entry contains the identifier of the parent cell and the face number.

In the above expression, a face number ($fn=0.1.2.3$) is defined in each cell by the following face-node relations:

- face 0 = $\{node_1, node_2, node_3\}$
- face 1 = $\{node_2, node_3, node_0\}$
- face 2 = $\{node_3, node_0, node_1\}$
- face 3 = $\{node_0, node_1, node_2\}$,

where $node_i$ denotes the identifier of the i th node in a cell. Generally, a face fn is composed of $node_{mod(fn,4)}$, $node_{mod(fn+1,4)}$, and $node_{mod(fn+2,4)}$, where $mod(i,4)$ denotes the remainder when i is

```

struct    size    {
    int      nt=      NTETRA; /*number of cells*/
    int      ne=      NNODE;  /*number of nodes*/
    int      ne=      NEXT;   /*number of exterior faces*/
    } SIZE;

struct    tetrahedron {
    int      n[4];          /*node IDs*/
    int      ad[4];        /*adjacent cell IDs*/
    } T[NTETRA];

struct    nodal-data {
    float    p[3];          /*position coordinates*/
    float    S;            /*scalar value*/
    float    v[3];          /*vector values*/
    } N[NNODE];

struct    exterior    {
    int      pid;          /*parent cell ID*/
    int      fn;          /*face number*/
    } E[NEXT];

```

Table 2.1: Tetrahedral model

divided by 4. In this definition, a face with an odd number and a face with an even number are inconsistent with respect to the directions of their normal vectors, since these are oriented outward and inward from the parent cell. However, our algorithms are not affected by this inconsistency.

The storage requirement is as follows:

$$Storage = 32 \times NTETRA + 28 \times NNODE + 8 \times NEXT(\text{bytes}), \quad (2.19)$$

where NTETRA, NNODE, and NEXT denote the numbers of tetrahedral cells, node points, and exterior faces, respectively.

Figure 2.6 shows a tetrahedral model that consists of two tetrahedral cells. A cell labeled 1 is composed of nodes labeled 1, 5, 4, and 2, and is connected to a cell labeled 2 at face 0. Faces 1, 2, and 3 are exterior faces labeled 1, 2, and 3. A cell labeled 2 is composed of nodes labeled 2, 4, 5, and 3, and is connected to a cell labeled 1 at face 3. Faces 0, 1, and 2 are exterior faces labeled 4, 5, and 6. The viewing ray first intersects the model at the exterior face labeled 1. By referring to the exterior face table, we find that it is face 1 of the cell labeled 1. The viewing ray exits from face 0, which is found by referring to the adjacency information to be identical with face 3 of the cell labeled 2. Next, it exits from face 0 of the

cell labeled 2, which is found to be identical with the exterior face labeled 4. We illustrate this traverse schematically in the lower part of the figure.

2.5 Summary of this chapter

In this chapter, we have discussed the data structure of a 3-D FEM result and described problems that might be faced when efficient interpolation techniques are developed.

- Generally, data interpolation includes an iterative procedure that can be unstable and time-consuming.
- In the original FEM result, the information on “who is neighbor of whom” is not straightforward.

Our solutions to these problems are as follows:

- Decompose each volume cell into tetrahedral cells.
- Generate a list for adjacent volume cells.

In order to realize the solution, we proposed a tetrahedral model. In the following chapters, we will describe volume visualization techniques based on this model.

2.6 Appendix

2.6.1 Linear brick cell

Local coordinate system

In a linear brick cell, each node point, $\vec{X}_i = (x_i, y_i, z_i), i = 0, 1, 2, 3, 4, 5, 6, 7$, is mapped to the local coordinate system (see Figure 2.7).

Interpolation function

In a linear brick cell, a function $N_0(\vec{u})$ can be defined as

$$N_0(\vec{u}) = 0.125 \times (1 - u)(1 - v)(1 - w)$$

by considering three planes:

- $f_0(\vec{u}) = 1 - u = 0$, which is a plane through node points 1, 2, 5, and 6
- $f_1(\vec{u}) = 1 - v = 0$, which is a plane through node points 2, 3, 6, and 7

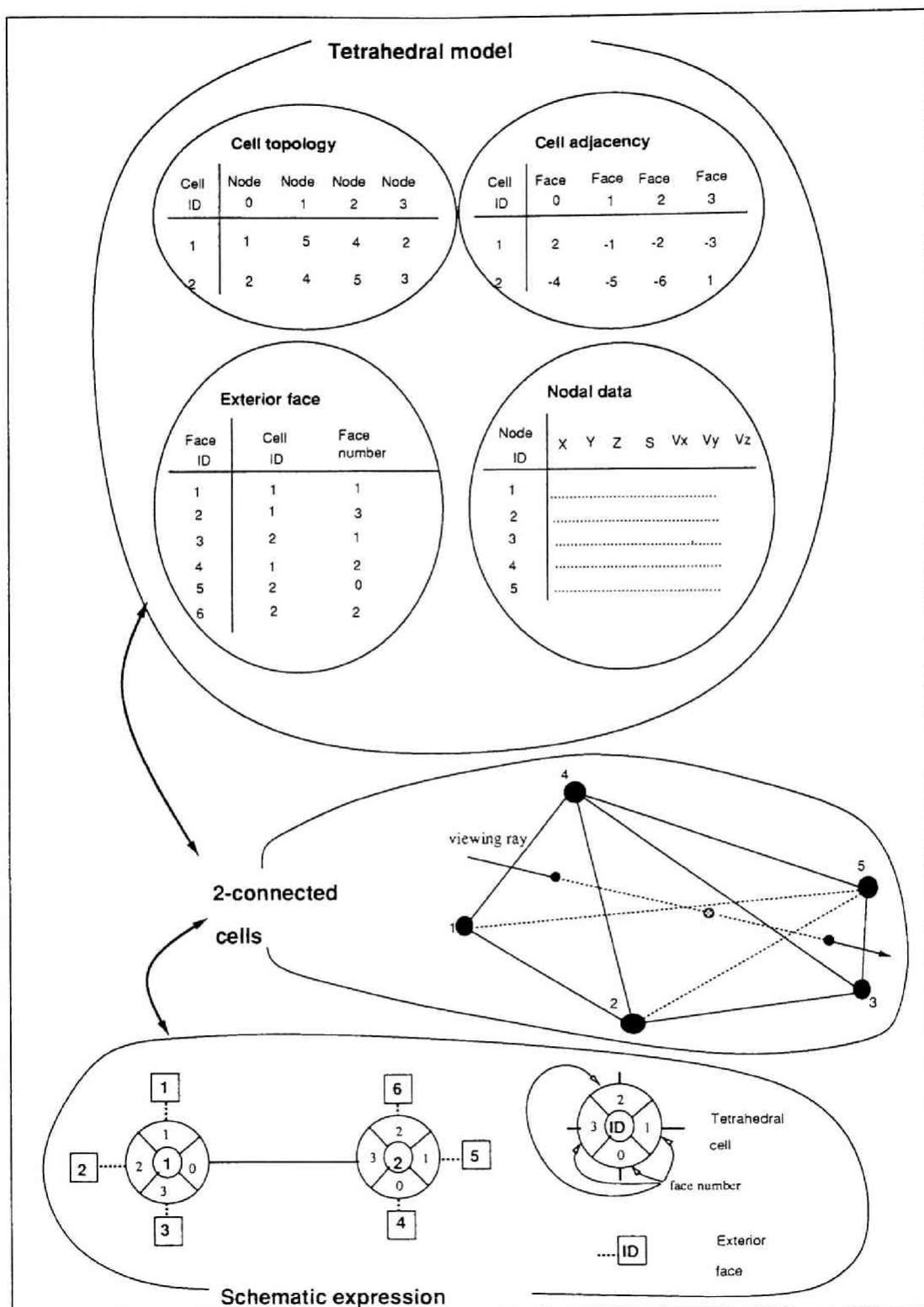


Figure 2.6: Example of a tetrahedral model

Table 2.2: Interpolation function values for a linear brick cell

Location	Face center0	Face center1	Face center2	Face center3	Face center4	Face center5	Volume center
(u,v,w)	(0,0,1)	(0,0,-1)	(0,1,0)	(0,-1,0)	(1,0,0)	(-1,0,0)	(0,0,0)
N_0	0	$\frac{1}{4}$	0	$\frac{1}{4}$	0	$\frac{1}{4}$	$\frac{1}{8}$
N_1	0	$\frac{1}{4}$	0	$\frac{1}{4}$	$\frac{1}{4}$	0	$\frac{1}{8}$
N_2	0	$\frac{1}{4}$	$\frac{1}{4}$	0	$\frac{1}{4}$	0	$\frac{1}{8}$
N_3	0	$\frac{1}{4}$	$\frac{1}{4}$	0	0	$\frac{1}{4}$	$\frac{1}{8}$
N_4	$\frac{1}{4}$	0	0	$\frac{1}{4}$	0	$\frac{1}{4}$	$\frac{1}{8}$
N_5	$\frac{1}{4}$	0	0	$\frac{1}{4}$	$\frac{1}{4}$	0	$\frac{1}{8}$
N_6	$\frac{1}{4}$	0	$\frac{1}{4}$	0	$\frac{1}{4}$	0	$\frac{1}{8}$
N_7	$\frac{1}{4}$	0	$\frac{1}{4}$	0	0	$\frac{1}{4}$	$\frac{1}{8}$

- $f_2(\vec{u}) = 1 - w = 0$, which is a plane through node points 4, 5, 6, and 7.

Generally, the interpolation functions, $N_k(\vec{u})$ ($k=0, 1, 2, 3, 4, 5, 6, 7$), can be described as

$$N_k(\vec{u}) = 0.125 \times (1 + u_k u)(1 + v_k v)(1 + w_k w).$$

2.6.2 Parabolic brick cell

Local coordinate system

A parabolic cell is a cell for which the mid-point of each edge is defined in order to improve the accuracy of the calculation. In a parabolic brick cell, each node point, $\vec{X}_i = (x_i, y_i, z_i)$, $i=0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19$ is mapped to the local coordinate system (see Figure 2.8).

Interpolation function

In a parabolic brick cell, a function $N_0(\vec{u})$ can be defined as

$$N_0(\vec{u}) = 0.125 \times (1 - u)(1 - v)(1 - w)(-u - v - w - 2)$$

by considering four planes:

- $f_0(\vec{u}) = 1 - u = 0$, which is a plane through node points 2, 3, 4, 10, 16, 15, 14, and 9
- $f_1(\vec{u}) = 1 - v = 0$, which is a plane through node points 4, 5, 6, 11, 18, 17, 16, and 10
- $f_2(\vec{u}) = 1 - w = 0$, which is a plane through node points 12, 13, 14, 15, 16, 17, 18, and 19
- $f_3(\vec{u}) = -u - v - w - 2 = 0$, which is a plane through node points 1, 8, and 7.

Generally, the interpolation functions $N_k(\bar{u})$ ($k=0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19$) can be described as

$$N_0(\bar{u}) = 0.125 \times (1-u)(1-v)(1-w)(-u-v-w-2)$$

$$N_1(\bar{u}) = 0.25 \times (1-u)(1+u)(1-v)(1-w)$$

$$N_2(\bar{u}) = 0.125 \times (1+u)(1-v)(1-w)(u-v-w-2)$$

$$N_3(\bar{u}) = 0.25 \times (1+u)(1+v)(1-v)(1-w)$$

$$N_4(\bar{u}) = 0.125 \times (1+u)(1+v)(1-w)(u+v-w-2)$$

$$N_5(\bar{u}) = 0.25 \times (1-u)(1+u)(1+v)(1-w)$$

$$N_6(\bar{u}) = 0.125 \times (1-u)(1+v)(1-w)(-u+v-w-2)$$

$$N_7(\bar{u}) = 0.25 \times (1-u)(1+v)(1-v)(1-w)$$

$$N_8(\bar{u}) = 0.25 \times (1-u)(1-v)(1+w)(1-w)$$

$$N_9(\bar{u}) = 0.25 \times (1+u)(1-v)(1+w)(1-w)$$

$$N_{10}(\bar{u}) = 0.25 \times (1+u)(1+v)(1+w)(1-w)$$

$$N_{11}(\bar{u}) = 0.25 \times (1-u)(1+v)(1+w)(1-w)$$

$$N_{12}(\bar{u}) = 0.125 \times (1-u)(1-v)(1+w)(-u-v+w-2)$$

$$N_{13}(\bar{u}) = 0.25 \times (1-u)(1+u)(1-v)(1+w)$$

$$N_{14}(\bar{u}) = 0.125 \times (1+u)(1-v)(1+w)(u-v+w-2)$$

$$N_{15}(\bar{u}) = 0.25 \times (1+u)(1+v)(1-v)(1+w)$$

$$N_{16}(\bar{u}) = 0.125 \times (1+u)(1+v)(1+w)(u+v+w-2)$$

$$N_{17}(\bar{u}) = 0.25 \times (1-u)(1+u)(1+v)(1+w)$$

$$N_{18}(\bar{u}) = 0.125 \times (1-u)(1+v)(1+w)(-u+v+w-2)$$

$$N_{19}(\bar{u}) = 0.25 \times (1-u)(1+v)(1-v)(1+w)$$

Table 2.3: Interpolation function values for a parabolic brick cell

Location	Face center0	Face center1	Face center2	Face center3	Face center4	Face center5	Volume center
(u,v,w)	(0,0,1)	(0,0,-1)	(0,1,0)	(0,-1,0)	(1,0,0)	(-1,0,0)	(0,0,0)
N_0	0	$-\frac{1}{4}$	0	$-\frac{1}{4}$	0	$-\frac{1}{4}$	$-\frac{1}{4}$
N_1	0	$\frac{1}{2}$	0	$\frac{1}{2}$	0	0	$\frac{1}{4}$
N_2	0	$-\frac{1}{4}$	0	$-\frac{1}{4}$	$-\frac{1}{4}$	0	$-\frac{1}{4}$
N_3	0	$\frac{1}{2}$	0	0	$\frac{1}{2}$	0	$\frac{1}{4}$
N_4	0	$-\frac{1}{4}$	$-\frac{1}{4}$	0	$-\frac{1}{4}$	0	$-\frac{1}{4}$
N_5	0	$\frac{1}{2}$	$\frac{1}{2}$	0	0	0	$\frac{1}{4}$
N_6	0	$-\frac{1}{4}$	$-\frac{1}{4}$	0	0	$-\frac{1}{4}$	$-\frac{1}{4}$
N_7	0	$\frac{1}{2}$	0	0	0	$\frac{1}{2}$	$\frac{1}{4}$
N_8	0	0	0	$\frac{1}{2}$	0	$\frac{1}{2}$	$\frac{1}{4}$
N_9	0	0	0	$\frac{1}{2}$	$\frac{1}{2}$	0	$\frac{1}{4}$
N_{10}	0	0	$\frac{1}{2}$	0	$\frac{1}{2}$	0	$\frac{1}{4}$
N_{11}	0	0	$\frac{1}{2}$	0	0	$\frac{1}{2}$	$\frac{1}{4}$
N_{12}	$-\frac{1}{4}$	0	0	$-\frac{1}{4}$	0	$-\frac{1}{4}$	$-\frac{1}{4}$
N_{13}	$\frac{1}{2}$	0	0	$\frac{1}{2}$	0	0	$\frac{1}{4}$
N_{14}	$-\frac{1}{4}$	0	0	$-\frac{1}{4}$	$-\frac{1}{4}$	0	$-\frac{1}{4}$
N_{15}	$\frac{1}{2}$	0	0	0	$\frac{1}{2}$	0	$\frac{1}{4}$
N_{16}	$-\frac{1}{4}$	0	$-\frac{1}{4}$	0	$-\frac{1}{4}$	0	$-\frac{1}{4}$
N_{17}	$\frac{1}{2}$	0	$\frac{1}{2}$	0	0	0	$\frac{1}{4}$
N_{18}	$-\frac{1}{4}$	0	$-\frac{1}{4}$	0	0	$-\frac{1}{4}$	$-\frac{1}{4}$
N_{19}	$\frac{1}{2}$	0	0	0	0	$\frac{1}{2}$	$\frac{1}{4}$

2.6.3 Linear wedge cell

Local coordinate system

In a linear wedge cell, each node point, $\bar{X}_i(x_i, y_i, z_i)$, $i = 0, 1, 2, 3, 4, 5$, is mapped to the local coordinate system (see Figure 2.9).

Interpolation function

In a linear wedge cell, a function $N_0(\bar{u})$ can be defined as

$$N_0(\bar{u}) = -\frac{(1-w)(3u + \sqrt{3}v - 1)}{6}.$$

by considering two planes:

- $f_0(\bar{u}) = 1 - w = 0$, which is a plane through node points 3, 4, and 5
- $f_1(\bar{u}) = 3u + \sqrt{3}v - 1 = 0$, which is a plane through node points 1, 2, 5, and 4.

Table 2.4: Interpolation function values for a linear wedge cell

Location	Face center0	Face center1	Face center2	Volume center
(u,v,w)	$(0, \frac{\sqrt{3}}{6}, 0)$	$(\frac{1}{4}, \frac{\sqrt{3}}{6}, 0)$	$(-\frac{1}{4}, \frac{\sqrt{3}}{12}, 0)$	$(0, 0, 0)$
N_0	$\frac{1}{4}$	0	$\frac{1}{4}$	$\frac{1}{6}$
N_1	$\frac{1}{4}$	$\frac{1}{4}$	0	$\frac{1}{6}$
N_2	0	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{6}$
N_3	$\frac{1}{4}$	0	$\frac{1}{4}$	$\frac{1}{6}$
N_4	$\frac{1}{4}$	$\frac{1}{4}$	0	$\frac{1}{6}$
N_5	0	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{6}$

Generally, the interpolation functions, $N_k(\bar{u})$ ($k=0, 1, 2, 3, 4, 5$), can be described as

$$\begin{aligned}
N_0(\bar{u}) &= -\frac{(1-w)(3u+\sqrt{3}v-1)}{6} \\
N_1(\bar{u}) &= -\frac{(1-w)(-3u+\sqrt{3}v-1)}{6} \\
N_2(\bar{u}) &= -\frac{(1-w)(2\sqrt{3}v+1)}{6} \\
N_3(\bar{u}) &= -\frac{(1+w)(3u+\sqrt{3}v-1)}{6} \\
N_4(\bar{u}) &= -\frac{(1+w)(-3u+\sqrt{3}v-1)}{6} \\
N_5(\bar{u}) &= -\frac{(1+w)(2\sqrt{3}v+1)}{6}
\end{aligned}$$

2.6.4 Parabolic wedge cell

Local coordinate system

In a parabolic wedge cell, each node point, $\bar{X}_i(x_i, y_i, z_i)$, $i=0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14$ is mapped to the local coordinate system (see Figure 2.10).

2.6.5 Interpolation function

In a parabolic wedge cell, a function $N_0(\bar{u})$ can be defined as

$$N_0(\bar{u}) = \frac{(6u+2\sqrt{3}v+3w+4)(3u+\sqrt{3}v-1)(1-w)}{18}$$

by considering three planes:

- $f_0(\bar{u}) = -(6u+2\sqrt{3}v+3w+4)$, which is a plane through node points 1, 6, and 5

- $f_1(\bar{u}) = (3u + \sqrt{3}v - 1)$, which is a plane through node points 2, 3, 4, 8, 13, 12, 11, and 7
- $f_2(\bar{u}) = 1 - w = 0$, which is a plane through node points 9, 10, 11, 12, 13, and 14.

Generally, the interpolation functions, $N_k(\bar{u})$ ($k=0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14$), can be described as

$$\begin{aligned}
N_0(\bar{u}) &= \frac{(6u + 2\sqrt{3}v + 3w + 4)(3u + \sqrt{3}v - 1)(1 - w)}{18} \\
N_1(\bar{u}) &= \frac{(-6u + 2\sqrt{3}v - 2)(3u + \sqrt{3}v - 1)(1 - w)}{9} \\
N_2(\bar{u}) &= \frac{(6u - 2\sqrt{3}v - 3w - 4)(3u - \sqrt{3}v + 1)(1 - w)}{18} \\
N_3(\bar{u}) &= \frac{(6u - 2\sqrt{3}v + 2)(2\sqrt{3}v + 1)(1 - w)}{9} \\
N_4(\bar{u}) &= \frac{(4\sqrt{3}v - 3w - 4)(2\sqrt{3}v + 1)(1 - w)}{18} \\
N_5(\bar{u}) &= \frac{(-6u - 2\sqrt{3}v + 2)(2\sqrt{3}v + 1)(1 - w)}{9} \\
N_6(\bar{u}) &= \frac{(-3u - \sqrt{3}v + 1)(1 + w)(1 - w)}{3} \\
N_7(\bar{u}) &= \frac{(3u - \sqrt{3}v + 1)(1 + w)(1 - w)}{3} \\
N_8(\bar{u}) &= \frac{(2\sqrt{3}v + 1)(1 + w)(1 - w)}{3} \\
N_9(\bar{u}) &= \frac{(6u + 2\sqrt{3}v - 3w + 4)(3u + \sqrt{3}v - 1)(1 + w)}{18} \\
N_{10}(\bar{u}) &= \frac{(-6u + 2\sqrt{3}v - 2)(3u + \sqrt{3}v - 1)(1 + w)}{9} \\
N_{11}(\bar{u}) &= \frac{(6u - 2\sqrt{3}v + 3w - 4)(3u - \sqrt{3}v + 1)(1 + w)}{18} \\
N_{12}(\bar{u}) &= \frac{(6u - 2\sqrt{3}v + 2)(2\sqrt{3}v + 1)(1 + w)}{9} \\
N_{13}(\bar{u}) &= \frac{(4\sqrt{3}v + 3w - 4)(2\sqrt{3}v + 1)(1 + w)}{18} \\
N_{14}(\bar{u}) &= \frac{(-6u - 2\sqrt{3}v + 2)(2\sqrt{3}v + 1)(1 + w)}{18}
\end{aligned}$$

2.6.6 Linear tetrahedral cell

Local coordinate system

In a linear tetrahedral cell, each node point, $\bar{X}_i = (x_i, y_i, z_i)$, $i = 0, 1, 2, 3$, is mapped to the local coordinate system (see Figure 2.11).

Table 2.5: Interpolation function values for a parabolic wedge cell

Location	Face center0	Face center1	Face center2	Face center3	Face center4	Volume center
(u,v,w)	(0,0,1)	(0,0,-1)	(0,- $\frac{\sqrt{3}}{6}$,0)	($\frac{1}{4},\frac{\sqrt{3}}{6}$,0)	(- $\frac{1}{4},\frac{\sqrt{3}}{12}$,0)	(0,0,0)
N_0	0	$-\frac{1}{9}$	$-\frac{1}{4}$	0	$-\frac{1}{4}$	$-\frac{2}{9}$
N_1	0	$\frac{4}{9}$	$\frac{1}{2}$	0	0	$-\frac{1}{9}$
N_2	0	$-\frac{1}{9}$	$\frac{1}{4}$	$\frac{1}{4}$	0	$-\frac{1}{9}$
N_3	0	$\frac{4}{9}$	0	$\frac{1}{2}$	0	$-\frac{1}{9}$
N_4	0	$-\frac{1}{9}$	0	$-\frac{1}{4}$	$-\frac{1}{4}$	$-\frac{1}{9}$
N_5	0	$\frac{4}{9}$	0	0	$\frac{1}{2}$	$-\frac{1}{9}$
N_6	0	0	$\frac{1}{2}$	0	$\frac{1}{2}$	$-\frac{1}{9}$
N_7	0	0	$\frac{1}{2}$	$\frac{1}{2}$	0	$-\frac{1}{9}$
N_8	0	0	0	$\frac{1}{2}$	$\frac{1}{2}$	$-\frac{1}{9}$
N_9	$-\frac{1}{9}$	0	$-\frac{1}{4}$	0	$-\frac{1}{4}$	$-\frac{1}{9}$
N_{10}	$-\frac{1}{9}$	0	$\frac{1}{2}$	0	0	$-\frac{1}{9}$
N_{11}	$-\frac{1}{9}$	0	$-\frac{1}{4}$	$-\frac{1}{4}$	0	$-\frac{1}{9}$
N_{12}	$-\frac{1}{9}$	0	0	$\frac{1}{2}$	0	$-\frac{1}{9}$
N_{13}	$-\frac{1}{9}$	0	0	$-\frac{1}{4}$	$-\frac{1}{4}$	$-\frac{1}{9}$
N_{14}	$\frac{1}{9}$	0	0	0	$\frac{1}{2}$	$-\frac{1}{9}$

Interpolation function

In a linear tetrahedral cell, a function $N_0(\vec{u})$ can be defined as

$$N_0(\vec{u}) = u$$

by considering a plane

- $f_0(\vec{u}) = u = 0$, which is a plane through node points 1, 2, and 3.

Generally, the interpolation functions, $N_k(\vec{u})$ ($k=0, 1, 2, 3$), can be described as:

$$N_0(\vec{u}) = u$$

$$N_1(\vec{u}) = v$$

$$N_2(\vec{u}) = w$$

$$N_3(\vec{u}) = z = 1 - u - v - w$$

2.6.7 Parabolic tetrahedral cell

Local coordinate system

In a parabolic tetrahedral cell, each node point, $\bar{X}_i = (x_i, y_i, z_i)$, $i = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9$, is mapped to the local coordinate system (see Figure 2.12).

Interpolation function

In a parabolic tetrahedral cell, a function $N_0(\bar{u})$ can be defined as

$$N_0(\bar{u}) = u(2u - 1)$$

by considering two planes:

- $f_0(\bar{u}) = u = 0$, which is a plane through node points 2, 3, 4, 8, 9, and 7
- $f_1(\bar{u}) = 2u - 1 = 0$, which is a plane through node points 1, 5, and 6.

Generally, the interpolation functions, $N_k(\bar{u})$ ($k=0, 1, 2, 3, 4, 5, 6, 7, 8, 9$), can be described as

$$N_0(\bar{u}) = u(2u - 1)$$

$$N_1(\bar{u}) = 4uv$$

$$N_2(\bar{u}) = v(2v - 1)$$

$$N_3(\bar{u}) = 4vw$$

$$N_4(\bar{u}) = w(2w - 1)$$

$$N_5(\bar{u}) = 4wu$$

$$N_6(\bar{u}) = 4zu$$

$$N_7(\bar{u}) = 4vz$$

$$N_8(\bar{u}) = 4wz$$

$$N_9(\bar{u}) = z(2z - 1)$$

Table 2.6: Interpolation function values for a parabolic tetrahedral cell

Location	Face center0	Face center1	Face center2	Face center3	Volume center
(u,v,w,z)	$(0, \frac{1}{3}, \frac{1}{3}, \frac{1}{3})$	$(\frac{1}{3}, 0, \frac{1}{3}, \frac{1}{3})$	$(\frac{1}{3}, \frac{1}{3}, 0, \frac{1}{3})$	$(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}, 0)$	$(\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4})$
N_0	0	$-\frac{1}{9}$	$-\frac{1}{9}$	$-\frac{1}{9}$	$-\frac{1}{4}$
N_1	0	0	$\frac{4}{9}$	$\frac{1}{9}$	$\frac{1}{4}$
N_2	$-\frac{1}{9}$	0	$-\frac{1}{9}$	$-\frac{1}{9}$	$-\frac{1}{4}$
N_3	$\frac{4}{9}$	0	0	$\frac{1}{9}$	$\frac{1}{4}$
N_4	$-\frac{1}{9}$	$-\frac{1}{9}$	0	$-\frac{1}{9}$	$-\frac{1}{4}$
N_5	0	$\frac{4}{9}$	0	$\frac{1}{9}$	$\frac{1}{4}$
N_6	0	$-\frac{1}{9}$	$\frac{4}{9}$	$-\frac{1}{9}$	$-\frac{1}{4}$
N_7	$\frac{4}{9}$	0	$-\frac{1}{9}$	0	$\frac{1}{4}$
N_8	$-\frac{1}{9}$	$\frac{4}{9}$	0	0	$-\frac{1}{4}$
N_9	$-\frac{1}{9}$	$-\frac{1}{9}$	$-\frac{1}{9}$	0	$-\frac{1}{4}$

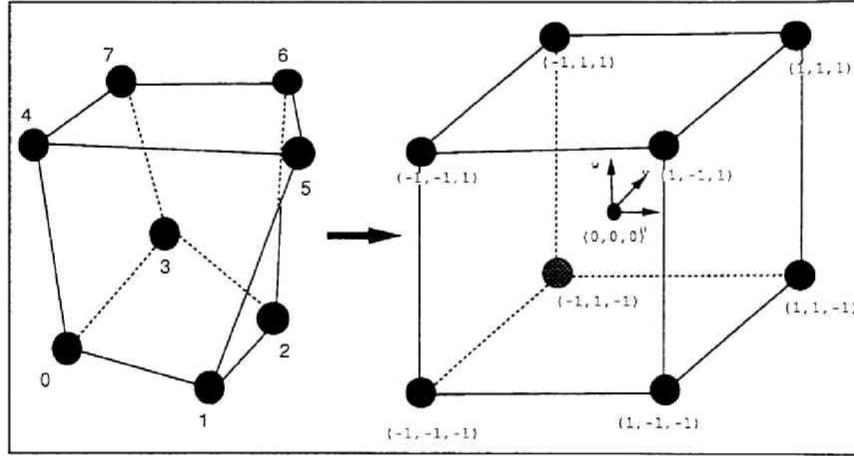


Figure 2.7: Linear brick cell

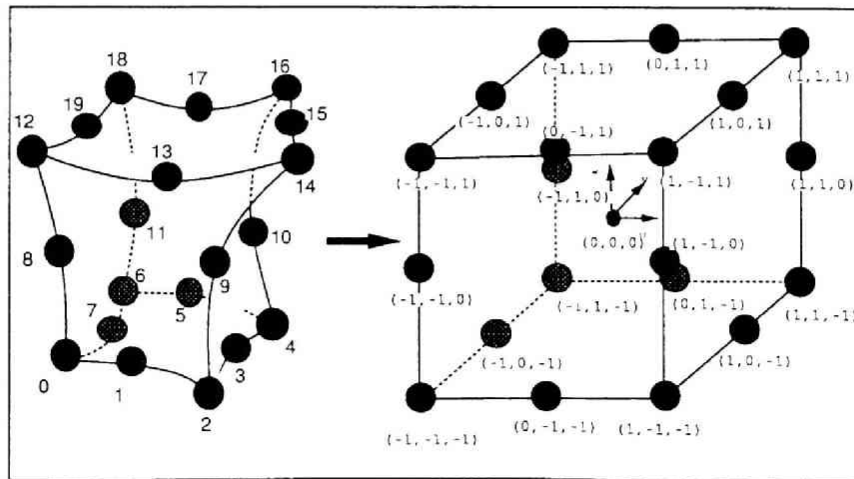


Figure 2.8: Parabolic brick cell

2.6. APPENDIX

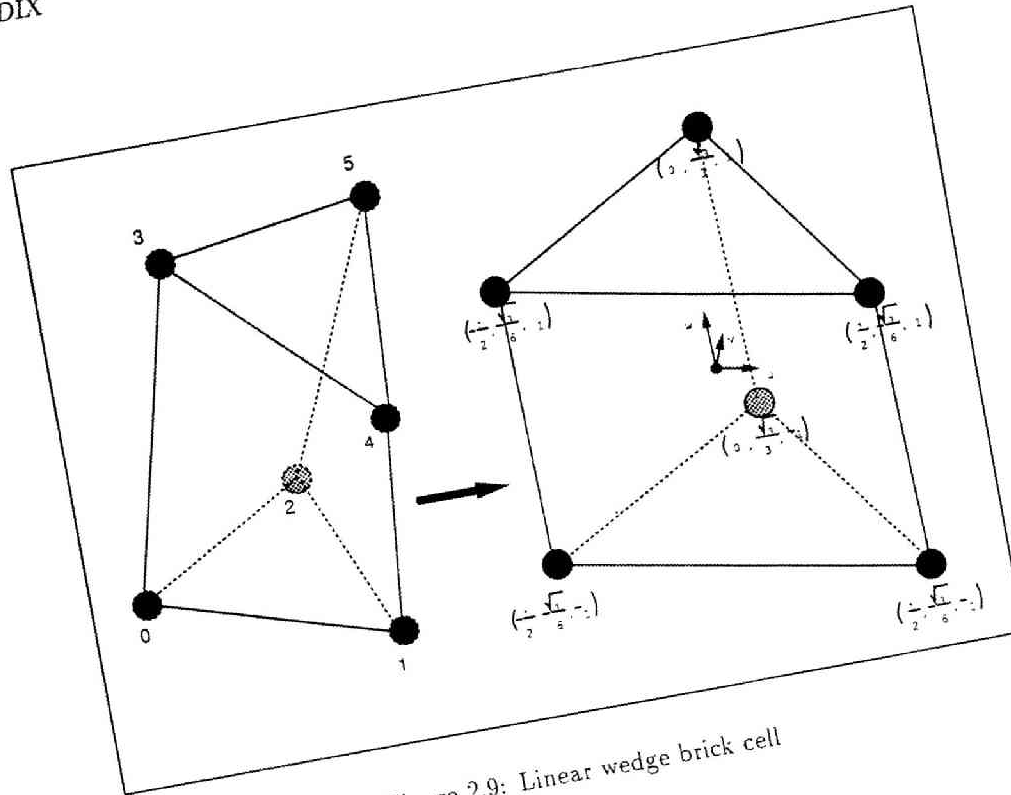


Figure 2.9: Linear wedge brick cell

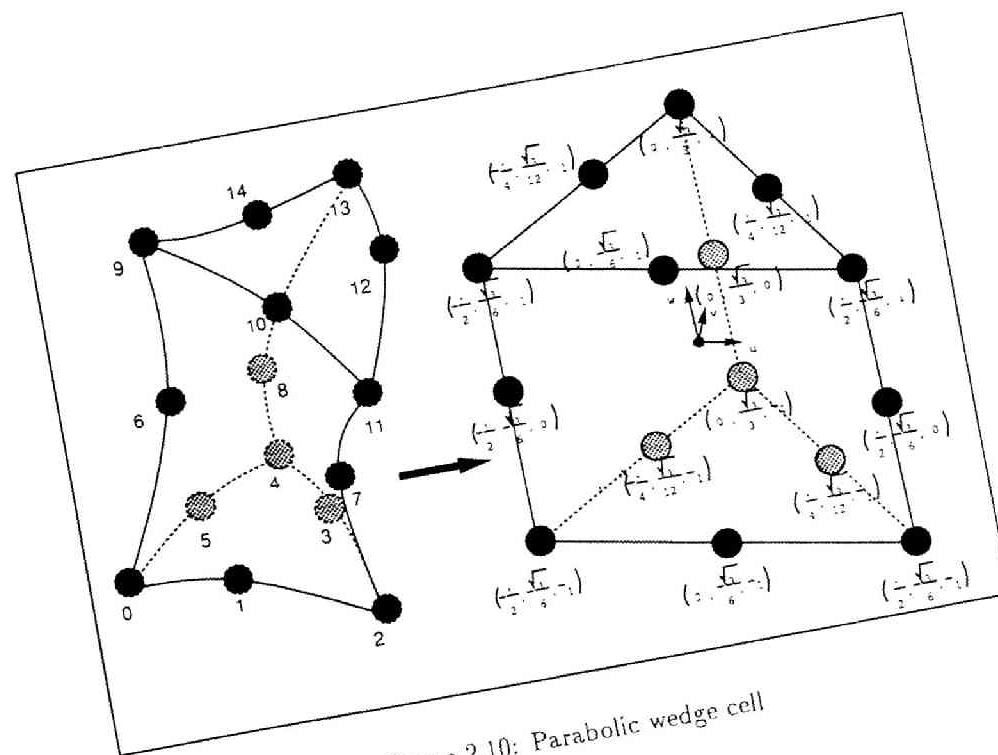


Figure 2.10: Parabolic wedge cell

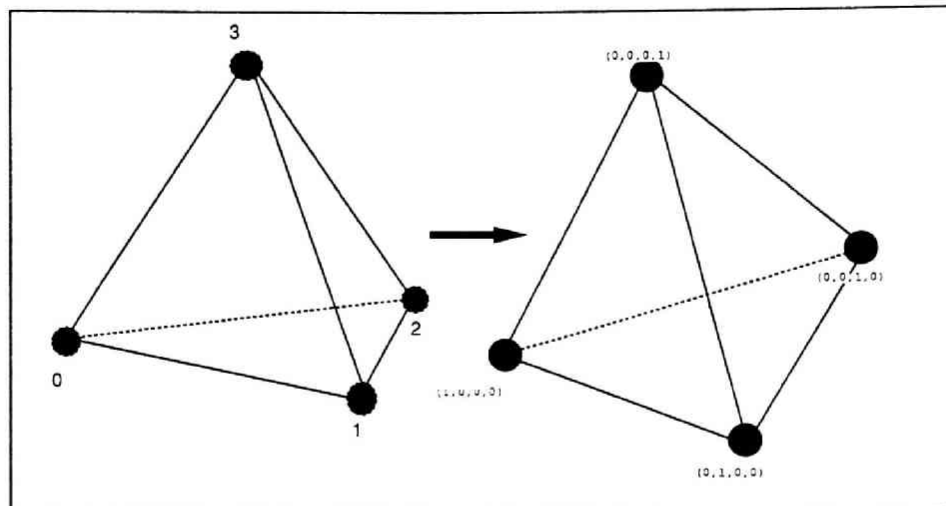


Figure 2.11: Linear tetrahedral cell

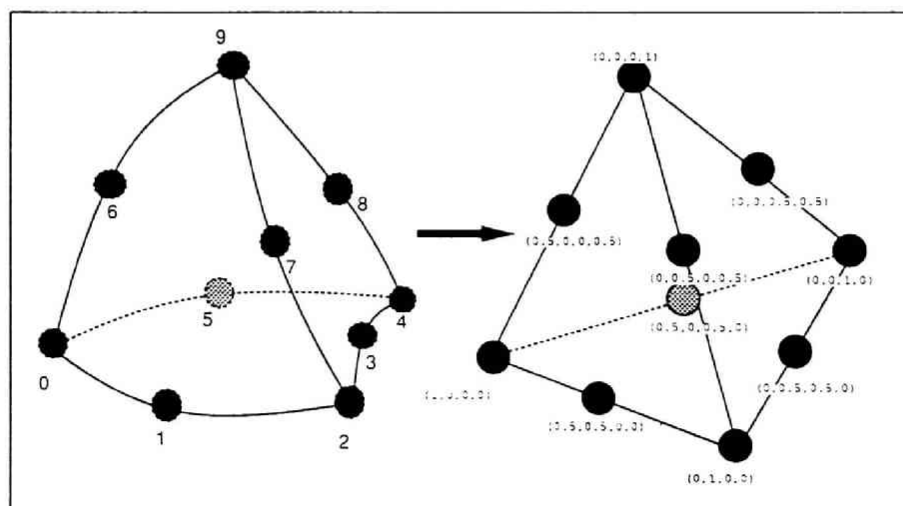


Figure 2.12: Parabolic tetrahedral cell

Chapter 3

Isosurface display technique

An isosurface is defined as a set of points that satisfy the equation

$$S(\vec{X}) - C = 0, \quad (3.1)$$

where $S(\vec{X})$ is a scalar function on a given volume dataset and C is a constant scalar value. A criterion surface, such as, an allowable temperature or stress surface, can be displayed by using isosurface display techniques. Such a criterion surface helps users to make engineering decisions. Indeed, displaying an isosurface statically is useful for showing the whole distribution of the scalar data, but the surface constant C must be changed in real time in an interactive visualization system. To realize interactivity, it is highly desirable to march (traverse) only those cells that have points of intersection with the isosurface. because in general the number of cells that actually intersect with the isosurface is only a small fraction of the total number of tetrahedral cells. It can be estimated as $O(N^{2/3})$, where N is the total number of cells.

To show the data distribution more specifically, a volume slice display is often employed. Since a volume slice is a kind of isosurface, most techniques for generating an isosurface can be shared. To display a volume slice, a slice function that describes the slice surface (for example a linear combination of coordinate values) is specified in advance. After the volume slice data have been generated as an isosurface of the function, the originally defined scalar data are mapped onto the slice by applying an interpolation technique. To improve the efficiency of the technique in an interactive environment, the same effort as for an isosurface is required in order to avoid an exhaustive search for intersected cells.

The purpose of this chapter is to present an efficient technique for generating isosurfaces of irregular volumes by polygonal approximation. Our technique can be categorized as “marching tetrahedra.”

However, unlike other marching tetrahedra techniques, it is fully table-driven and employs an efficient traverse technique based on cell adjacency information. In this chapter, we first give an overview of the marching tetrahedra technique. We then describe our table-driven technique for generating polygonal data within a tetrahedral cell and propose an extrema-based marching tetrahedra technique. Finally, the proposed technique is evaluated on the basis of the time cost.

3.1 Overview of marching tetrahedra

The marching tetrahedra algorithm consists of two procedures. The first is to generate triangular facets within a tetrahedral cell, and the second is to march (traverse) the next cell in order to check whether the cell intersects the surface.

To generate the triangular facets within a tetrahedral cell, a difference $g_i = S(\bar{X}_i) - C$ is first calculated at each node point $\bar{X}_i$. Based on the signs of the values, an integer (index) value is assigned to the tetrahedral cell¹ (see Figure 3.1). The index value is then used as a pointer to a pre-computed table.

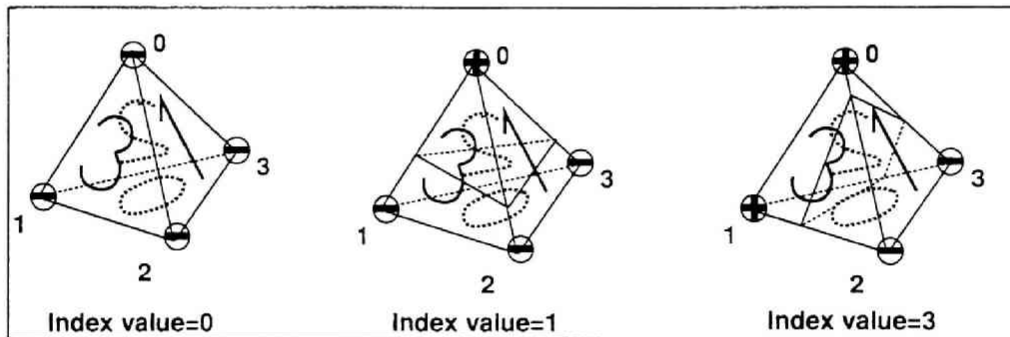


Figure 3.1: Signs given at node points in a tetrahedral cell

which describes intersected edges as combinations of edge-vertex (node) identifiers, so that they can form triangular facets (see Table 3.1). Note that the facets can be defined in a single tetrahedral cell without the ambiguity that a marching cube technique suffers from [Durs88], because the data gradient is constant and the local isosurface becomes planar. After intersected edges have been returned, the surface intersection points are calculated along the edges by linear interpolation. Generally, an edge is shared by several cells. To give a unique identifier to each intersection and avoid duplicate calculations on the edge, hashing is introduced.

¹ Because scalar data values are linearly distributed along an edge, the checking can be done by comparing the signs of the difference values at the end points.

Table 3.1: Pre-computed intersection table (P:+,N:-)

Index value	Combination of signs	Intersected edges	Intersected faces
0	(NNNN)	None	None
1	(PNNN)	(0,1)(0,2)(0,3)	1,2,3
2	(NPNN)	(0,1)(1,2)(1,3)	0,2,3
3	(PPNN)	(0,2)(0,3)(1,2)(1,3)	0,1,2,3
...		...	
15	(PPPP)	None	None

To maximize the performance, it is important to traverse only tetrahedral cells whose edges intersect the isosurface, because not all tetrahedral cells intersect the isosurface. In our table, a set of intersected cell faces is also registered (see Table 3.1). By using the information, our algorithm can selectively handle those tetrahedral cells that are connected to intersected faces.

3.2 Triangular facet generation within a cell

3.2.1 Related work

To construct triangular facets in a tetrahedral cell, Giles devised a four-bit index for the cell by setting the k -th bit to "0" or "1," depending on whether the value at the k -th node is less than the surface value [Gile90]. This index, which can be any integer from 0 to 15, is used as a pointer to a lookup table, as we have already mentioned. If the value at a node point equals the surface value, constructed triangular facets may be degenerated. This degeneration must be handled specially to save the memory space for the triangular facets.

To cope with the case in which the value equals the surface value, Koide et al. consider three states for a node point: "negative," "zero," and "positive" [Koid86]. First, the numbers of node points having negative values and zeros within the cell are counted and the node identifiers are rearranged in the order negative, zero, and positive; that is to say, the first N_m are the points having negative values, the next N_z are points having a zero value, and the remainder are points having positive values. After the rearrangement, a matching table for the node identifiers is constructed. Next, an index value for a lookup table is calculated by using N_m and N_z . Although the rearrangement reduces the size of the lookup table to 15 entries, two additional processes are required:

- Restore the original node identifiers by using the matching table.

Table 3.2: Intersection table based on three states (-:N,0:Z,+:P)

Index value	Combination of signs	Intersected edges	Intersected faces
0	(NNNN)	None	None
1	(ZNNN)	None	None
2	(PNNN)	(0,1)(0,2)(0,3)	1,2,3
3	(NZNN)	None	None
4	(ZZNN)	None	None
5	(PZNN)	(1,1)(0,2)(0,3)	1,2,3
6	(NPNN)	(0,1)(1,2)(1,3)	0,2,3
7	(ZPNN)	(0,0)(1,2)(1,3)	0,2,3
8	(PPNN)	(0,2)(0,3)(1,2)(1,3)	0,1,2,3
...		...	
81	(PPPP)	None	None

- Exchange the order of the formed triangular facets if the rearranged node points yield a negative orientation for the tetrahedron.

3.2.2 Fully table-driven approach

To efficiently generate triangular facets on the assumption that a node point may have one of three states (negative, zero, or positive), our approach employs a larger lookup table. The size is 81, which is the fourth power of 3. The table contains pairs of node identifiers for triangular facets to be generated, and the number of internal faces that intersect the surface (see Table 3.2).

Obviously, the required memory space in our approach is larger than in previous approaches, because of the table size, but this drawback is trivial. Our approach is more efficient, because it requires no post-processing and is fully table-driven. The procedure is as follows:

- Set the index value to zero.
- Compare the data values at four node points with the surface value. If the k -th value equals the surface value, the k -th power of three is added to the index value. If the k -th value is more than the surface value, twice the k -th power of three is added to the index value.
- Return the vertices of triangular facets as pairs of node point identifiers, and the number of intersected faces.

The lookup table for our approach could be constructed manually, but it has been obtained by giving Giles's algorithm 81 cases of data values, where a data value is changed in the order negative, zero, and

positive at each node point.

3.3 Marching to the next cell

3.3.1 Related work

To reduce the number of volume cells searched for isosurfaces, Gallagher proposed a technique for grouping volume cells that have consecutive identifiers within the same span tolerance [Gall91]. In the technique, the possible range of scalar data values in the volume data is divided up into a number of ranges, or "buckets." These ranges can be linear subdivisions of the range. After buckets have been determined, all volume cells are examined in order of their cell identifiers. During the examination, the cell's node values are evaluated to determine the span of their scalar values and the number of span buckets. If the volume cell shares the same lower bound bucket and bucket span as the current group, this cell is added to the group. If not, it is used as the base address of a new group. As a result, the technique produces a compressed data structure containing a list of volume cells that include a given isosurface value C . However, this technique is less useful if a high percentage of cells have large spans of values and little compression of consecutive identifiers can be realized.

Haimes developed a technique for determining the cells that are crossed by an isosurface with active cell lists [Haim91a]. The active cell lists are constructed from two lists ordered according to cells' minimum and maximum values, based on a given isosurface value C . Although the active list contains only cells that actually intersect the isosurface, its reconstruction for interactive use and the sorting process for the pre-process may take a considerable time when the number of volume cells is large and the maximum value of the cells' spans is smaller than the difference in the value C .

Speray proposed a technique for constructing a volume slice by using cell adjacency information [Sper90]. Given an intersected cell, the looking table uses that cell's contents to list the cell faces that should be moved through, and intersected cells are found by a breadth-first traversal of an adjacency graph in which the nodes are the volume cells and the arcs are their connectivities across cell faces. To search for an intersected cell (hot spot) in the first place, the technique uses a line-of-sight face-hopping approach from a known cell. When the hot spot is driven out of sight, an alternative hot spot must be defined manually.

Strid developed a similar technique by searching for all exterior cells that are crossed by a volume

slice and then constructing the volume slice from the crossed exterior cells by using cell connectivity information to check neighboring cells to determine whether they are crossed [Stri89]. This technique works well for a planar slice, but not for isosurfaces that do not cross the boundaries of the volume data.

3.3.2 Extremum-based marching tetrahedra

Our technique efficiently searches for a tetrahedral cell that intersects an isosurface, starting from an extremum point in the volume data. It does not depend on the span size of the cell and, unlike Speray's technique, it automatically searches for intersected cells. It is common for an initial point to be located in the vicinity of an extremum point when a non-linear equation is solved on the basis of Newton iteration. Our technique is inspired by this. This technique, extremum-based marching tetrahedra, consists of three procedures (see Figure 3.2):

- Find extremum points in the volume data.
- Search for seed cells from the extremum points.
- Propagate intersected cells from the seed cells.

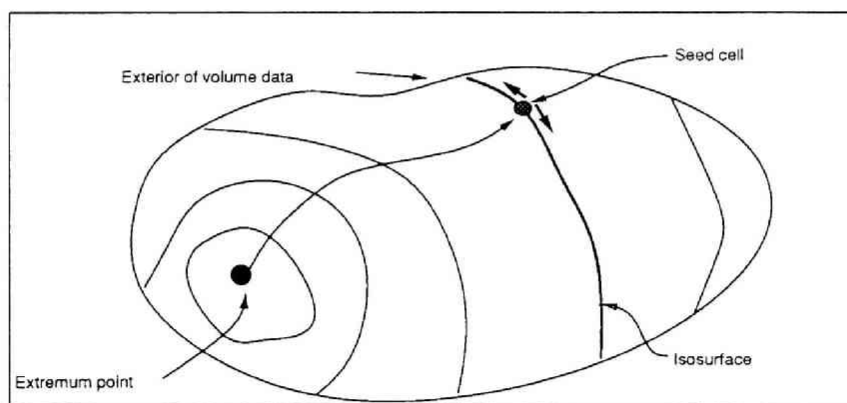


Figure 3.2: Overview of extremum-based marching tetrahedra

Our technique can cope with the case in which an isosurface is composed of a multiple set of triangular facets, because generally there are the related extremum points in the volume data.

Finding extremum points

An extrema point is defined as a point at which a data value becomes minimum or maximum locally. If a volume dataset is described in the tetrahedral model, the extrema point is one of the tetrahedral cell's node points, because the data distribution is linear within the cell.

With this result, extrema points could be searched for by preparing a list of the node points that are connected to a node point, and evaluating the differences in the data values there. This technique requires a large memory space for storing the list and considerable time for constructing it. Instead, our technique evaluates the differences at each cell without constructing such a list, and marks each node point with the type of extremum, minimum or maximum. Since a node point is generally shared by several tetrahedral cells, marked node points are accessed more than once. If a different type is stored in advance, it is marked "no extremum." If the type is identical, the gradient magnitude is compared with that stored in advance, and the identifier of the cell that has the stronger gradient is recorded.

When a volume slice is displayed, an extremum is considered not for the scalar data originally defined but for slice function data. Since extremum points exist on the boundary faces in this case, the finding procedures can be performed at each exterior face, for efficiency.

After the identifiers of cells that contain extremum points have been determined, two ordered lists of the cells are formed, EL_{max} for maximum extrema and EL_{min} for minimum extrema, which are ordered according to the extremum values. When the constant C is specified, the active cell list EL_{active} is formed by taking the section of list EL_{min} with cells whose extremum values are less than C and by taking the section of list EL_{max} with cells whose extremum values are larger than C .

Searching for seed cells

A seed cell is a source point from which a cell traverse is made, and is defined as any of the cells that intersect a specified surface (isosurface or volume slice). Our technique searches for seed cells by traversing cells in accordance with the cell adjacency information, starting from a cell within the active extremum list. It is desirable that a scalar gradient vector should be referred to for the traversal. Except for the entry face, there are three faces that the traversal can proceed to in a tetrahedral cell. It is possible to select a suitable face by using the gradient vector field.

Instead, our technique employs a more efficient approach, because it may take a considerable time to

calculate a gradient vector at each tetrahedral cell. The technique first compares the four data values in a tetrahedral cell and then selects an exit face that is opposite the point with the maximum or minimum value if the searching starts from a minimum or maximum extremum point, respectively (see Figure 3.3). This gives a good approximation to the above traversal, because a vector directed from the center of

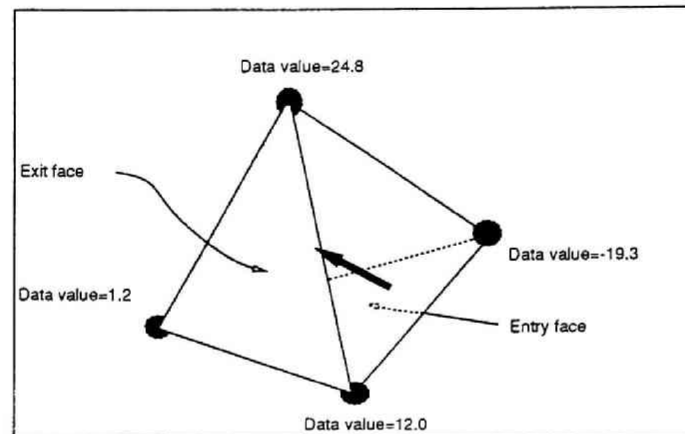


Figure 3.3: Searching from a minimum extremum point

gravity of a tetrahedral cell to the center of a face opposite the point with the maximum value makes an acute angle with the scalar gradient vector (see the appendix for the proof).

When a volume slice is constructed, seed cells are searched for along exterior faces. In this case, our technique compares the three data values in a triangle of the exterior face and selects as its exit edge the one that is opposite to the point with the maximum value.

To eliminate redundant searching, our technique marks each cell in a list with the surface number if that cell has been visited. This list needs to be reset only when the slice number wraps around, and can be commonly used in the next procedure, which is propagating.

Propagating from seed cells

Once a seed cell has been found, its lookup table entry lists the faces that should be propagated through to generate the isosurface. This propagating procedure generates proliferously from the seed cell triangular facets that approximate the isosurface. This procedure is a breadth-first traversal of the subgraph of $G(V,A)$ in which the nodes² are intersected cells and the arcs are intersected faces (see Figure 3.4). The implementation using an FIFO queue is as follows:

²In the rest of this paper, we use the word “node” to describe the vertices of tetrahedral cells.

1. Register the identifier of a seed cell in the FIFO queue.
2. Take the identifier of a tetrahedral cell from the FIFO queue. Quit if the queue is empty.
3. Calculate the index value from the difference values g_i at tetrahedral cell points and get information on the intersected edges and intersected faces.
4. Calculate the coordinates of the intersection, give a unique identifier, and register it in a hash table if the calculation has not previously been done. Otherwise, get the identifier for the intersection point from the hash table.
5. Output triangular facets as a set of identifiers.
6. Get the identifiers of neighboring tetrahedral cells connected through the intersected face.
7. Register the identifiers in the FIFO queue if they have not yet been registered there.
8. Return to step 2.

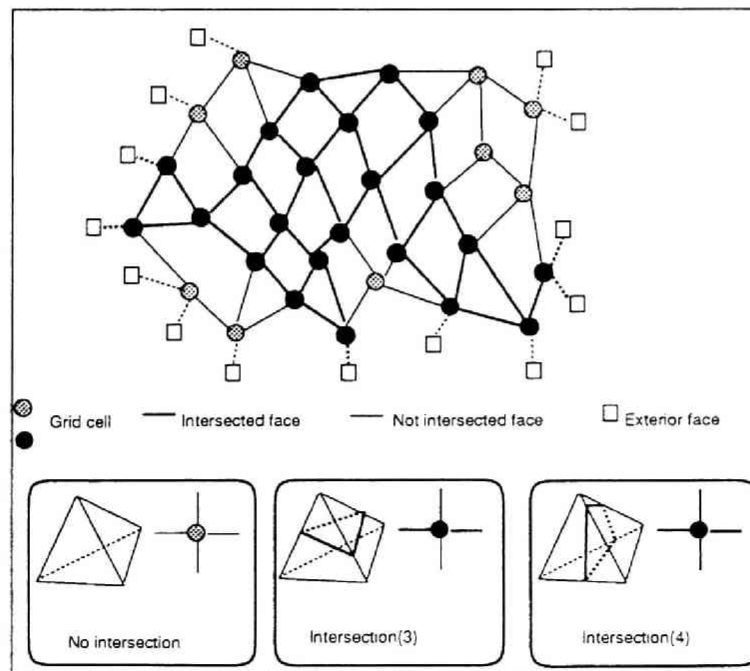
Figure 3.4: Traversal of the partial graph of $G(V.A)$

Table 3.3: Results of performance evaluation of facet generation techniques

Technique	<i>Giles's technique</i>	<i>Koide's technique</i>	<i>Our technique</i>
Time cost (msec)	1.052	1.137	0.868

3.4 Performance evaluation

3.4.1 Facet generation based on fully table-driven approach

To evaluate our facet generation technique in comparison with Giles's algorithm and Koide's algorithm, we measured the total time taken to generate triangular facets within a single tetrahedral cell for all combinations of signs at node points (81 cases). The measured CPU costs are shown in Table 3.3.

3.4.2 Extremum-based marching tetrahedra

Four irregular volume datasets have been selected to evaluate our extremum-based marching tetrahedra technique. These datasets are categorized as non-convex, because they involve cavities and holes. All calculations were performed on an IBM 3090 system. To demonstrate the efficiency of the algorithm, let us compare it with another technique, which has the following features:

- The procedure for generating triangular facets within a cell is the same.
- To march the next cell, a brute-force approach is employed, that is, tetrahedral cells are processed in the order in which they are stored.

The time costs, T_{extre} and T_{brute} , of generating fifty concentric spheres (iso-distance surfaces) by our technique and the brute-force technique respectively, were measured for the datasets: Table 3.4 contains the following statistics on the datasets the number of tetrahedral cells (N), the number of generated triangles (N_t), and the number of their vertices (N_v). T_{depth} is the time cost when a depth-first traversal is employed for the propagation procedure. It is shown that the breadth-first traversal is more efficient than the depth-first one. T_{seed} is the cost of searching for the seed cells from extremum points and may be negligible in the total time cost.

Here, we analyze our technique by introducing cost functions. By considering the algorithms of the techniques, we estimate the time cost functions T_{brute} and T_{extre} . The cost function T_{brute} can be divided

Table 3.4: Results of performance evaluation of cell traverse techniques

Dataset name	no. 1	no. 2	no. 3	no. 4
N	10167	61680	372500	557868
N_t	59412	253822	792111	866732
N_v	32170	131951	404710	439710
T_{brute}	4.36	22.02	128.30	155.86
T_{extre} (seconds)	2.41	10.08	25.54	33.28
Rate	1.81	2.18	5.02	4.68
T_{depth}	3.36	12.07	36.58	41.50
T_{seed}	0.044	0.280	1.090	1.824

into two parts: T_{check} and T_{int} .

$$T_{brute} = \underbrace{t_{check} \times N}_{T_{check}} + \underbrace{t_{int} \times N_v}_{T_{int}} \quad (3.2)$$

T_{check} is the total cost of checking whether a tetrahedral cell intersects with the isosurface. Therefore, it is proportional to the number of tetrahedral cells (N). T_{int} is the total cost of calculating intersection points in tetrahedral cells. Therefore, it is proportional to the number of intersection points (N_v). The cost function T_{extre} can be divided into two parts: T_{seed} and the rest $T_{rest} = T_{extre} - T_{seed}$.

$$T_{extre} = \underbrace{t_{seed} \times N}_{T_{seed}} + \underbrace{t_{rest} \times N_v}_{T_{rest}} \quad (3.3)$$

T_{seed} is proportional to the length of the search path to a seed cell, and is proportional to the number of extremum points. The length is $O(N^{1/3})$. The number of extremum points is related to the complexity of the exterior faces, because the extremum points appear on these faces. For simplicity, we assume that it is proportional to the number of the exterior faces, $O(N^{2/3})$. Therefore, T_{seed} is $O(N) = O(N^{1/3}) \times O(N^{2/3})$. The remainder of the cost function is the cost T_{FIFO} of handling the FIFO queue and the cost T_{int} of calculating intersection points at each triangle. T_{FIFO} is $O(N_t)$ and T_{int} is $O(N_v)$. Because N_v is proportional to N_t in many cases, both costs can be considered as $O(N_v)$ for simplicity. Therefore, T_{rest} is $O(N_v)$.

From the statistical information, these costs are estimated as follows:

- $t_{check} = 2.317 \times 10^{-4}$
- $t_{int} = 5.904 \times 10^{-5}$
- $t_{seed} = 3.765 \times 10^{-6}$

Table 3.5: Comparison with brute-force algorithm

Dataset name	no. 1	no. 2	no. 3	no. 4
T_{brute}	4.36	22.02	128.30	155.86
T_{check}	2.36	14.29	86.31	129.26
T_{int}	1.82	7.79	24.31	26.60
(seconds)				
T_{extre}	2.41	10.08	25.54	33.28
T_{seed}	0.04	0.28	1.09	1.82
T_{rest}	2.16	9.21	28.75	31.45
(seconds)				

- $t_{rest} = 6.994 \times 10^{-5}$

Substituting these costs into Eq. 3.2 and Eq. 3.3, we can calculate each component of the defined total cost functions $(T_{check}, T_{int}, T_{seed}, T_{rest})$, as shown in Table 3.5. The result means that the main factor in the total cost of the brute-force technique is the cost of searching for intersected cells, especially when the size of the dataset is large, and that the total cost of the extremum-based marching tetrahedra technique is proportional to the number of generated triangular facets if we neglect the cost of searching for the seed cells.

3.5 Summary of this chapter

We have proposed a high-performance technique for generating a series of isosurfaces. This technique first constructs extremum point lists and automatically searches for the seed cells from the lists. Once seed cells have been found, a breadth-first traverse of the adjacency graph efficiently gives a complete set of triangular facets to represent isosurfaces. We successfully applied the technique to generating isosurfaces, and confirmed its efficiency. In future, we will apply a flow topology analysis to a scalar gradient field of a given volume dataset and construct a list of seed cells based on the topological characteristics.

3.6 Appendix

3.6.1 Proof

Suppose without losing generality that S_0 is the maximum among four scalar values S_i ($i=0,1,2,3$). Let $\bar{X}_i$, $\bar{F}_i$, $\bar{G}$, and S_i , ($i=0,1,2,3$) be vectors for the i th node point, the i th face center, and the center of gravity, and a data value for the i th node point. The data value at a point within a tetrahedral cell can

be described by using a linear combination of the coordinates (x,y,z):

$$S = P \times x + Q \times y + R \times z + C. \quad (3.4)$$

The gradient vector $\nabla S = (P, Q, R)$ can be calculated by applying the above equation to each node point:

$$\begin{aligned} \begin{pmatrix} S_1 - S_0 \\ S_2 - S_0 \\ S_3 - S_0 \end{pmatrix} &= \begin{bmatrix} x_1 - x_0 & y_1 - y_0 & z_1 - z_0 \\ x_2 - x_0 & y_2 - y_0 & z_2 - z_0 \\ x_3 - x_0 & y_3 - y_0 & z_3 - z_0 \end{bmatrix} \times \begin{pmatrix} P \\ Q \\ R \end{pmatrix} \\ &= [M] \bullet \nabla S \end{aligned}$$

A vector directed from the center of gravity to the center of the i th face is described as $\vec{G}F_i$. The difference between the inner products $\vec{G}F_0 \bullet \nabla S$ and $\vec{G}F_1 \bullet \nabla S$ is evaluated as

$$\begin{aligned} \vec{G}F_0 \bullet \nabla S - \vec{G}F_1 \bullet \nabla S &= F_1 \vec{F}_0 \bullet \nabla S \\ &= \frac{1}{3}(\vec{X}_0 - \vec{X}_1) \bullet \nabla S \\ &= \frac{1}{3}(x_0 - x_1, y_0 - y_1, z_0 - z_1)[M]^{-1} \begin{pmatrix} S_1 - S_0 \\ S_2 - S_0 \\ S_3 - S_0 \end{pmatrix} \\ &= \frac{1}{3}(-1, 0, 0) \begin{pmatrix} S_1 - S_0 \\ S_2 - S_0 \\ S_3 - S_0 \end{pmatrix} \\ &= \frac{1}{3}(S_0 - S_1) > 0 \end{aligned}$$

Since the other cases ($i=2,3$) can be handled in the same way, $\vec{G}F_0 \bullet \nabla S > \vec{G}F_i \bullet \nabla S$ ($i=1,2,3$), which means that the inner product has the maximum value at the 0th face.

Incidentally, because the sum of four vectors $\vec{G}F_i$ becomes a zero vector,

$$\sum_{i=0}^3 \vec{G}F_i \bullet \nabla S = 0 \quad (3.5)$$

This means that the maximum value among the four inner products, $\vec{G}F_i \bullet \nabla S$, is positive, which proves the above rule. (End of proof)

Chapter 4

Cloudy representation technique

A technique that represents a given volume dataset by using clouds is generally called volume rendering. Since our perceptual system assumes that sensory input arises from physically plausible phenomena, and forms an interpretation of the input on that basis, it is desirable that volume data should be represented by physically plausible objects. For example, an air-borne particle density field in a clean room, which is calculated by FEM techniques, can be satisfactorily represented by the volume rendering technique, because this creates an image similar to what we see in our laboratory. In general, this technique can be used to show the overall features of a given volume dataset.

As we have noted in Chapter 1, these techniques can be classified into two approaches: the image order approach and the object order approach. We have developed techniques based on both approaches for irregular volume datasets. In each approach, a tetrahedral cell is selected as a visualization primitive, for reasons of stability and efficiency of computation.

The image order approach generally takes a lot of CPU time, although it can produce a high-quality image. The most computationally intensive portions of the image order approach are testing for intersections of a ray and an irregular volume, and integration of brightness along rays. In order to relax these two bottlenecks of computation, we have developed a new cell traverse method. It reduces the number of intersection tests by using image coherence, and interpolates data efficiently along a ray by taking advantage of the characteristics of tetrahedral cells.

In our object order approach, we approximate volume rendering images with a stack of concentric spheres. On the assumption that sampling points along a viewing ray in the image order approach are arranged uniformly, they form a group of concentric spheres around a viewing point. These spheres are extracted as isosurfaces (iso-distance surfaces) located at the same distance from a viewing point. The

triangles are scan-converted on a frame buffer in back-to-front order to generate a volume rendering image. Commercial graphics hardware capable of rendering partially transparent triangles can be used to improve the performance.

In this chapter, we first describe a light scattering equation that has been used for photo-realistic image synthesis, and then give a simplified equation called a brightness equation for volume rendering techniques. Next, we propose two techniques based on the image and object order approaches. Finally, we use the results of the performance evaluation to show how these techniques can be effectively applied.

4.1 Light scattering equation

The synthesis of images with clouds has been addressed by a number of investigators. Blinn introduced the use of density models in computer graphics [Blin82]: He considered plane parallel atmospheres and assumed that the clouds are composed of a number of particles whose reflectivity (albedo) is small. Other researchers adapted his models to more general shapes. Max defined clouds as densities with boundaries defined by analytic functions [Max83]. Voss fractally generated densities with a series of plane parallel models, yielding images of striking realism [Voss83]. Kajiya and Von Herzen gave an alternative model that deals with multiple scattering against particles with high albedo, and they further developed Blinn's low albedo model and gave a ray-tracing algorithm for it. Up to this point, the light sources were outside the cloud and the model described how the particles in the clouds scatter and absorb the light.

Here, we discuss the relevant physical parameters and set up an equation that describes the scattering of radiation in the clouds. The scattering equation can be derived by considering a differential cylindrical volume (tube) dt , where dt is the length of the tube, which has a unit cross-section area and whose direction is expressed by using a vector $\vec{D}$. If we consider a smaller tube that has a radius, r , identical to the particle's and contains a particle, we find that the difference in brightness (intensity) B between the two ends of the tube is given by

$$dB(\vec{X}, \vec{D}) = -\text{absorbed} + \text{scattered} \quad (4.1)$$

$$= (-B(\vec{X}, \vec{D}) + \int p(\vec{D}, \vec{D}') \times B(\vec{X}, \vec{D}') d\omega) \times \pi r^2 \rho' dt, \quad (4.2)$$

where ρ' is the number of particles per unit volume (see Figure 4.1).

This expression tells us that the light scattered in the direction $\vec{D}$ is a linear operator of the light

incident upon the particle from all angles. The function $p(\vec{D}, \vec{D}')$ is called the phase function, and gives the amount of light scattered from direction $\vec{D}$ to direction $\vec{D}'$.

The scattering equation is solvable analytically only in a few very special cases. Moreover, it is very difficult to solve even numerically without assumptions that reduce the dimensionality of the brightness $B(\vec{X}, \vec{D})$, a function of six real variables. For photo-realistic image generation, low-albedo particles have often been assumed; this leads to a single scattering of the radiation from the light source to the eye. In this case, only the directions from the light sources to the particle position need to be considered in the scattering term of Eq. 4.2.

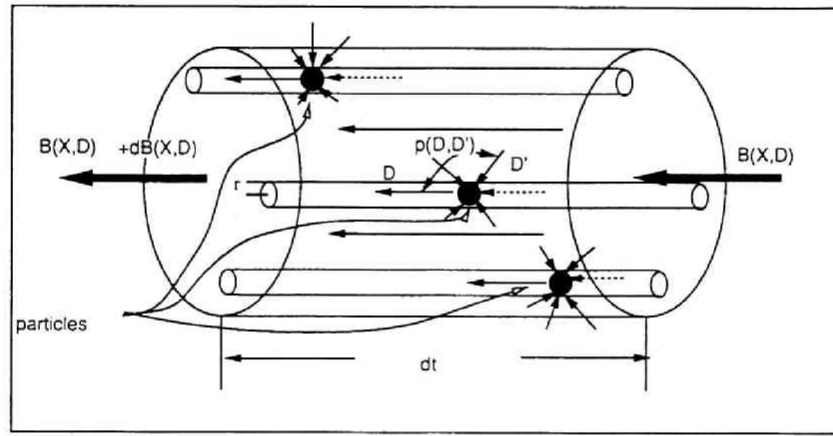


Figure 4.1: Difference in brightness B between the two ends of a tube

4.2 Brightness equation

For volume rendering techniques, a different assumption has been used, in which the cloud particle itself emits light with no scattering. On this assumption, the scattering term of Eq. 4.2 is replaced with a term that represents an emission, and the brightness becomes a function of a single parameter t , which is used to parameterize a straight line (a viewing ray) between an eye position and a particle position:

$$dB(t) = -\text{absorbed} + \text{emitted} \quad (4.3)$$

$$= (-B(t) + c(t)) \times \pi r^2 \rho(t) dt \quad (4.4)$$

$$= (-B(t) + c(t)) \times \rho(t) dt \quad (4.5)$$

$$\frac{dB(t)}{dt} = (-B(t) + c(t)) \times \rho(t) \quad (4.6)$$

where a new variable ρ , defined by $\pi r^2 \rho l$, denotes the number of particles per unit length along a tube whose radius is identical to the particle's, the variables, $\bar{D}$ have been removed on account of the assumption of no scattering, and $c(\vec{X})$ denotes the luminosity per unit area of a particle projection. (In the following discussion, we use the word 'color' for the luminosity.)

Although the introduction of the emission term and the removal of the scattering term cannot make the equation usable for photo-realistic image synthesis, they give two major advantages:

- They reduce the computation time spent on integrating the particle density from the light sources to the particle position.
- They emphasize the internal structure of the scalar field that might be occluded.

Here, we solve the brightness equation Eq. 4.6 by use of the integrating factor $\exp(\int_{t_n}^t \rho(u) du)$ (see the appendix). The solution is as follows:

$$B = \int_{t_n}^{t_0} c(t) \times \rho(t) \times \exp\left(-\int_t^{t_0} \rho(\lambda) d\lambda\right) dt. \quad (4.7)$$

In general, the brightness B and the luminosity c have three components of red, green, and blue. The parameters t_0 and t_n represent the nearest and the furthest distances from a viewing point to the volume along a viewing ray.

The exponential term in Eq. 4.7 expresses the probability that a luminosity can reach an observer's eye, according to the approximation given by a Poisson distribution. The luminosity (c) is not constant in general, because it is calculated from the scalar value and the scalar gradient at a given location by using some shading model to provide a satisfactory illusion of volume data. To solve it numerically, we subdivide the domain of integration into n subdomains, in which we can regard the luminosity as a constant value,

$$B = B_1 + \cdots + B_k + \cdots + B_n, \quad (4.8)$$

where

$$B_k = \int_{t_k}^{t_{k-1}} c_k \times \rho(t) \times \exp\left(-\int_t^{t_0} \rho(\lambda) d\lambda\right) dt. \quad (4.9)$$

The k th term is expanded as

$$B_k = \left(\prod_{i=0}^{k-2} \exp\left(-\int_{t_{i+1}}^{t_i} \rho(\lambda) d\lambda\right) \right) \times \int_{t_k}^{t_{k-1}} c_k \times \rho(t) \times \exp\left(-\int_t^{t_{k-1}} \rho(\lambda) d\lambda\right) dt \quad (4.10)$$

$$= \left(\prod_{i=0}^{k-2} \exp(-\int_{t_{i+1}}^{t_i} \rho(\lambda) d\lambda) \right) \times c_k \times (1 - \exp(-\int_{t_k}^{t_{k-1}} \rho(\lambda) d\lambda) dt), \quad (4.11)$$

where the simplification based on integration by substitution is applied to the final integration term in the second line (see the appendix). Let us define an opacity value, α_k , in the k th subdomain:

$$\alpha_k = 1 - \exp(-\int_{t_k}^{t_{k-1}} \rho(\lambda) d\lambda) \quad (4.12)$$

$$\approx 1 - \exp(-\rho_k \times \Delta t), \quad (4.13)$$

where Δt is the length of the subdomain and ρ_k is a representative density. Typically, these opacity values are specified from scalar values in a table, assuming that Δt is fixed [Levo88]. Implicitly, this relates a particle density to a scalar value. If an actual length $\Delta t'$ is not the fixed one, the following correction is applied:

$$\alpha_{k'} = 1 - (1 - \alpha_k)^{\frac{\Delta t'}{\Delta t}} \quad (4.14)$$

Finally, we can express the brightness equation in discrete systems. Eliminating an exponential term from Eqs. 4.13 and 4.11, we have

$$B = \sum_{i=1}^n c_i \times (\alpha_i \prod_{j=1}^{i-1} (1 - \alpha_j)). \quad (4.15)$$

Eq. 4.15 is useful when Eq. 4.7 is calculated in a discrete form.

4.3 Image order approach

For cloudy representation of a given volume dataset, it is necessary to calculate Eq. 4.15 for each viewing ray that passes through an eye position and a pixel position on an image screen. In the image order approach, for each viewing ray, a scalar value s_i and a scalar gradient ∇s_i are computed at the center of basically equal-sized volume sample segments along the ray in front-to-back order, by interpolating from the scalar values and scalar gradients at the node points of the tetrahedral cell that includes the sampling location. The pseudo-color and opacity α_i at the sampling location are determined by reference to pre-defined color and opacity lookup tables from the sampled scalar value s_i . A shaded color c_i is then computed by shading the pseudo-color, using some shading model such as Phong's model [Phon75], where the surface normal is the normalized scalar gradient at the location. An implementation example is as follows:

- Cast a ray, which is equivalent to a pixel position, into a volume dataset.
- Interpolate a scalar value and a scalar gradient at a sampling point along the ray.
- Calculate the luminosity and opacity by using transfer functions and some shading model.
- Composite the color values by using the opacity for the final pixel values.

The corresponding algorithm is as follows:

For a pixel position (p)

For a sampling point (m)

1. $s_m^p \leftarrow \text{interpolate-in-cell}(s^{\text{node1}}, s^{\text{node2}}, s^{\text{node3}}, s^{\text{node4}})$
2. $\nabla s_m^p \leftarrow \text{interpolate-in-cell}(\nabla s^{\text{node1}}, \nabla s^{\text{node2}}, \nabla s^{\text{node3}}, \nabla s^{\text{node4}})$
3. $\alpha_m^p \leftarrow \text{opacity}(s_m^p)$
4. $c_m^p \leftarrow \text{luminosity}(\text{color}(s_m^p), \nabla s_m^p)$
5. $B_m^p \leftarrow B_{m+1}^p \times (1 - \alpha_m^p) + c_m^p \times \alpha_m^p$

where

- B_m^p is the brightness resulting from integrating Eq. 4.15 from the n th sampling point to the m th sampling point ($n \geq m$):

$$B_m^p = \sum_{i=m}^n c_i^p \times (\alpha_i^p \prod_{j=m}^{i-1} (1 - \alpha_j^p)). \quad (4.16)$$

The final brightness is $B^p = B_1^p$.

- s_m^p , ∇s_m^p , α_m^p , and c_m^p denote a scalar value, a scalar gradient, an opacity, and a luminosity at the m th sampling point along the p th viewing ray (pixel position).
- $\text{interpolate-in-cell}(\cdot)$ is an operator that interpolates a value from the values at four node points (node1, node2, node3, and node4) that define a cell (in this case, a tetrahedral cell).
- $\text{opacity}()$ is a function for transferring a scalar value to an opacity value.
- $\text{color}()$ is a function for transferring a scalar value to its pseudo-color, which generally has three components of red, green, and blue.

- `luminosity(,)` means a shading model. For simplicity, we assume that the model is a function of pseudo-colors and a scalar gradient.

One major advantage of the image order approach is that geometric data, such as polygonally defined objects, can be easily integrated into the above procedure. An integrated rendering algorithm will be given in the next section. On the other hand, a well-known disadvantage of this approach is that the computation cost is huge because it does not take advantage of coherence within the volume data sets. In the following sections, we will describe three techniques for

- reducing the cost of ray-face intersection tests,
- reducing the cost of interpolation of the sampled values, and
- integrating geometric data.

4.3.1 Reducing the cost of ray-face intersection tests

Related work

In order to calculate the brightness of Eq. 4.15 for a given viewing ray, an exterior face that intersects the viewing ray is first searched for. Then, the cells along the ray are traversed until it exits from another exterior face. An additional calculation to check for reentry of the ray into the volume is required in handling a nonconvex mesh.

The cost of ray-face intersection tests strongly affects the total computational expense of volume rendering. Rubin and Whitted reported that most of the total computation time of a ray-tracing program is spent on intersection tests [Rubi80]. Various algorithms for reducing the number of intersection tests have been developed. Some of them use simple bounding volumes: Rubin and Whitted replaced exhaustive searching for intersections by checking with simple bounding volumes [Rubi80], while Weghorst et al. examined how bounding volumes can be selected in such a way as to reduce the computational cost of the intersection tests [Wegh84]. Another type of improvement employs techniques for subdividing three-dimensional space. Glassner investigated the effects of partitioning the space with an octree data structure [Glas84]. Fujimoto et al. compared octrees with a rectangular linear grid. A shortcoming of this type of technique is that some rays must be cast that do not pass through any face [Fuji85].

The above algorithms take no account of the knowledge that neighboring rays are very likely to intersect the same exterior face. Obviously, a new technique that exploits image coherence must be found in order to decrease the computational expense. Weghorst et al. created an item buffer to improve the first ray-intersection test [Wegh84]. This concept can be applied to the test for the intersection of a ray and the nearest (or furthest) exterior face of a tetrahedral model. Moreover, it can be extended to an efficient ray-face intersection algorithm, if we assume that refraction, reflection, and shadow rays are not considered in the ray tracing of a tetrahedral model.

Scan-conversion of exterior faces

One promising idea for such an intersection algorithm is to cast viewing rays into a volume dataset from each exterior face, which is processed either from back to front (BTF) or from front to back (FTB). In BTF, only back-facing exterior faces are processed. In FTB, only front-facing exterior faces are processed. Note that we process about half of the exterior faces in either case. Before casting rays, we need to calculate the priority of the back- or front-facing exterior faces by depth sorting. The intersection of a viewing ray and an exterior face can be easily determined by scan-converting it on the screen. If the visualized volume is convex, the priority need not be calculated, because there is no overlapping of back- or front-facing exterior faces when they are scan-converted on the screen. In this case, exterior faces can be scan-converted at random. The intersections can be incrementally calculated by using the digital differential analyzer (DDA) approach, because they exist within the face projection on the screen. If a vertex that is shared by triangles on the screen coincides with a pixel position, duplicate viewing rays may be cast. This leads to a visual artifact in the generated image. The coordinates of the vertex can be perturbed infinitesimally to eliminate such degenerate cases.

We start the cell traverse from a position that is incrementally decided. Reference to the cell adjacency component of the tetrahedral model enables us to search for cells that intersect the viewing ray successively until the ray reaches an exterior face. When it exits, the calculated brightness is stored at the corresponding pixel position of the frame buffer. If no other exterior face is scan-converted to this pixel, the value becomes a final one; otherwise, the result is used as an initial value for calculating the brightness. The whole volume, either convex or nonconvex, is finally traversed when all the exterior faces have been processed in order of priority, as shown in Figure 4.2.

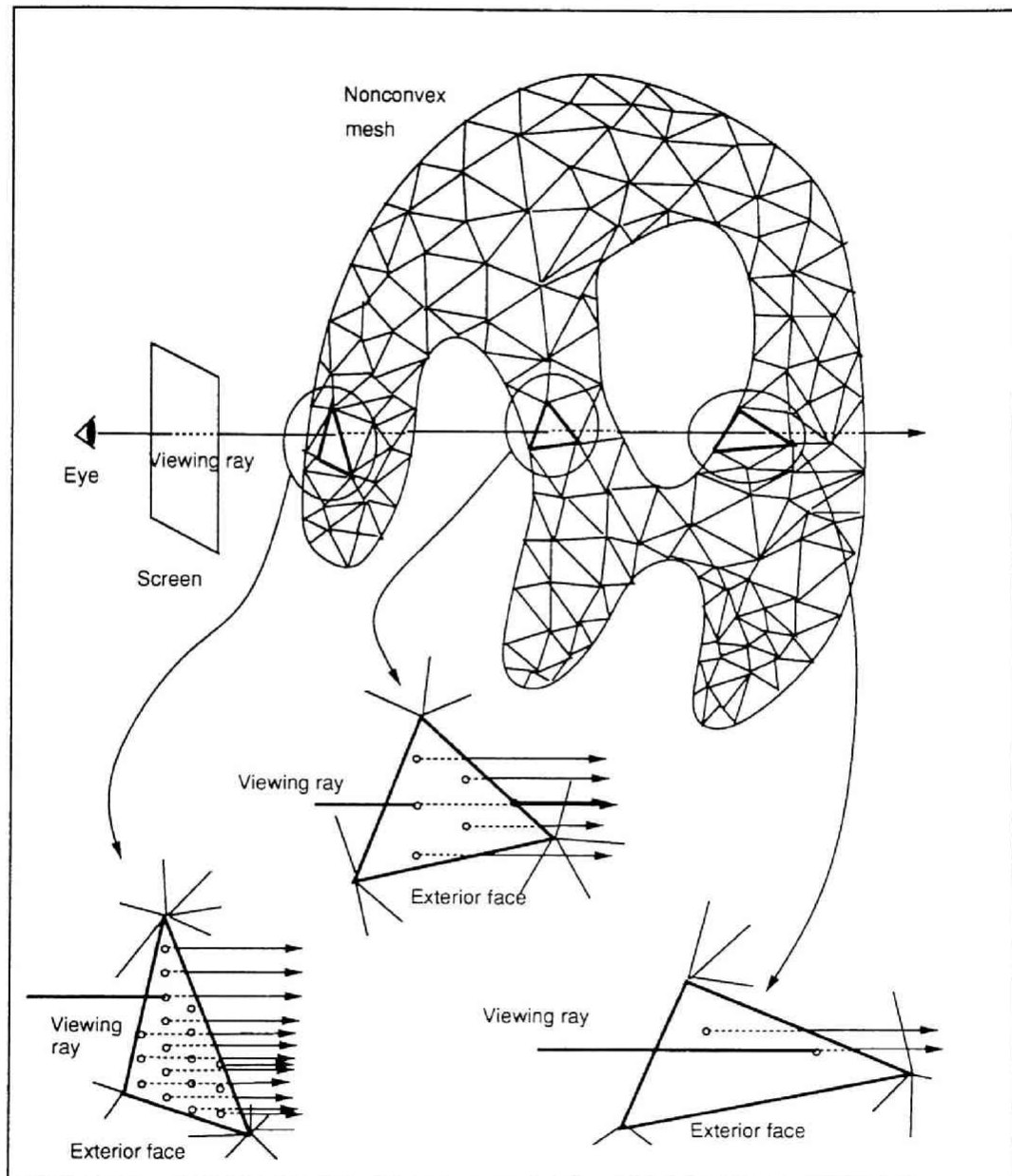


Figure 4.2: Scan-conversion of exterior faces

4.3.2 Reducing the cost of interpolation of the sampled values

Interpolation in a tetrahedral cell

In order to calculate the luminosity and opacity at a sampling point rapidly, it is very important that data, such as scalar data and scalar gradients, should be efficiently interpolated along a viewing ray. In general, a scalar value S_x at a point X in a tetrahedral cell (ABCD) is interpolated as

$$S_x = N_A \times S_A + N_B \times S_B + N_C \times S_C + N_D \times S_D \quad (4.17)$$

where

- N_A, N_B, N_C , and N_D are the interpolation functions of the tetrahedral cell at node points A, B, C, and D, respectively, and the equation $N_A + N_B + N_C + N_D = 1$ holds.
- S_A, S_B, S_C , and S_D are the scalar values at A, B, C, and D, respectively.

Obviously, it takes a lot of CPU time to interpolate scalar values at multiple sampling points in the cell by simply repeating this method. Our concern is with the interpolation at sampling points along a viewing ray. In this connection, we should remember that the data distribution in a tetrahedral cell is linear in any direction, as we have shown. Using this feature, we developed a new method of interpolation, which we call a linear sampling method. The procedure for interpolation has three steps. To illustrate this, we assume that a viewing ray enters a cell at point P in triangle (ABC).

Linear sampling method

First step: The first step is to determine the face through which the ray leaves a cell. In our approach, a ray is mapped to a point in a pixel plane, since we consider only a viewing ray, namely a ray from an eye position. Therefore, we simplify the problem to one of determining whether a point that represents the ray is included in the triangle on the screen. The triangle and the ray are expressed in a normalized projection coordinate system (NPC). We use an example to check whether the ray leaves the cell at the point Q in triangle CDA, as shown in Figure 4.3.

Note that the check is performed in two-dimensional space. If the vectors $\vec{AC}$ and $\vec{AD}$ are parallel, the point is not included in it. The ray does not leave the cell from CDA, and another triangle of the cell is checked. If the vectors are not parallel, the vector $\vec{AQ}$ can be expressed as a linear combination of

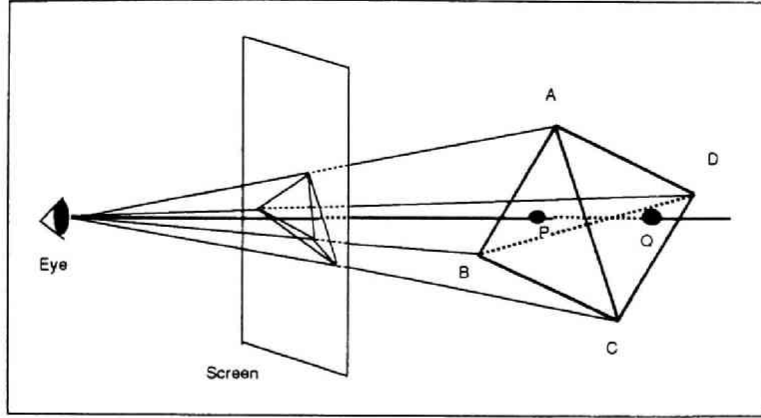


Figure 4.3: First step of linear sampling method

them:

$$\vec{AQ} = s_Q \times \vec{AC} + t_Q \times \vec{AD}, \quad (4.18)$$

where s_Q and t_Q are weighting values. The matrix system for (s_Q, t_Q) is

$$\begin{bmatrix} x_C - x_A & x_D - x_A \\ y_C - y_A & y_D - y_A \end{bmatrix} \times \begin{pmatrix} s_Q \\ t_Q \end{pmatrix} = \begin{pmatrix} x_Q - x_A \\ y_Q - y_A \end{pmatrix}. \quad (4.19)$$

where $(x_C, y_C), (x_D, y_D), (x_A, y_A)$, and (x_Q, y_Q) are the coordinates of points A, B, C, and P in NPC, respectively. The system is solvable, because the matrix is regular. If s_Q and t_Q satisfy the following conditions:

- $s_Q \geq 0$
- $t_Q \geq 0$
- $s_Q + t_Q \leq 1$,

the point Q is inside the triangle CDA and checking need not be done at further candidate faces. Otherwise, the point is outside the triangle and another triangle is checked. Our approach checks two triangles on average.

In contrast, the conventional approach first calculates the distances along the ray at which it intersects the planes of three candidate exit triangles and then selects as the exit face the triangle that has the minimum distance value. The triangle and the ray are expressed in a view reference coordinate system (VRC). This approach has the merit of being able to handle various other rays in addition to viewing rays.

However, it always needs to check three triangles in three-dimensional space, which is computationally inefficient in comparison with our approach.

Second step: The second step is to interpolate data (S_Q) at the intersection (point Q) of a cell and a viewing ray on the face determined in the first step, as shown in Figure 4.4. By replacing X with Q in Eq. 4.17, the data (S_Q) are interpolated as

$$S_Q = s \times S_C + t \times S_D + (1 - s - t) \times S_A, \quad (4.20)$$

where $s = N_C$, $t = N_D$, and $N_B = 0$. For this step, we propose two interpolation techniques that differ according to the way in which the weighting values (s, t) are calculated: an accurate technique and an approximate technique.

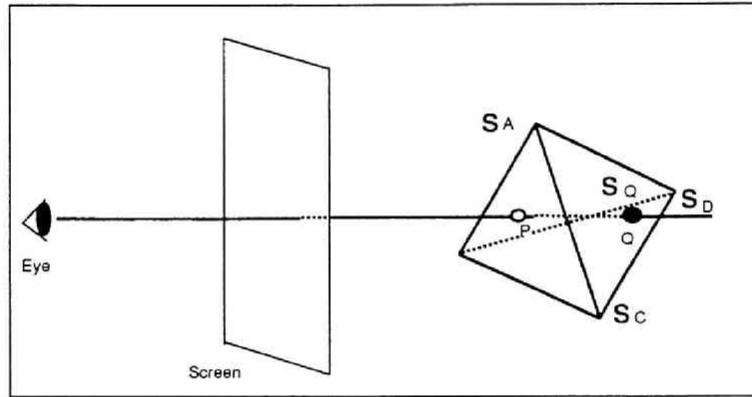


Figure 4.4: Second step of linear sampling method

In the accurate technique, the weighting values can be obtained by solving matrix systems similar to Eq. 4.19, not in an NPC but in a VRC. All coordinates should be inversely transformed from an NPC into a VRC. Before the transformation, the z -coordinate of the point Q is calculated by using Eq. 4.20. Without losing generality, we can assume that the triangle CDA is not parallel to the z -axis, because a regular triangle cannot be parallel to all three orthogonal axes x , y , and z . This assumption means that the parallel projection of the triangle CDA onto the xy -plane is not a straight line. As the point Q is in the triangle CDA,

$$\vec{AQ} = s_{QE} \times \vec{AC} + t_{QE} \times \vec{AD}, \quad (4.21)$$

where s_{QE} and t_{QE} are weighting values. Note that the values s_{QE} and t_{QE} are generally different from s_Q and t_Q of Eq. 4.19, respectively. Taking account of the x -components and y -components, the matrix

system for (s_{QE}, t_{QE}) is

$$\begin{bmatrix} x_{CE} - x_{AE} & x_{DE} - x_{AE} \\ y_{CE} - y_{AE} & y_{DE} - y_{AE} \end{bmatrix} \times \begin{pmatrix} s_{QE} \\ t_{QE} \end{pmatrix} = \begin{pmatrix} x_{QE} - x_{AE} \\ y_{QE} - y_{AE} \end{pmatrix}, \quad (4.22)$$

where (x_{CE}, y_{CE}) , (x_{DE}, y_{DE}) , (x_{AE}, y_{AE}) , and (x_{QE}, y_{QE}) are the coordinates of points C, D, A, and Q in the VRC, respectively. For efficient calculation, the coordinates of the nodal data component in the tetrahedral model should be stored in both the NPC and the VRC. If the triangle is parallel to the z-axis, the y-components and z-components, or the z-components and x-components, could be considered instead, according to whether the triangle is parallel to the x-axis or y-axis, respectively. The calculated weighting values (s_{QE}, t_{QE}) are used for (s, t) in Eq. 4.20.

In the approximate technique, the weighting values (s, t) are approximated by using the values of (s_Q, t_Q) calculated in the first step. No calculation for the inverse transformation is required in this technique. The technique ensures C_0 continuity across edges shared with neighboring triangles. However, it is not accurate, because it depends on the viewing parameters, such as the location of the viewing point. For example, when a relatively large triangle is rendered by this technique, a noticeable artifact appears in the generated image. Since we can assume that the tetrahedral cells handled are relatively small, we do not expect such an artifact to cause any problems. This expectation is confirmed by two images generated by using the two interpolation techniques given in the section on the performance evaluation.

Third step: The third step is to interpolate data at a sampling point, labeled X, along the ray by using the data (S_P, S_Q) , as shown in Figure 4.5. We may assume that S_P has been previously interpolated. Using an interior division ratio r , defined as

$$r = \frac{\tilde{P}X}{PQ}, \quad (4.23)$$

we can calculate the data at a sampling point labeled X (S_X) as

$$S_X = r \times S_Q + (1 - r) \times S_P, \quad (4.24)$$

because they are linearly distributed along the segment $\tilde{P}Q$. When interpolating the data at another sampling point along the ray in the same cell, we have only to repeat the third step.

4.3.3 Integrating geometric data

Many data visualization problems require both volume data and geometric data representing 3-D geometric objects to be displayed together in a single image. When visualizing the results of analysis of the

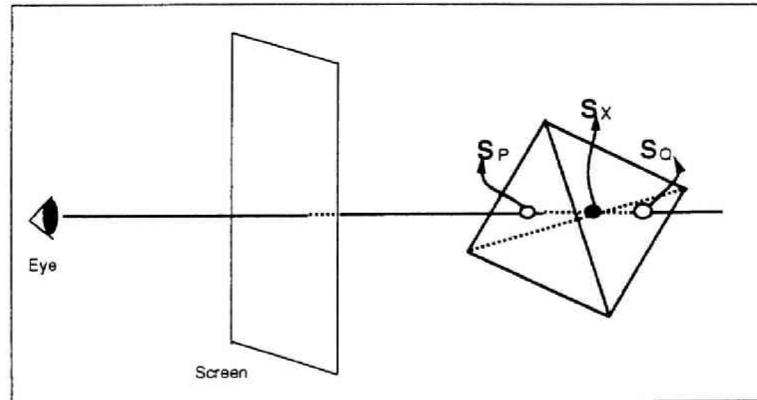


Figure 4.5: Third step of linear sampling method

thermal fluid flow around an object, for example, it is necessary to display not only the distributions of physical quantities around or on the surface of the object, such as the temperature, density, and pressure, but also the object's geometry and the tracer particles or streamlines that represent the motion of the fluid, in order to obtain further scientific insights.

Related work

Some techniques for displaying volume and surface data directly have recently been developed, using hybrid rendering algorithms. These techniques eliminate conversion artifacts by preserving the original data representations. In some of these techniques, geometrically defined objects are rendered by using a z-buffer algorithm [Good89, Kauf90] or a scanline A-buffer algorithm [Eber90], and volume data are rendered by using a volume rendering algorithm, followed by a depth-sorted compositing post-process. The first of these methods uses a z-buffer algorithm and consequently cannot render multiple layers of semi-transparent surfaces of geometrically defined objects. The technique proposed by Johnson and Moshier [John89] extends a conventional ray-tracing method by introducing a point primitive to deal with volume data.

Examples of extensions of a volume rendering algorithm based on ray tracing for handling both volume and surface data include the methods of Levoy [Levo90c] and Miyazawa and Sugimoto [Miya89]. The first uses several techniques that take advantage of spatial coherence to reduce the rendering time, but it cannot correctly render semi-transparent geometric solid objects interspersed with volume data, because it casts rays into volume data and polygon data separately and composites their results in depth-sorted

order in a post-process. In the second method, semi-transparent geometric solid objects interspersed with volume data can be correctly rendered, because rays are cast into volume data and polygon data simultaneously.

Integration of geometric data

Figure 4.6 shows an overview of the data flow in the integrated rendering algorithm. The algorithm handles volume data that are a set of scalar values defined on tetrahedral cells, surface data composed of lists of polygons, line data composed of lists of straight lines, and point data composed of lists of particles.

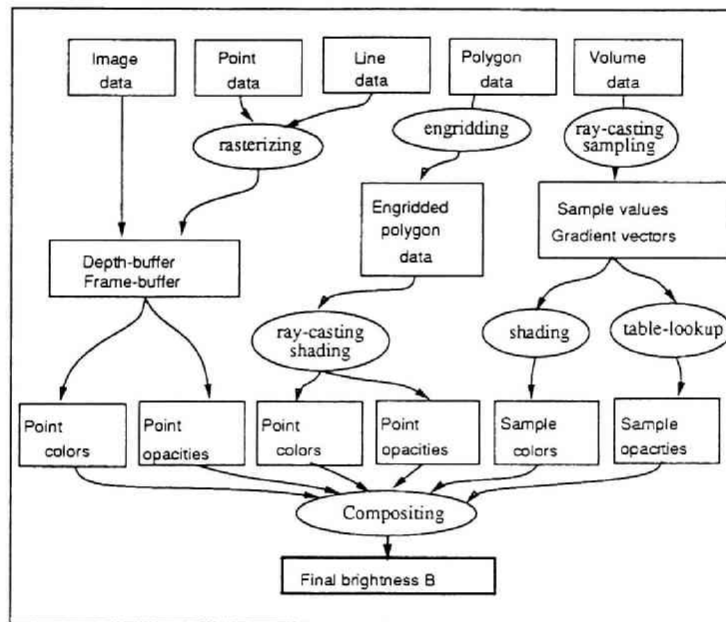


Figure 4.6: Overview of integrated rendering

In the pre-processing, a bounding volume that includes all the polygons specifying objects is defined and is divided into equal-sided voxels, and for each voxel a list of polygons that intersect it is created. These data structures allow efficient calculation of ray-polygon intersections, based on the idea of fast ray-casting using uniform space subdivision [Clea88, Fuji86].

After the pre-processing, background image data are saved in a frame-buffer, and all depths t_d of a depth buffer are set to the maximum value. Point and line data are overlaid on the background, saving the color C_d of the point and line at each pixel in the frame-buffer and the depth t_d of the point and line, that is, the distance from the viewpoint, at each pixel in the depth-buffer. In the technique, we have used

fully opaque points and lines as point and line data; the opacities α_d of the points and lines are equal to 1.

The algorithm then casts rays from the viewpoint through a pixel into the volume. First, for each ray, all intersections between the ray and polygons are computed by using a uniform space subdivision ray-casting method, and the results of ray-polygon intersections, such as the depth t_s and the outer normal of the polygon $\vec{N}_s$ at each intersection, and the surface sidedness IO at each intersection, which indicates whether the ray hits the inside or outside of the polygon, are stored as a list of data on the history of ray-polygon intersections. “Outside intersection” of a ray with a polygon that is a part of a solid object implies that the ray enters the solid object, and “inside intersection” implies that the ray exits the object. The data in the list are sorted in ascending order of the distance between the viewpoint and the intersections. Each intersection is shaded by using a model developed by Phong [Phon75] in the compositing process, where the intersection contributes to the pixel color, yielding a shaded color C_s for the intersection. The opacity of the surface α_s also is computed in the compositing process. We have used the opacity of the surface α_s , which is $1 - K_t$, where K_t is the transparency coefficient of the surface.

Next, for each ray, a sample value s_{v_i} and a gradient vector ∇s_{v_i} are computed at the center of basically equal-sized volume sample segments v_i along the ray in front-to-back order, by interpolating from the values and gradient vectors at the nodes of the cell that includes the sampling location. The pseudo color c_{v_i} and opacity α_{v_i} at the sampling location are determined by reference to pre-defined color-opacity look-up tables from the resampled value s_{v_i} at that location. A shaded color C_{v_i} is then computed by shading the intrinsic color c_{v_i} at that location, using Phong’s shading model [Phon75], where the surface normal is the normalized scalar gradient ∇s_{v_i} at the location.

Simultaneously, the shaded volume colors C_{v_i} and opacities α_{v_i} , the shaded polygon colors C_s and opacities α_s , and the color C_d and opacity α_d at the pixel in the frame-buffer are composited in depth-sorted order. If the i th contribution along a ray is a volume sample, $c_i = C_{v_i}$ and $\alpha_i = \alpha_{v_i}$ at the sample location whose depth is t_{v_i} . If the contribution is a ray-polygon intersection point, $c_i = C_s$ and $\alpha_i = \alpha_s$ at the intersection whose depth is t_s . If the contribution consists of line, point, or background data in the frame-buffer, $c_i = C_d$ and $\alpha_i = \alpha_d$ at the pixel whose depth is t_d . A final brightness value B can be calculated by substituting the c_i and α_i into Eq. 4.15.

Integration of isosurfaces

Isosurfaces are very useful for interpretation of volume data, as well as their cloudy representation. To visualize isosurfaces by the image order approach, the local opacity around the visualizing data is usually raised. However, there is a slight loss of surface sharpness, because the sampling points are not always on the isosurface. There are many requirements for achieving sharpness in visualizing the result of a numerical simulation, such as a 3-D FEM result. Max presented an algorithm for compositing a combination of density clouds and pre-computed isosurfaces [Max90]. We achieve the same effect not by explicitly extracting isosurfaces but by placing the sampling points on the intersections between the viewing ray and isosurfaces. Using the scalar data (S_P, S_Q) at the intersections and the visualizing data S , we can calculate the ratio r , which is defined above, as

$$r = \frac{S_P - S}{S_P - S_Q}. \quad (4.25)$$

This can be used to locate the sampling point on the surface.

4.3.4 Performance evaluation

Cost functions

The algorithm for calculating the final image is given in Figure 4.7.

We divide the total cost, which excludes the cost of depth sorting of exterior faces, into three parts. The cost of depth sorting will be discussed in the next section. The first part, t_{re} , is the cost of calculating the intersections of viewing rays and front- or back-facing exterior faces, and the colors and opacities at the intersections. The second part is the cost of calculating the intersections of viewing rays and interior faces, and the scalar values and scalar gradients at the intersections (the first and second steps of the linear sampling method). An interior face is one that is shared by another cell in a volume. This cost is denoted as t_c^{ac} when the accurate technique is employed as the second step, and t_c^{ap} when the approximate technique is employed. The third part, t_s , is the cost of calculating the colors and opacities at sampling points (the third step of the linear sampling method). Consequently, the total cost functions (T_{AC}, T_{AP}) of this rendering algorithm are estimated as

$$T_{AC} = N_{re} \times t_{re} + N_C \times t_C^{AC} + N_S \times t_S, \quad (4.26)$$

$$T_{AP} = N_{re} \times t_{re} + N_C \times t_C^{AP} + N_S \times t_S, \quad (4.27)$$

Calculate the priority of exterior-faces

For an exterior face

1. Scan-convert the face
2. Calculate a pixel position by DDA
 - For a viewing ray that is cast from the position**
 - a. Calculate the color and opacity
 - b. Composite them into a final brightness
 - c. Find the first cell
 - For a cell**
 - 1) Search for the exit triangle
 - 2) Calculate the weighting values (s, t)
 - 3) Interpolate the scalar and gradient
 - For a sampling point**
 - a) Calculate the ratio r
 - b) Interpolate the scalar and gradient
 - c) Calculate the color and opacity
 - d) Composite them into a final brightness
 - 4) Search for the next cell
 - d. Search for the exterior face
 - e. Calculate the color and opacity
 - f. Composite them into a final brightness

Figure 4.7: Fast algorithm for the image order approach

where

- N_{re} is the number of times that viewing rays intersect front- or back-facing exterior faces.
- N_C is the number of times that viewing rays visit tetrahedral cells.
- N_S is the number of sampling points along viewing rays.

To demonstrate the efficiency of the algorithm, let us compare it with another algorithm, which has the following features:

- The linear sampling part is the same. (The accurate technique is employed as the second step.)
- The calculation of ray-face intersection uses bounding volumes whose shape is a frustum of a quadrangular pyramid, so that the intersection tests can be reduced to simple comparisons against the limits of the volumes on a pixel plane. (Exterior faces are projected onto the plane, as in the proposed algorithm.)

In this algorithm, given in Figure 4.8, a fourth cost, t_r , is added. This is for testing the bounding volume for intersections. The first cost, t_{re} , in this algorithm includes the cost of searching among all

the candidate faces whose bounding volumes intersect a viewing ray for the face that the ray intersects.

If the cost is denoted as t_{re} , the cost function (T_{bv}) is

$$T_{bv} = N_{re} \times t_{re} + N_C \times t_C^{AC} + N_S \times t_S + N_r \times N_e \times t_r, \quad (4.28)$$

where

- N_e is the number of front- or back-facing exterior faces.
- N_r is the number of viewing rays that hit the volume. $N_r \leq N_{re}$, because a viewing ray may intersect more than one front- or back-facing exterior face.

Calculate the priority of exterior-faces

Calculate the bounding volumes of the faces

For a viewing ray

For an exterior face

1. Find the bounding volumes that the ray hits

For an exterior face surrounded by the volume

1. Find the exterior faces that the ray hits

For an exterior face that the ray hits

- a. Calculate the color and opacity
- b. Composite them into a final brightness
- c. Find the first cell

For a cell

- 1) Search for the exit triangle
- 2) Calculate the weighting values (s, t)
- 3) Interpolate the scalar and gradient

For a sampling point

- a) Calculate the ratio r
- b) Interpolate the scalar and gradient
- c) Calculate the color and opacity
- d) Composite them into a final brightness

4) Search for the next cell

- d. Search for the exterior face
- e. Calculate the color and opacity
- f. Composite them into a final brightness

Figure 4.8: Conventional algorithm for the image order approach

The main difference between the two algorithms is the object that controls the outermost loop: an exterior face for the former and a viewing ray for the latter. A ray-by-ray approach for intersection tests such as that in the latter algorithm requires a kind of “exhaustive search” for intersections. Indeed, the cost t_r of one test is trivial, because the test is a simple comparison, but the number of tests, $N_r \times N_e$.

is very large. Therefore, the product $N_r \times N_e \times t_r$ is not negligible. A face-by-face approach such as the former algorithm is more efficient, because it makes use of coherence to avoid unnecessary intersection tests.

Result and discussion

To demonstrate the efficiency of our algorithm, four irregular data sets of CFD results were rendered, as shown in Figures 4.9-4.12. These data sets, which involve cavities or holes, are categorized as nonconvex meshes. All the images were computed on an IBM 3090 at a resolution of 512×512 pixels.

Table 4.1 contains statistics on the test images.

Table 4.1: Statistics on benchmark tests for the image order approach

Dataset name	no. 1	no. 2	no. 3	no. 4
N	10167	61680	372500	557868
N_r	89694	165677	101107	127616
N_{re}	91740	194408	105018	132190
N_e	888	2716	6023	8839
N_c	2001800	7364650	6842093	11782659
N_s	2794063	7475567	2828874	4012530
Total time				
T_{bv}	520.1	2125.4	2374.5	4188.1
T_{ac}	211.5	661.3	539.0	906.7
T_{ap}	151.0	483.6	359.7	606.9
(seconds)				
T_{sort}	0.3	2.6	14.2	29.2
(seconds)				

From the values of T_{ac} in four datasets, the cost components of the total cost function in Eq. 4.26 are calculated as

$$t_{re} = 3.98 \times 10^{-4} \quad (4.29)$$

$$t_C^{AC} = 6.77 \times 10^{-5} \quad (4.30)$$

$$t_S = 1.29 \times 10^{-5} \quad (4.31)$$

The validity of these costs can be confirmed by using other datasets. By using the values of T_{bv} in four datasets and the calculated costs (t_{re}, t_C^{AC}, t_S) , the costs, t_r and $t_{re}t$, of Eq. 4.28 are calculated as

$$t_r = 2.80 \times 10^{-6} \quad (4.32)$$

$$t_{re}' = 1.49 \times 10^{-3}. \quad (4.33)$$

The value of t_{re}' is more than that of t_{re} . This is attributed to the cost of checking the exterior faces that are not intersected by a ray but that are surrounded by the bounding volumes hit by the ray. The value of t_S is less than that of t_C . This means that the cost at sampling points has less effect on the total cost than that at cell intersections. In other words, the total cost is not doubled even if twice as many sampling points are placed in such a way as to increase the image quality. By using the values of T_{ap} in four datasets, and the calculated costs, the cost, t_C^{AP} , in Eq. 4.27 is calculated as

$$t_C^{AP} = 4.12 \times 10^{-5}. \quad (4.34)$$

This cost is about forty percent lower than that of the accurate technique. Moreover, we cannot observe any noticeable difference between the two images in Figures 4.14 and 4.15, which were generated by using the two interpolation techniques, because the size of each tetrahedral cell is very small in relation to the extent of the image plane. Therefore, an approximate technique is preferable as the second step of the linear sampling method.

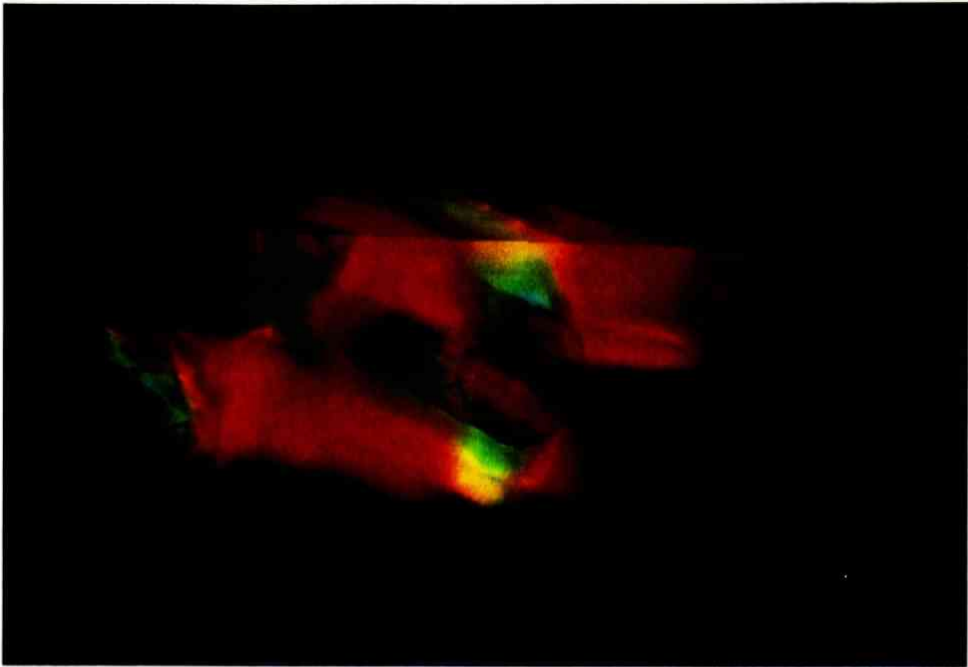


Figure 4.9: Sample image of dataset no. 1

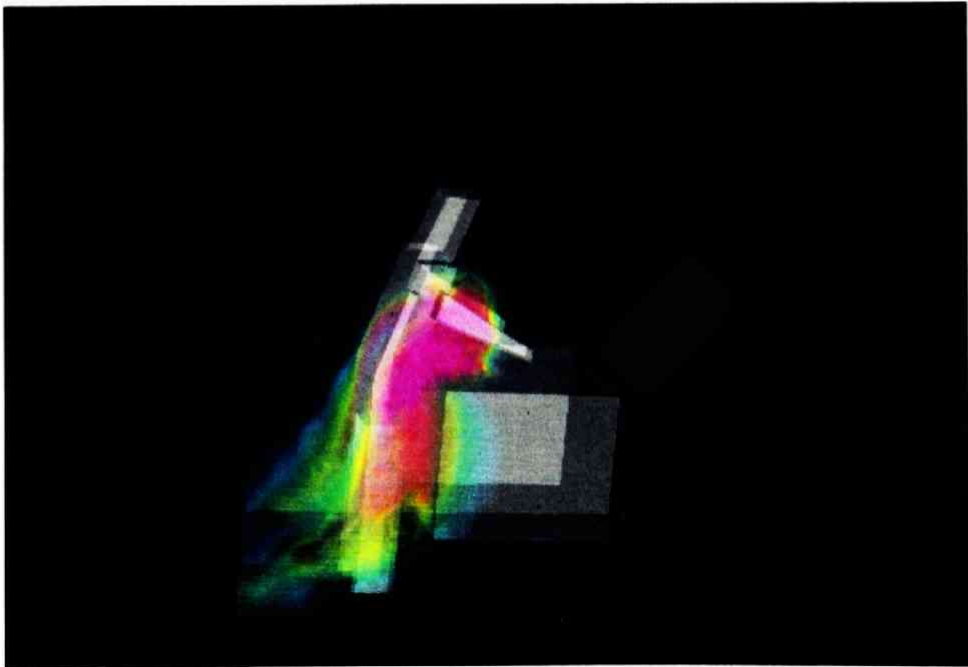


Figure 4.10: Sample image of dataset no. 2

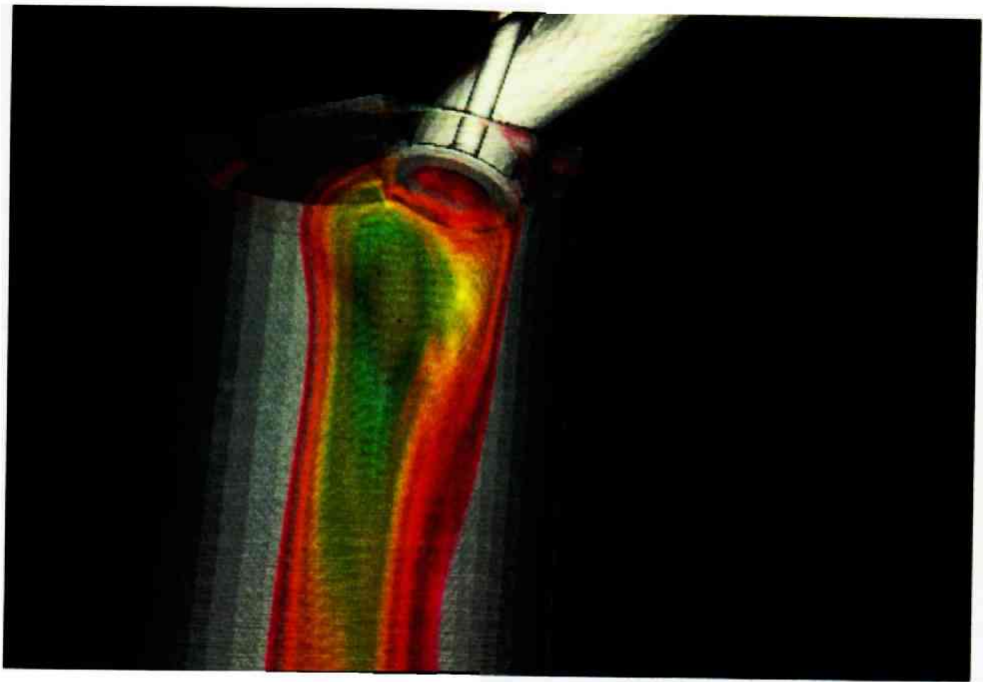


Figure 4.11: Sample image of dataset no. 3

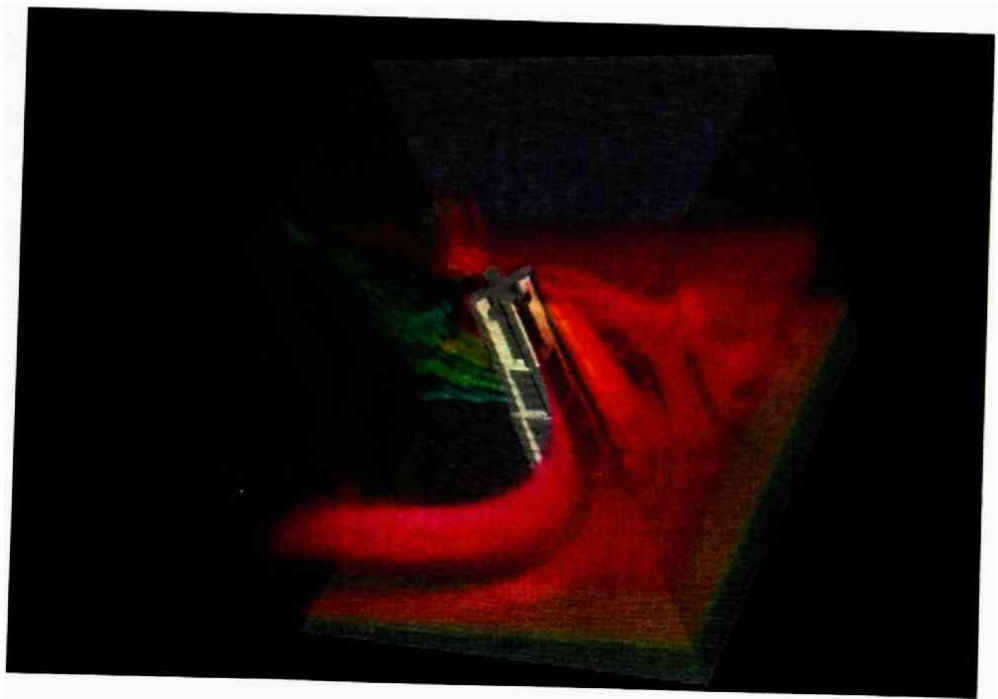


Figure 4.12: Sample image of dataset no. 4

The total time costs of dataset no. 4 were calculated by substituting these calculated costs and the statistics in Table 4.1 into Eqs. 4.26-4.28, as shown in Figure 4.13. The results of the calculation show

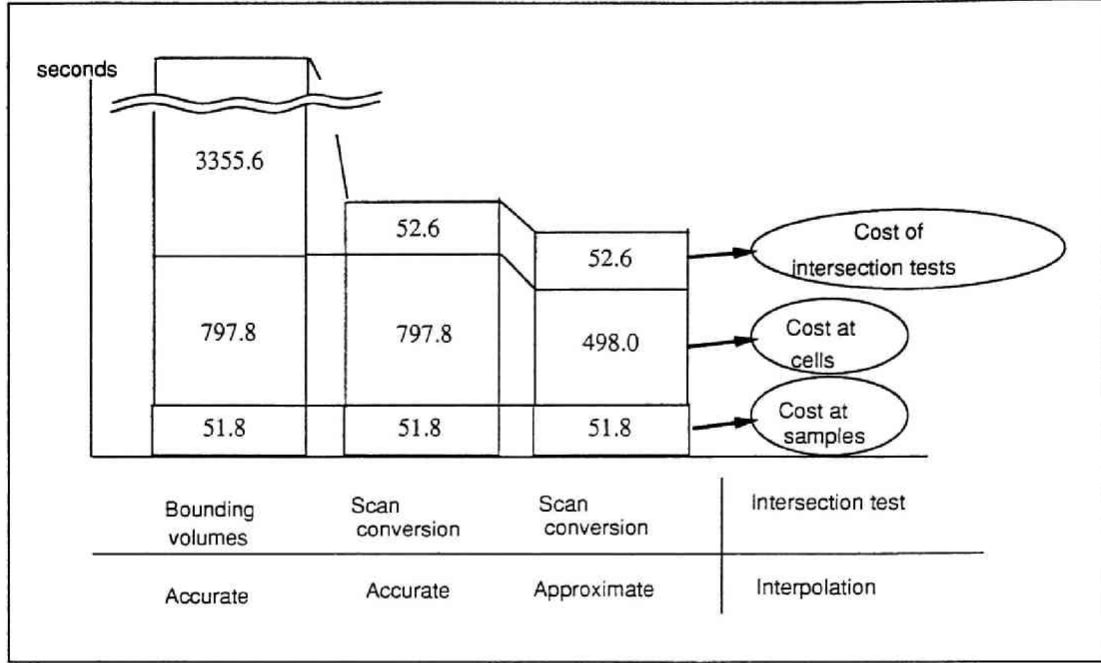


Figure 4.13: Improvement in the performance when dataset no. 4 is used.

the following:

1. If we adopt the intersection test based on bounding volumes, most of the total time cost is for the calculation of ray-face intersection tests.
2. If we adopt the intersection test based on scan conversion, the time required for the tests is very greatly reduced.
3. Interpolation at intersections of rays and cells still occupies a large part of the total computation time, although employment of an approximate interpolation technique reduces it significantly.

The total time cost is $O(N_C)$ in a rough estimate from Eqs. 4.26 and 4.27. The number N_C can be described as

$$N_C = N_C^{ray} \times N_{re}, \quad (4.35)$$

where N_C^{ray} is the average number of volume cells that a viewing ray visits from the entry point until the exit point. This number is estimated as $O(N^{1/3})$, where N is the number of volume cells. Therefore, the

performance of our method except for depth sorting of front- or back-facing exterior faces is approximately $O(N^{1/3})$.

Our algorithm requires depth sorting of front- or back-facing exterior faces as a pre-process. In the above test, an approximate priority was calculated by simply sorting the z-coordinates of the face-centroids. The priority resulting from the approximation may be incorrect and may lead to noticeable artifacts in the image. However, even this approximate sorting can generate a correct priority in many cases, because, unlike interior faces, the front- or back-facing exterior faces that overlap on a screen are likely to be sufficiently distant from each other for the sorting. The time cost, T_{sort} , of the sorting is small for the test datasets, as shown in Table 4.1. For larger datasets, this cost might not be negligible, because it is $O(N_e^2)$. In order to improve the performance, we can use a more efficient sorting algorithm such as radix sorting, whose performance is estimated as $O(N_e)$ [Suth74]. For correct sorting, we could employ the list-priority algorithm proposed by Newell [Newe72]. Although an exterior face may be split in this case, our algorithm can be made to work simply by replacing a component of an original exterior face with components of the split exterior faces in the tetrahedral model.

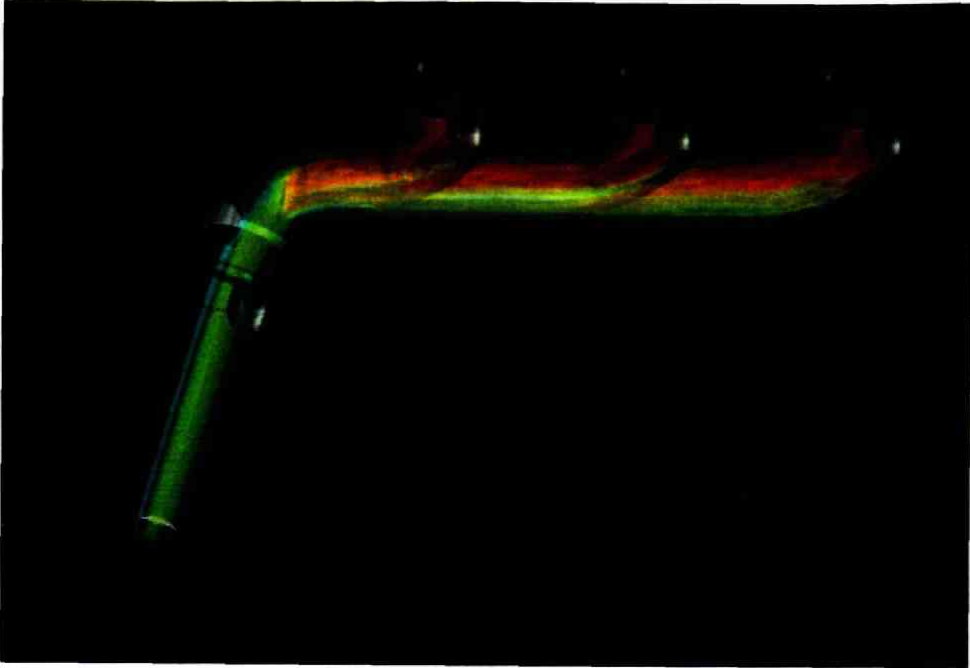


Figure 4.14: Approximate calculation of weighting values

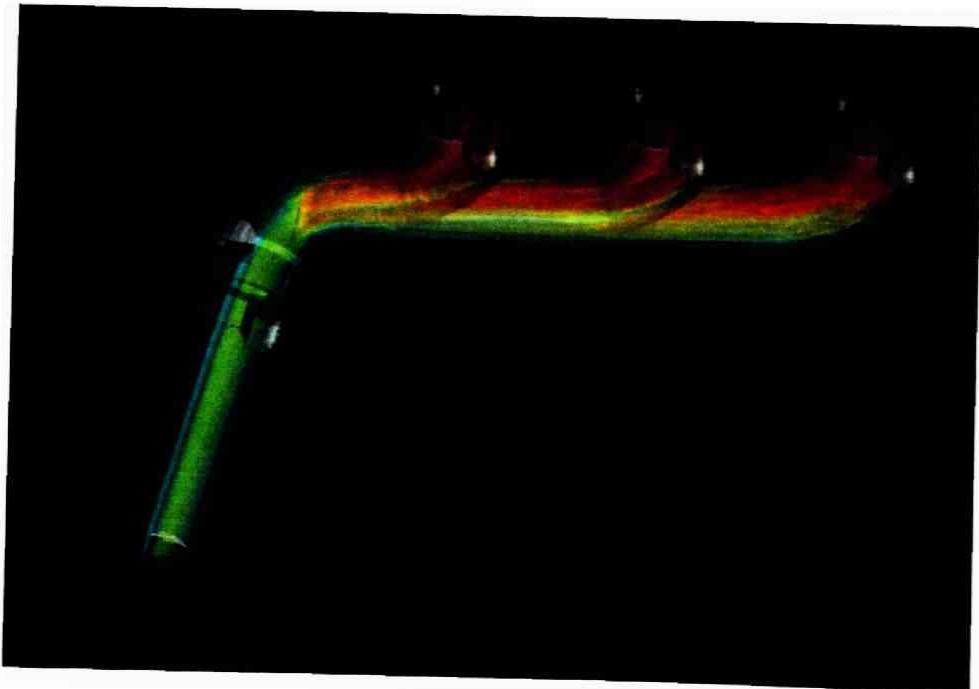


Figure 4.15: Accurate calculation of weighting values

4.4 Object order approach

The object order approach is sometimes called splatting. Shirley and Tuchman propose a Projected Tetrahedra (PT) splatting algorithm to generate a volume rendering image by using tetrahedral cells [Shir90]. In splatting, as in splatting a snow ball against a wall [West89], each tetrahedral cell is projected onto the screen in order of visibility from back to front to build up a semitransparent image. The contribution of each tetrahedral cell to the image is proportional to the thickness of the splat. The splat is rendered as a set of up to four triangles, which have a common vertex at the point of maximum thickness of the splat. At this common vertex, the opacity and the luminosity are calculated by averaging the two opacities and the two luminosities at cell entry and exit points. The opacities and the luminosities are provided through user-specified lookup tables. At all other vertices the opacities are zero and the luminosities are obtained directly through the lookup table. The opacity and the luminosity are linearly interpolated over the interior of the triangle.

Although the PT algorithm achieves efficient volume rendering, it has two major drawbacks. One is related to the visibility ordering of tetrahedral cells, which should be performed each time a new viewing position is specified. Techniques for this ordering have recently become an active area of research [Will92b]. However, some problems remain to be solved. For example, a non-convex mesh may have cycles that make a correct ordering impossible, which results in a huge computation. The other is related to visible artifacts. The silhouettes of tetrahedral cells tend to be visible, especially for coarse meshes. For simplicity, let us take a homogeneous volume dataset. In the PT algorithm, the opacity is linearly interpolated, although it is exponentially and upward convexly distributed between the maximum-thickness vertex and the zero-thickness vertex. The intermediate regions in the splat may be evaluated more translucently.

Bruce modifies the opacity values by using the thickness of the volume cell in his irregular volume rendering algorithm [Bruc92]. In the algorithm, after the faces of all the volume cells rather than the cells themselves have been sorted from back to front, each face is scan-converted in turn by linearly interpolating the vertex luminosity, opacity, and z values across the face. This results in a set of color values c_{face} , opacity values o_{face} , and z values z_{face} for each pixel in the projection of the face. Each face is composited by a technique that takes into account the distance Δz between the current face and the

face immediately behind the current one at the current pixel location. Linear approximation of Eq. 4.13 has been employed, for efficiency, to modify the opacity value.

This algorithm has the same drawbacks as the PT algorithm, with two additions. One is the low accuracy of image generation. The opacity and luminosity are evaluated on the current face, although they should be evaluated at the intermediate points between the current face and the face immediately behind it. This results in an erroneous image. The other is that no commercial graphic hardware supports face compositing on the basis of the incorporation of Δz .

In the following sections, we will propose a volume rendering technique that follows the object order approach. The technique has two major features:

- It employs an equal-sized volume sample segment, so opacity values need not be modified on the basis of exponential calculation. A set of concentric spheres around the viewing position is selected as an “object” (splat) in order to realize the sampling in the object order approach.
- It allows the luminosity and opacity values to be calculated by interpolating scalar data on the splat as well as by simply interpolating the luminosity and opacity values at the vertices. The latter interpolation may result in artificial blurring of the generated image, because the relation between the scalar values and the opacity and luminosity values is non-linear in general. If some blur exists in the image, the visualization tool user cannot tell which is responsible to the blur, the data themselves or the interpolation technique.

4.4.1 Splatting with concentric spheres

Basic concept

The outer loop of the implementation example of the image order approach in Section 4.3 is controlled by viewing rays. In this algorithm, the location of sampling points may be arbitrary at each viewing ray. In our approach, we assume that these sampling points are uniformly spaced. On this assumption, a set of these points forms a group of concentric spheres round a viewing point in the case of a perspective projection, or a group of planes perpendicular to a viewing ray in the case of a parallel projection (see Figures 4.16 and 4.17).

Because the location of sampling points is fixed, the outer and inner loops of the algorithm can be exchanged. The above algorithm calculates a final brightness for each pixel position. On the other hand,

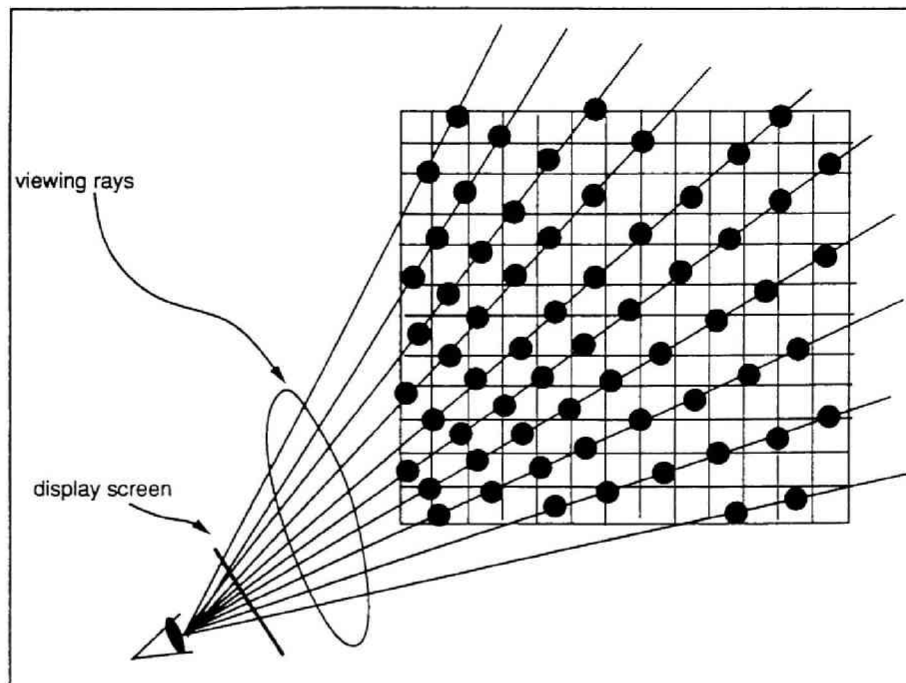


Figure 4.16: Volume rendering based on ray-casting

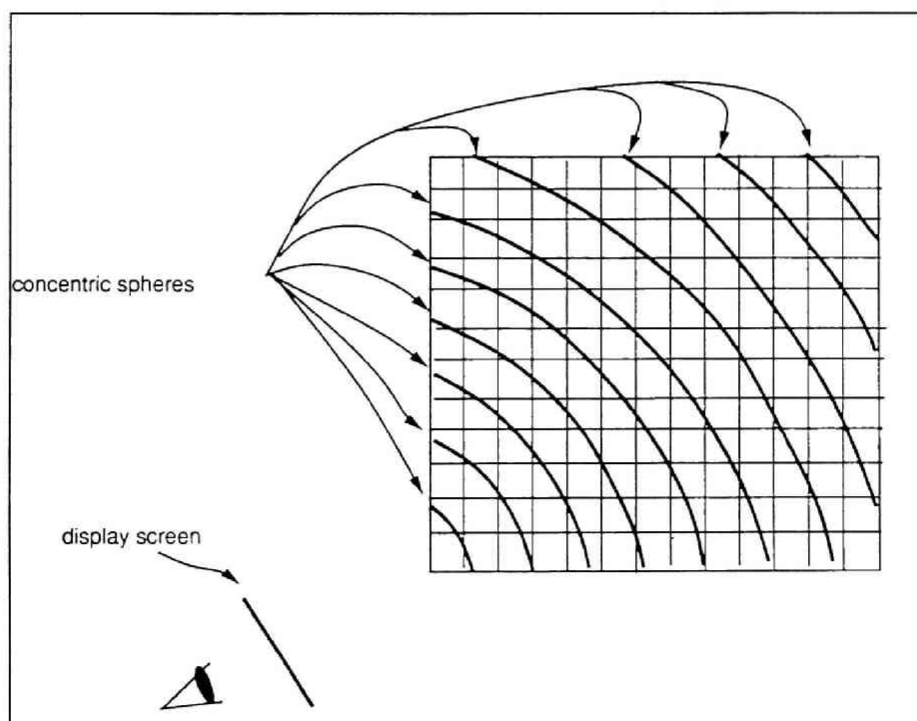


Figure 4.17: Volume rendering based on splatting

the algorithm whose loops are exchanged calculates the contributions to the final brightness values of all pixel positions for each sampling point, and therefore a frame buffer is required in order to store intermediate brightness values. In order to make the calculation at each pixel position efficient, we introduce triangles as intermediate geometries. These triangles approximate each sphere formed by the sampling points. The intersections of the sphere and volume cells are used as their vertices. We extract these triangles as an iso-distance surface (see Section 4.4.1) and scan-convert them on the screen to generate a volume rendering image (see Section 4.4.1). An overview of the algorithm is as follows:

```

For a sampling point (m)
  Extract triangles on the  $m$ th sphere
    For a triangle
      - Scan-convert the triangle on the screen
        For a pixel position (p)
           $B_m^p \leftarrow B_{m+1}^p \times (1 - \alpha_m^p) + c_m^p \times \alpha_m^p$ 

```

Figure 4.18: Overview of the splatting algorithm using concentric spheres

In order to implement this algorithm, we propose three methods (Method-1, Method-2, and Method-3), which is distinguished by the phase during which the operators `luminosity()` and `opacity()` act:

- In method-1, both operators act during the phase in which triangles are extracted.
- In method-2, `luminosity()` acts during the phase in which triangles are extracted, and `opacity()` acts during the phase in which triangles are scan-converted.
- In method-3, both operators act during the phase in which triangles are scan-converted.

This surface-based approach can generate an image at a lower cost by decreasing the number of the spheres (m) according to the required quality of the image, unlike the other surface-based approaches [Shir90, Max90]. This flexibility in controlling the efficiency is important for interactive visualization.

Preprocess

The first preprocess calculates a scalar gradient at each node point of the cells. This gradient is used for the shading computation in volume rendering. The second preprocess calculates the distance from a given viewing point to each node. For perspective projection, the distance value L_i at a grid point

labeled i , whose coordinates are (X_i, Y_i, Z_i) , is calculated as

$$L_i = (X_i^2 + Y_i^2 + Z_i^2)^{1/2}, \quad (4.36)$$

where all coordinates are expressed in a view reference coordinate system. For parallel projection, L_i is equal to Z_i (see Figure 4.19).

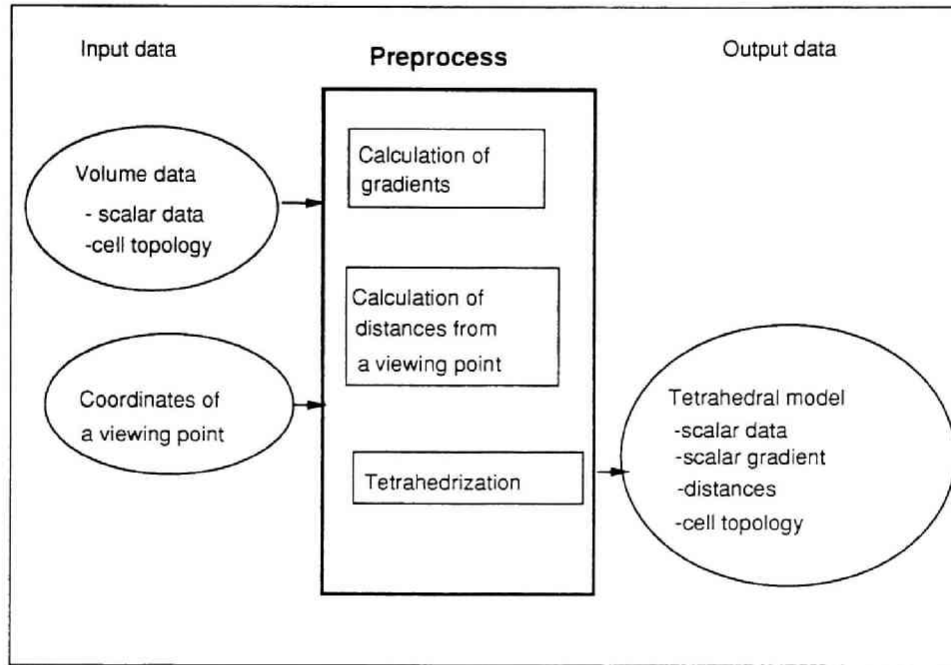


Figure 4.19: Outline of the algorithm (preprocess)

Iso-distance surface generation

Our algorithm uses an isosurface generation algorithm in order to extract iso-distance surfaces. Techniques that generate isosurfaces from a set of tetrahedral cells have been reported [Koid86, Koya90]. They represent an isosurface as a set of triangular facets extracted from a cell. The process is as follows:

In general, an edge is shared by several cells. Hashing is introduced in order to provide each intersection with a unique identifier and to avoid duplicate calculations on the edge.

To generate as many concentric sphere surfaces as the number of sampling points, distance values are arranged from minimum to maximum, in several graduations. The above process can be used to extract triangulated isosurfaces with distance values (iso-distance surfaces). The extraction may be in either descending or ascending order of the distance values. Here, the former is chosen, because a display of a

Extract triangles on the m th sphere

For a cell

Identify cells that intersect the iso-distance surface (the sphere).

For an identified cell

1. Search for edges that intersect the iso-distance surface in the cell.

For an intersecting edge

- a. Calculate the intersection by linear interpolation.
- b. Calculate values at the intersection if necessary.

2. Connect the intersections to form triangles.

Figure 4.20: Extract triangles on the m th sphere

scalar volume from back to front is more informative than one from front to back. To calculate the values at an intersection in the above process, the scalar values (s_m^{vertex}) to be visualized and their gradients (∇s_m^{vertex}) are first interpolated at the vertices of the triangles (see Figure 4.21):

$$s_m^{vertex} \leftarrow \text{interpolate-on-edge}(s^{node1}, s^{node2})$$

$$\nabla s_m^{vertex} \leftarrow \text{interpolate-on-edge}(\nabla s^{node1}, \nabla s^{node2}),$$

where

- $\text{interpolate-on-edge}(,)$ is an operator that interpolates a value from the values at two node points that define an edge of a cell.
- s^{node1} and s^{node2} denote the scalar values at two node points (node1 and node2, respectively).
- ∇s^{node1} and ∇s^{node2} denote the scalar gradients at node1 and node2, respectively.

Related operators then act on scalar values or scalar gradients in each method (method-1, method-2, or method-3). Note that no operator acts in method-3.

Method-1 (see Figure 4.22)

Calculate the values at the intersection

1. $s_m^{vertex} \leftarrow \text{interpolate-on-edge}(s^{node1}, s^{node2})$
2. $\nabla s_m^{vertex} \leftarrow \text{interpolate-on-edge}(\nabla s^{node1}, \nabla s^{node2})$
3. $\alpha_m^{vertex} \leftarrow \text{opacity}(s_m^{vertex})$
4. $c_m^{vertex} \leftarrow \text{luminosity}(\text{color}(s_m^{vertex}), \nabla s_m^{vertex})$

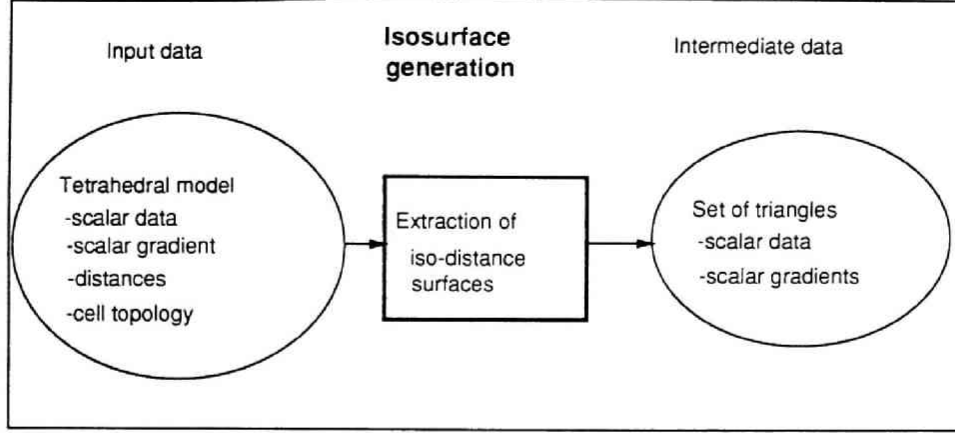


Figure 4.21: Outline of the algorithm (iso-distance surface generation)

Method-2 (see Figure 4.22)

Calculate the values at the intersection

1. $s_m^{vertex} \leftarrow \text{interpolate-on-edge}(s^{node1}, s^{node2})$
2. $\nabla s_m^{vertex} \leftarrow \text{interpolate-on-edge}(\nabla s^{node1}, \nabla s^{node2})$
3. $c_m^{vertex} \leftarrow \text{luminosity}(\text{color}(s_m^{vertex}), \nabla s_m^{vertex})$

Method-3 (see Figure 4.22)

Calculate the values at the intersection

1. $s_m^{vertex} \leftarrow \text{interpolate-on-edge}(s^{node1}, s^{node2})$
2. $\nabla s_m^{vertex} \leftarrow \text{interpolate-on-edge}(\nabla s^{node1}, \nabla s^{node2})$

where c_m^{vertex} and α_m^{vertex} denote a luminosity and an opacity at one of the vertices that defines an extracted triangle.

Scan-conversion of triangulated iso-distance surfaces

Extracted triangles have two components:

- A vertex data component, whose entry contains position coordinates and
 - an opacity and a luminosity (for method-1), or

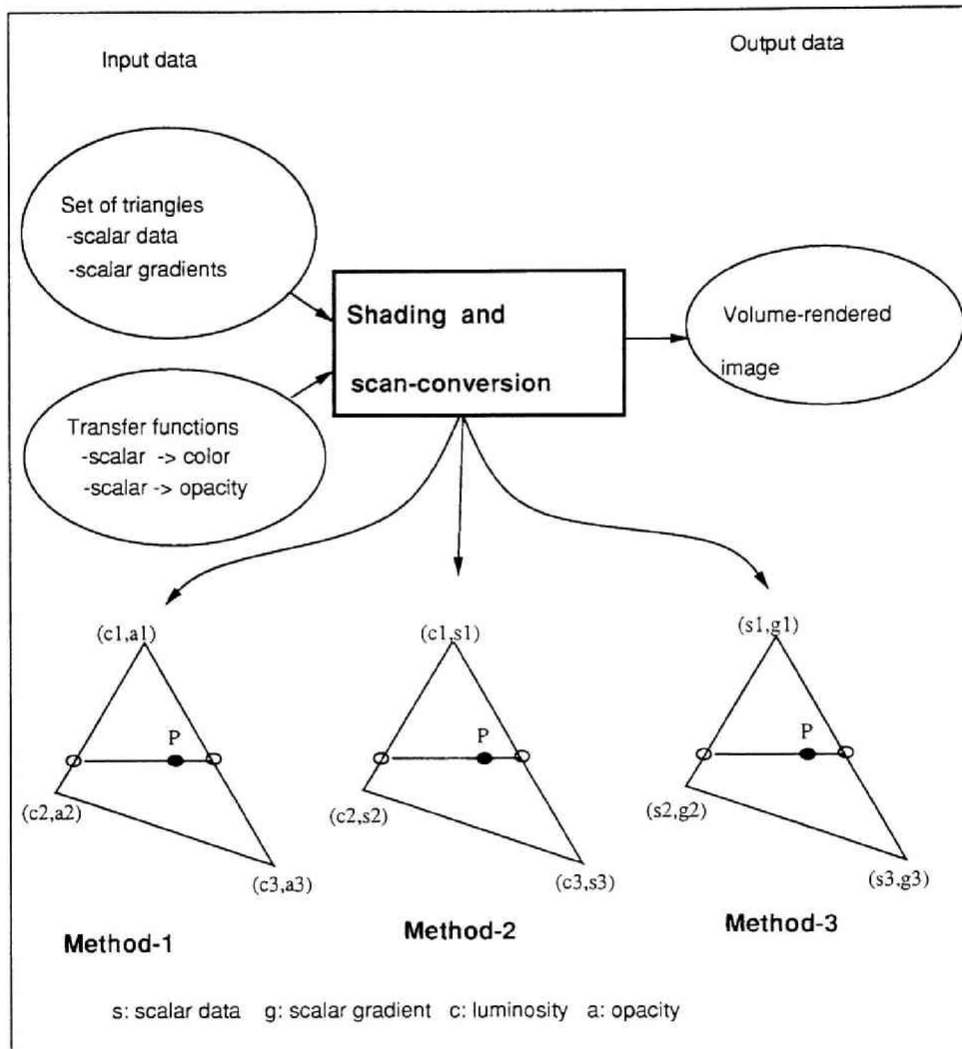


Figure 4.22: Outline of the algorithm (Method-1,2,3)

- a scalar value and a luminosity (for method-2), or
 - a scalar value and its gradient (for method-3).
- A triangle topology component, whose entry contains the identifiers of its three vertex data components.

To generate a volume rendering image, the extracted triangles are scan-converted on the screen. Before scan-conversion of the triangles, the coordinates of the vertices are transformed into a normalized projection coordinate system (NPC). The Z-components of the transformed coordinates are the same at each vertex of a triangle, because the triangle is always perpendicular to a viewing ray. Consequently, if their triangles are extracted on the same sphere, these triangles may be scan-converted on a frame buffer in an arbitrary order. Moreover, this scan-conversion is not affected by interpolation error, because an interior division ratio is preserved in the triangle through the transformation.

The scan-conversion, which we denote as $\text{scan-convert}(.,.)$, linearly interpolates a value at a pixel position, which is incrementally decided by using the digital differential analyzer (DDA), from the three values at each of the vertices of a triangle. The types of value to be interpolated are different in each method (Method-1, Method-2, and Method-3):

Method-1 (see Figure 4.22)

Scan-convert the triangle with an opacity and a luminosity

Calculate a pixel position (p) by DDA

1. $c_m^p \leftarrow \text{scan-convert}(c_m^{\text{vertex1}}, c_m^{\text{vertex2}}, c_m^{\text{vertex3}})$
2. $\alpha_m^p \leftarrow \text{scan-convert}(\alpha_m^{\text{vertex1}}, \alpha_m^{\text{vertex2}}, \alpha_m^{\text{vertex3}})$

Method-2 (see Figure 4.22)

Scan-convert the triangle with a scalar and a luminosity

Calculate a pixel position (p) by DDA

1. $c_m^p \leftarrow \text{scan-convert}(c_m^{\text{vertex1}}, c_m^{\text{vertex2}}, c_m^{\text{vertex3}})$
2. $s_m^p \leftarrow \text{scan-convert}(s_m^{\text{vertex1}}, s_m^{\text{vertex2}}, s_m^{\text{vertex3}})$
3. $\alpha_m^p \leftarrow \text{opacity}(s_m^p)$

Method-3 (see Figure 4.22)

Scan-convert the triangle with a scalar and its gradient

1. $\nabla s_m^p \leftarrow \text{scan-convert}(\nabla s_m^{\text{vertex1}}, \nabla s_m^{\text{vertex2}}, \nabla s_m^{\text{vertex3}})$
2. $s_m^p \leftarrow \text{scan-convert}(s_m^{\text{vertex1}}, s_m^{\text{vertex2}}, s_m^{\text{vertex3}})$
3. $c_m^p \leftarrow \text{luminosity}(\text{color}(s_m^p), \nabla s_m^p)$
4. $\alpha_m^p \leftarrow \text{opacity}(s_m^p)$

Figure 4.23: Scan-conversion of triangles

The resulting values (c_m^p and α_m^p) are composited for the final pixel values (B_1^p):

$$B_m^p \leftarrow B_{m+1}^p \times (1 - \alpha_m^p) + c_m^p \times \alpha_m^p. \quad (4.37)$$

The values resulting from the above three scan-conversion techniques are all the same: c_m^p and α_m^p . However, the accuracy and effectiveness of the techniques are different, as we will discuss in Section 4.4.2.

Embedding geometric primitives

It is often necessary for volumetric data and opaque geometric primitives to appear together in a single image. Our method, for example, allows a geometric model of a flow obstacle to be embedded in a volume rendering image created from a result obtained by using computational fluid dynamics (CFD). To implement this technique, we first render opaque geometries by using a Z-buffer and then scan-convert triangulated isosurfaces on the frame buffer, omitting any contribution to a pixel if the z-value (the distance from a viewing point for perspective projection) of scan-converted triangles is larger than that in the Z-buffer.

4.4.2 Performance evaluation

Four datasets of CFD results have been rendered by using the following four volume rendering methods:

- Method-1
- Method-2
- Method-3
- Method-4 (ray tracing), in which colors and opacities are calculated from scalar values and scalar gradients at sampling points by using color and opacity transfer functions and a shading model.

These datasets are categorized into non-convex meshes, because they involve cavities and holes. All calculations were performed on an IBM 3090 system, and no graphics hardware was used for this evaluation.

The image quality and time costs for images generated by the four methods are discussed.

Image quality

Four images of the smallest dataset were rendered with a resolution of 512×512 , where the number of concentric spheres (for Method-1, Method-2, and Method-3) or sampling points (for Method-4) was 100. An opacity transfer function with peak values (1.0) at four scalar values was used for rendering. The corresponding colors are red, yellow, green, and blue. We compare the three images generated by Method-1, Method-2, and Method-3 with the image generated by Method-4, which we assume to be the most accurate. In Method-1, an opacity for a scalar value may not be consistent, because the opacity, which generally has a non-linear relation to a scalar value, is linearly interpolated. This inconsistency results in artificial blurring of the generated image (see Figure 4.24). The calculation of opacities in Method-2 is more accurate. There is no blurring in the image generated by Method-2 (see Figure 4.25), but some difference of color can be recognized, because the luminosities are linearly interpolated. There is little difference between the images generated by Method-3 and Method-4 (see Figure 4.26 and Figure 4.27). The very slight difference can be attributed to the approximation of a sphere by a set of triangles.

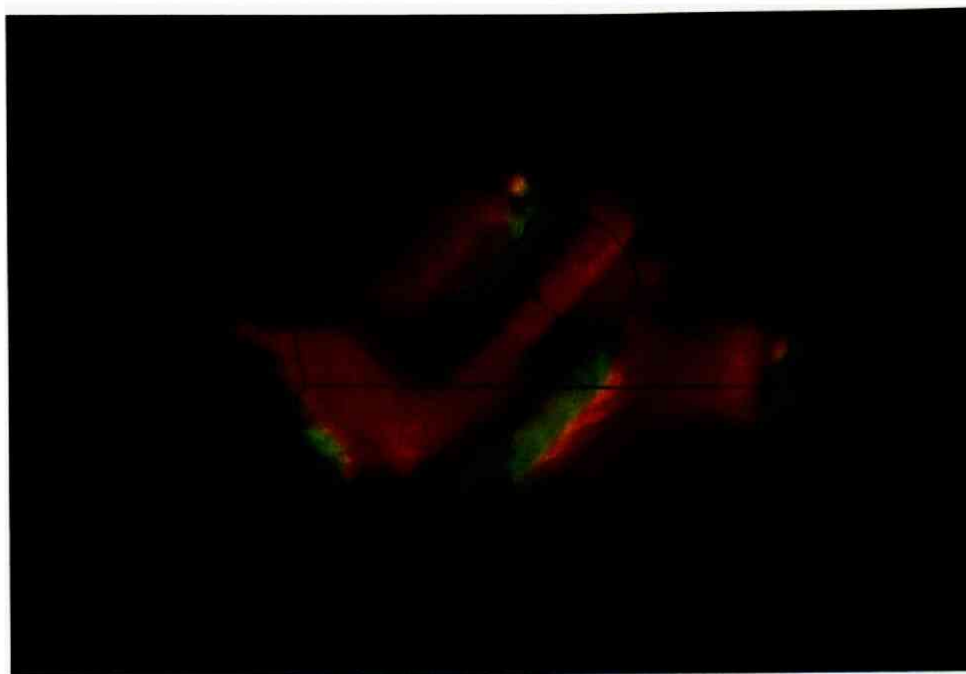


Figure 4.24: Method-1 (approximate opacities, approximate luminosities)

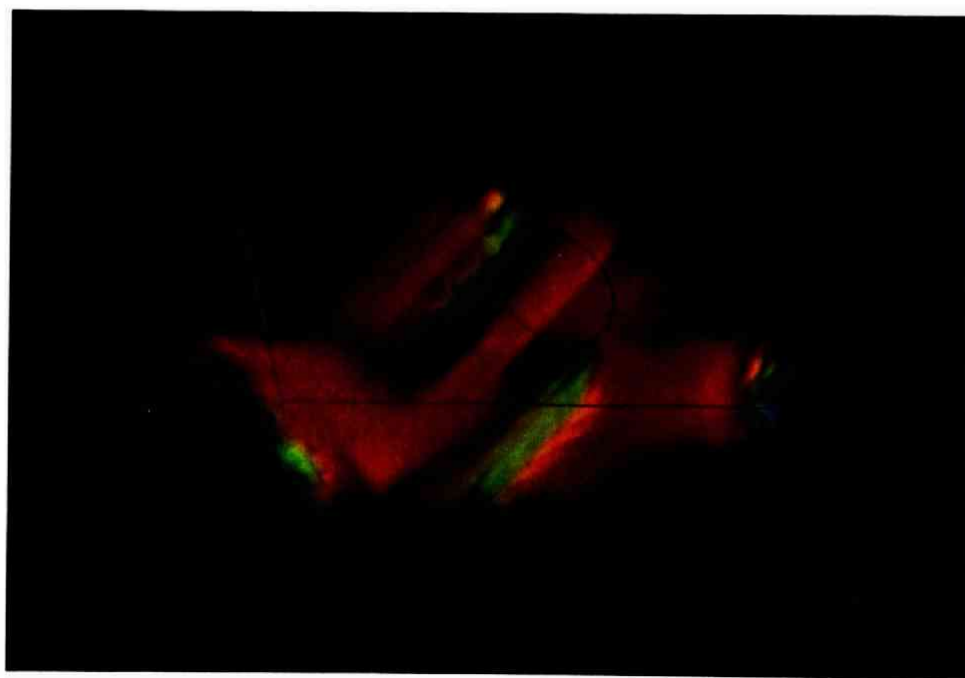


Figure 4.25: Method-2 (accurate opacity, approximate luminosities)

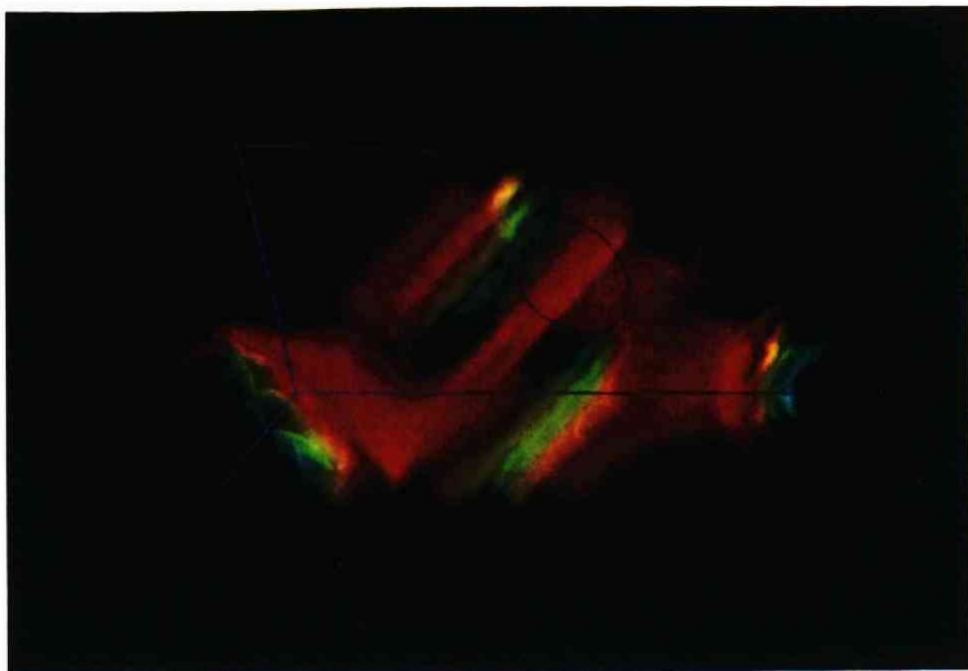


Figure 4.26: Method-3 (accurate opacity, accurate luminosities)

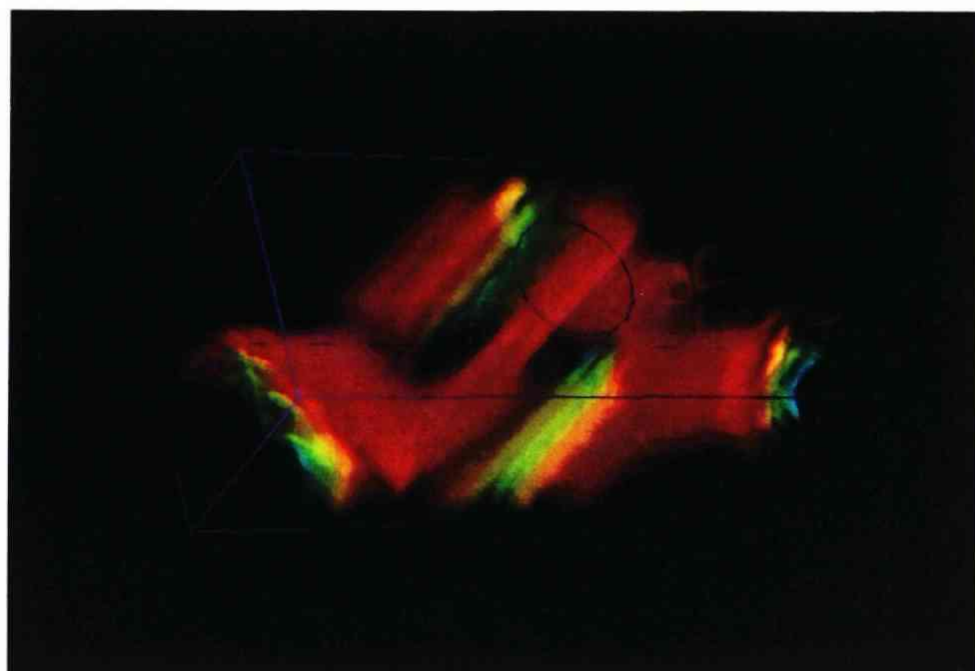


Figure 4.27: Method-4 (accurate opacity, accurate luminosities, raytracing)

Table 4.2: Results of performance evaluation

Dataset name	no. 1	no. 2	no. 3	no. 4
N	10167	61680	372500	557868
N_t	85021	392262	983154	1041613
N_v	46576	204530	503190	532952
N_s	2784136	7376272	2836272	3963339
Method-1				
Total time	54.7	181.9	481.9	642.4
T_{sfc}	14.8	78.8	372.8	519.4
T_{scan}	38.8	98.5	96.5	108.6
Method-2				
Total time	56.3	185.5	481.6	646.3
T_{sfc}	14.7	79.0	371.9	521.5
T_{scan}	40.4	101.8	97.3	110.4
Method-3				
Total time	73.1	229.7	498.3	667.0
T_{sfc}	14.4	78.1	370.0	517.2
T_{scan}	57.6	147.1	115.8	135.6
Method-4				
Total time (seconds)	211.5	661.3	539.0	906.7

Current graphics hardware can render only triangles with luminosities and opacities at their vertices, which confines our implementation to Method-1. In terms of image quality, Method-3 is the most attractive. To realize the method, however, we must wait for the advent of graphics hardware for Phong-shading. Method-2 calculates more accurate opacities than Method-1, which leads to an enhancement of the sharpness in the generated image, as shown in the above result. It can be realized if the current graphics hardware is equipped with a table lookup mechanism for converting scalar values to opacities.

Cost functions

The time costs ($T_{sfc}^i, T_{scan}^i, (i = 1, 2, 3)$) of generating iso-distance surfaces and scan-converting generated triangles were measured for four datasets. Table 4.2 contains the following statistics on the datasets: the number of tetrahedral cells (N), the number of generated triangles (N_t), the number of vertices (N_v), and the number of points at which the scan-conversions take place (N_s). Note that the difference between the total time and ($T_{sfc}^i + T_{scan}^i$) is the cost of transforming the vertices of extracted triangles into NPC. The results show that our algorithm (Method-1,2,3) gave a better performance than Method-4, especially when a small dataset was rendered.

The algorithm's efficiency is based on the following two assumptions:

- A relatively large number of viewing rays intersect a cell.
- Sampling points are uniformly arranged.

To illustrate the efficiency of our method, we select a tetrahedral cell that intersects an iso-distance surface. A basic technique for both methods is linear interpolation on an edge or in a triangle. For the method based on a ray-casting approach, the linear interpolation is performed first in a triangle (cell face), and then on an edge between intersections of the cell faces and a viewing ray [Koya91b]. Note that these calculations are independent for each viewing ray. For our method, the linear interpolation is performed first on a cell edge, in order to extract an iso-distance surface, and then in an extracted triangle by using scan-conversion techniques, which significantly reduce the computation time by taking advantage of the image coherence.

However, the above assumptions involve two disadvantages that must be traded off for the improvement in efficiency. The first is that the size of the volume cells is restricted, although cells of various sizes are generally mixed in a volume. If small cells not intersected by any ray are treated in our method, small triangles that never contribute to brightness values may be unnecessarily calculated in the cells. The second disadvantage is that the distribution of scalar data along a viewing ray is smoothed, whereas in reality node points are located with variant biasing. Therefore, our method is suitable for efficiently rendering medium-sized volumes with moderate image quality.

Here, we analyze our algorithm by introducing cost functions. In the scan-conversion of a triangle, the differences of vertex values such as opacities between one scanline and the next are first calculated and the interpolations are then performed at pixel positions. Consequently, the time cost function T_{scan}^i is estimated as

$$T_{scan}^i = t_t \times N_t + t_s^i \times N_s, \quad (4.38)$$

where

- t_t is the cost of calculating the differences in a triangle.
- t_s^i is the cost of interpolating vertex values at a pixel position in Method-i.

From the statistical information, the costs are estimated as

Table 4.3: Rendering performance

Dataset name	no. 1	no. 2	no. 3	no. 4
Performance $\frac{N_t}{T_{scan}^i}$				
Method-1	2191.3	3982.4	10188.1	9591.3
Method-2	2104.5	3853.3	10104.4	9434.9
Method-3 (triangles/sec)	1476.1	2666.6	8490.1	7681.5

- $t_t = 5.12 \times 10^{-5}$
- $t_s^1 = 1.08 \times 10^{-5}$
- $t_s^2 = 1.15 \times 10^{-5}$
- $t_s^3 = 1.75 \times 10^{-5}$.

In Method-2, a table lookup for finding an opacity from a scalar value at each pixel position is added to the processes in Method-1. The addition causes an increase in the cost from t_s^1 to t_s^2 . In Method-3, a shading calculation at each pixel position is added to the processes in Method-2. The addition causes an increase in the cost from t_s^2 to t_s^3 . A rendering performance is often evaluated by using the ratio $\frac{N_t}{T_{scan}^i}$. In this measurement, the ratio is about ten thousand triangles per second at most, as shown in Table 4.3. If we use a graphics hardware that can render one million triangles per second, T_{scan}^i can be only about a second at most.

The larger the size of the datasets, the larger the ratio of T_{sf_c} to T_{scan}^i becomes. In our method, a general-purpose isosurface generator was used to extract concentric spheres. A check is made to search for cells intersected by a concentric sphere each time such a sphere is extracted. Consequently, the time cost function $T_{sf_c}^i$ is estimated as

$$T_{sf_c}^i = t_c \times N + t_v^i \times N_v, \quad (4.39)$$

where

- t_c is the cost of searching for a cell intersected by the sphere.
- t_v^i is the cost of interpolating vertex values at an intersection of the sphere and the volume grid in Method-i.

Table 4.4: Time costs of generating iso-distance surfaces

Dataset name	no. 1	no. 2	no. 3	no. 4
Cost of surface generation				
Method-1				
T_{sfc}	14.8	78.8	372.8	519.4
T_c	8.2	49.6	299.5	448.5
T_v	6.8	29.7	73.0	77.3
Method-2				
T_{sfc}	14.7	79.0	371.9	521.5
T_c	8.2	49.6	299.5	448.5
T_v	6.8	29.7	73.0	77.3
Method-3				
T_{sfc}	14.4	78.1	370.0	517.2
T_c	8.2	49.6	299.5	448.5
T_v	6.5	28.4	69.9	74.1
(seconds)				

From the statistical information, the costs are estimated to be as follows:

$$t_c = 8.04 \times 10^{-4}$$

$$t_v^1 = 1.45 \times 10^{-4}$$

$$t_v^2 = 1.45 \times 10^{-4}$$

$$t_v^3 = 1.39 \times 10^{-4}$$

In Method-2, a shading calculation at each intersection is added to the processes in Method-3. The addition causes an increase in the cost from t_v^3 to t_v^2 . Note that the two increases in the cost, from t_s^2 to t_s^3 and from t_v^3 to t_v^2 , have the same value, 0.6×10^{-5} .

Substituting these costs in Method-1 into Eq. 4.39, we can calculate the total cost of searching for intersected cells, $T_c (= t_c \times N_c)$, and the total cost of interpolating vertex values, $T_v^i (= t_v^i \times N_v)$, as shown in Table 4.4.

The result means that most of the total cost of generating iso-distance surfaces is occupied by the cost of searching for intersected cells, especially when the size of the dataset is large. Reducing the cost of the search is an essential requirement of our method. The search can be localized by using cell adjacency or on the basis of an adequate classification of cells. This localized searching was proposed in the previous chapter.

4.5 Comparison of two techniques

We have described cloudy representation techniques based on two types of approach: image order approach and object order approach. In this final section, we discuss how these two approaches can be used properly. In general, the image order approach can be used for a high-quality renderer, and the object order approach can be used for an interactive pre-viewer. In the image order approach, geometric data such as points, lines, and semitransparent surfaces can be easily integrated with the clouds of a given volume dataset. Such integration will be especially useful when the simulation results are presented to people unfamiliar with the results. Therefore, the image order approach is suitable at the final stage of the numerical simulation. On the other hand, the object order approach is intended to give the scientist a working tool to provide a general idea of the spatial distribution of the scalar field and to roughly identify areas of interest, extrema, or hot spots at an early stage, and not necessarily to create a highly realistic or precise image for publication purposes.

Here, we consider the cost functions obtained in the benchmark tests of the two approaches, and discuss a criterion for using them in such a way as to maximize the performance. The two cost functions, T_I for our image order approach and T_O for our object order approach, are as follows:

$$\begin{aligned}
 T_I &= 3.98 \times 10^{-4} \times N_{re} + 6.77 \times 10^{-5} \times N_C + 1.29 \times 10^{-5} \times N_S \\
 T_O &= \underbrace{3.77 \times 10^{-6} \times N + 6.99 \times 10^{-5} \times N_v}_{T_{extre}} + \underbrace{5.12 \times 10^{-5} \times N_t + 1.75 \times 10^{-5} \times N_S}_{T_{scan}^3}
 \end{aligned}$$

where we select Method-3 for the object order approach and replace $T_{sf_c}^3$ with T_{extre} in Eq. 3.3. In order to compare these two functions easily, we introduce the following assumptions:

- $N_S = N_C$
- $N_t = 2 \times N_v = 1.5 \times N$
- $N_C = N_{re} \times N^{1/3}$,

where

- The first assumption implies that a single sampling point is located at each tetrahedral cell along a viewing ray.

Dataset name	no. 1	no. 2	no. 3	no. 4
N	10167	61680	372500	557868
$N^{1/3}$	21.66	39.51	71.95	82.32
N_{re}	91740	194408	105018	132190
$N_{re} \times N^{1/3}$	1987088	7681060	7556045	10881881
N_c	2001800	7364650	6842093	11782659

Table 4.5: Confirmation of the third assumption

Dataset name	no. 1	no. 2	no. 3	no. 4
N_c	2001800	7364650	6842093	11782659
N	10167	61680	372500	557868
$\frac{N_c}{N}$	196.9	119.4	65.2	21.1

Table 4.6: Average numbers of viewing rays that intersect a single cell

- The second assumption implies that each tetrahedral cell is intersected by only a single sphere surface, and that one or two triangular facets are generated at each cell.
- The third assumption implies that a viewing ray visits $N^{1/3}$ cells on average. This can be confirmed by using the statistical information obtained in the benchmark test (see Table 4.5).

By using these assumptions, the cost functions can be simplified to

$$\begin{aligned}
 T_I &= 3.98 \times 10^{-4} \times N_{re} + 8.06 \times 10^{-5} \times N_{re} \times N^{1/3} \\
 T_O &= 1.33 \times 10^{-4} \times N + 1.75 \times 10^{-5} \times N_{re} \times N^{1/3}
 \end{aligned}$$

If we suppose that the image order approach gives a better performance than the object order approach, and thus that $T_O > T_I$, then

$$\frac{N_c}{N} = \frac{N_{re} \times N^{1/3}}{N} < \frac{1.33 \times 10^{-4}}{6.31 \times 10^{-5} + \frac{3.98 \times 10^{-4}}{N^{1/3}}} < \frac{1.33 \times 10^{-4}}{6.31 \times 10^{-5}} < 3,$$

where $\frac{N_c}{N}$ is the average number of viewing rays that intersect a single cell. This result shows that the criterion for which should be selected, the image order approach or the object order approach, is whether there are fewer than three rays per cell. For the cases that we used in the benchmark tests, all the parameters ($\frac{N_c}{N}$) are above the criterion value (see Table 4.6). In order to investigate how the basic elements (N_{re} , $N^{1/3}$) influence the cost functions, we calculate the sensitivities of the cost functions to

the two independent variables $(N_{re}, N^{1/3})$ as

$$\begin{aligned}
 \frac{\partial T_I}{\partial N_{re}} &= 3.98 \times 10^{-4} + 8.06 \times 10^{-5} \times N^{1/3} \\
 \frac{\partial T_O}{\partial N_{re}} &= 1.75 \times 10^{-5} \times N^{1/3} \\
 \frac{\partial T_I}{\partial N_{re}} &> \frac{\partial T_O}{\partial N_{re}} \\
 \frac{\partial T_I}{\partial N^{1/3}} &= 8.06 \times 10^{-5} \times N_{re} \\
 \frac{\partial T_O}{\partial N^{1/3}} &= 3 \times 1.33 \times 10^{-4} \times N^{2/3} + 1.75 \times 10^{-5} \times N_{re} \\
 \frac{\partial T_O}{\partial N^{1/3}} &> \frac{\partial T_I}{\partial N^{1/3}} \quad (\frac{N_c}{N} < 6).
 \end{aligned}$$

The results show that

- The number of cells affects the performance of the object order approach more strongly when $\frac{N_c}{N} < 6$.
- The image resolution always affects the performance of the image order approach more strongly.

Since the number of cells and the image resolution are closely related to the currently available memory size and the display size, respectively, which approach will be dominant in future depends on trends in cost performance (size/cost).

4.6 Appendix

4.6.1 Solution of the brightness equation

We first multiply both members of Eq. 4.6 by the factor $\exp(\int_{t_n}^t \rho(u) du)$:

$$\exp(\int_{t_n}^t \rho(u) du) \frac{dB(t)}{dt} + \rho(t) \exp(\int_{t_n}^t \rho(u) du) B(t) = \exp(\int_{t_n}^t \rho(u) du) c(t) \rho(t), \quad (4.40)$$

then

$$\frac{d}{dt} \left[\exp(\int_{t_n}^t \rho(u) du) B(t) \right] = \exp(\int_{t_n}^t \rho(u) du) c(t) \rho(t). \quad (4.41)$$

Integrating both sides from t_n to t_0 , using the boundary condition that the brightness at t_n $B(t_n)=0$, yields

$$\left[\exp(\int_{t_n}^t \rho(u) du) B(t) \right]_{t_n}^{t_0} = \int_{t_n}^{t_0} \exp(\int_{t_n}^t \rho(u) du) c(t) \rho(t) dt \quad (4.42)$$

$$\exp(\int_{t_n}^{t_0} \rho(u) du) B(t_0) - B(t_n) = \int_{t_n}^{t_0} \exp(\int_{t_n}^t \rho(u) du) c(t) \rho(t) dt \quad (4.43)$$

This can be simplified to give

$$B(t_0) = \exp\left(-\int_{t_n}^{t_0} \rho(u)du\right) \int_{t_n}^{t_0} \exp\left(\int_{t_n}^t \rho(u)du\right) c(t) \rho(t) dt + B(t_n) \exp\left(-\int_{t_n}^{t_0} \rho(u)du\right) \quad (4.44)$$

Finally, we get

$$B = B(t_0) = \int_{t_n}^{t_0} c(t) \times \rho(t) \times \exp\left(-\int_t^{t_0} \rho(\lambda)d\lambda\right) dt \quad (4.45)$$

4.6.2 Simplification based on integration by substitution

By putting $\int_t^{t_{k-1}} \rho(\lambda)d\lambda$ as U , we get

$$\int_{t_k}^{t_{k-1}} \rho(t) \times \exp\left(-\int_t^{t_{k-1}} \rho(\lambda)d\lambda\right) dt = \int_{t_k}^{t_{k-1}} \rho(t) \times \exp(-U) dt \quad (4.46)$$

Because $\frac{dU}{dt} = -\rho(t)$ holds, then

$$\int_{\int_{t_k}^{t_{k-1}} \rho(\lambda)d\lambda}^0 -\exp(-U) dU = 1 - \exp\left(-\int_{t_k}^{t_{k-1}} \rho(\lambda)d\lambda\right). \quad (4.47)$$

Chapter 5

Data probing and stream line display techniques

In the previous two chapters, we have presented techniques for visualizing the overall features of simulation results. In this chapter, our focus will be on two techniques for displaying more localized information from a given irregular volume dataset: a data probing technique and a stream line display technique.

The data probing technique provides a virtual measurement environment, in which a data value is shown at an arbitrary position specified by an end user. Most end users are scientists or engineers who employ numerical simulation techniques to study physical phenomena which they have clarified through the measurement of related physical quantities. On the assumption that a volume dataset is a simulated physical phenomenon, the word “visualization” corresponds to the word “measurement.” Naturally, end users want to interact with their volume datasets as if they were measuring the physical quantities in their laboratory.

The stream line display technique provides a means of visualizing a 3-D vector field. One simple and common technique for visualizing a vector field is to draw arrows that depict its magnitude and direction. A drawback of this technique is that the resulting often becomes cluttered. The display of stream lines represents the vector field more clearly and can lead to a better understanding of it.

5.1 Related work

The scientific visualization community has done little research on techniques for displaying the localized features from irregular volume datasets. An important reason for the lack of such research is the inability to find an efficient method of spatial point location: that is, a method of determining that volume cell

contains an arbitrary position.

Spatial point location in irregular volume datasets is an important part of the stream line display technique, as well as of the data probing technique. When end users display stream lines, they wish to locate the starting points arbitrarily. Steve and Creon developed a virtual windtunnel for exploring three-dimensional unsteady flowfields [Brys92]. However, their current implementation is based on curvilinear volume datasets. Giles and Haimes proposed an algorithm for probing irregular volume datasets [Gile90]. The internal data structure of the algorithm has four different cell types: tetrahedra, pyramids, prisms, and hexahedra. This minimizes the memory requirements, but makes cell traverse inefficient, because the data in a cell are not necessarily distributed linearly. The algorithm restricts the data probing on a cutting plane displayed in a 2-D window.

Preparata and Tamassia have shown how to perform this operation on the volume dataset of N volume cells in $O(\log^2 N)$ time [Preparata 89]. However, their point location algorithm may be insufficiently fast for interactive operation. Neeman has applied the median cut approach to the point location problem [Neem90]. His algorithm works well on regular volume datasets, but it performs poorly on clusters of diagonal volume cells, whose faces are not parallel or nearly parallel to the coordinate axes, because adjacent cells have bounding boxes that overlap across much of their areas. Such clusters are often found in irregular volume datasets.

5.2 Data probing technique

5.2.1 Spatial point location

In the data probing technique, the volume cell that contains a given probing point must first be determined. In this respect, the conventional algorithms described in the previous section neglect a useful feature of 3-D FEM results, because they handle each volume cell independently. The 3-D region in which the FEM result is defined is filled with a number of volume cells without overlaps or gaps, and adjacent volume cells share node points. As a pre-process, we construct an adjacency graph $G(V,A)$ in which V , the nodes,¹ are the set of volume cells and A , the arcs, are volume cell faces. We have developed an efficient spatial point location algorithm based on a traversal of the graph $G(V,A)$. It assumes a starting point in any cell. This starting cell may be the cell that includes the previous probing point.

¹In the rest of this paper, we use the word “node” to describe a vertex of a tetrahedral cell.

The algorithm traverses cells along a line that is a function of a parameter t to the probing point such that $t=0$ is the starting point and $t=1$ is the target. The procedure is as follows:

1. Select a starting cell.
2. Choose the centroid of the cell as the starting point and define a traverse line.
3. Calculate an exit point (an intersection of the line and the cell face).
4. Calculate the parameter t at that point.
5. Terminate the traverse if the value of t exceeds 1.
6. Select the adjacent cell as a current cell and return to step 3 if there is an adjacent cell.
7. Select another volume cell and return to step 2 if there is no adjacent cell (the line reaches an exterior face).

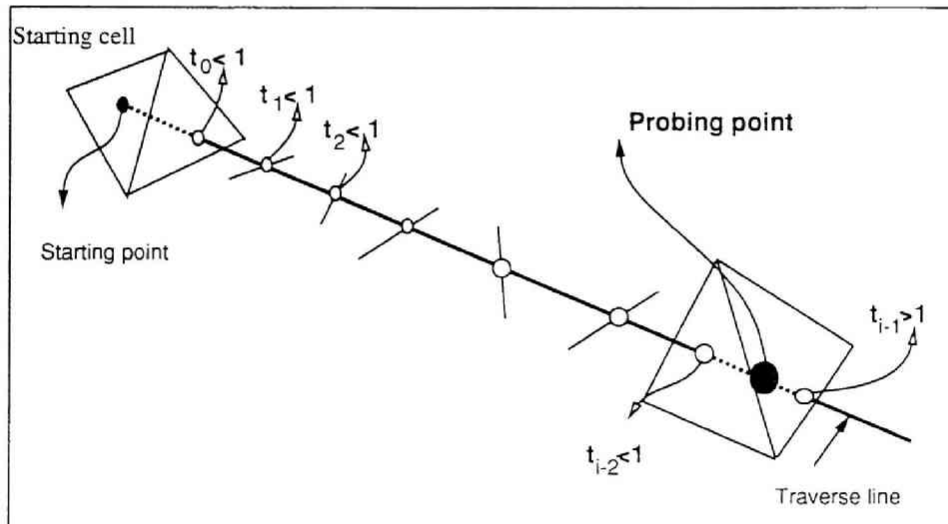


Figure 5.1: Spatial point location based on a cell traverse

Speray proposed a similar idea [Sper90]. Moreover, we developed a sophisticated approach for the case in which the probing point moves out of sight on the first traverse. This can happen when a given volume is non-convex. In this case, our system performs a re-traverse (step 7). To reduce the frequency of re-traversing, we select another volume cell that satisfies the following conditions (see Figure 5.2):

- One of the volume cell faces is an exterior face of the volume.

- The projection of the volume cell onto a plane (e.g. the XY-plane) includes the projection of the probing point.

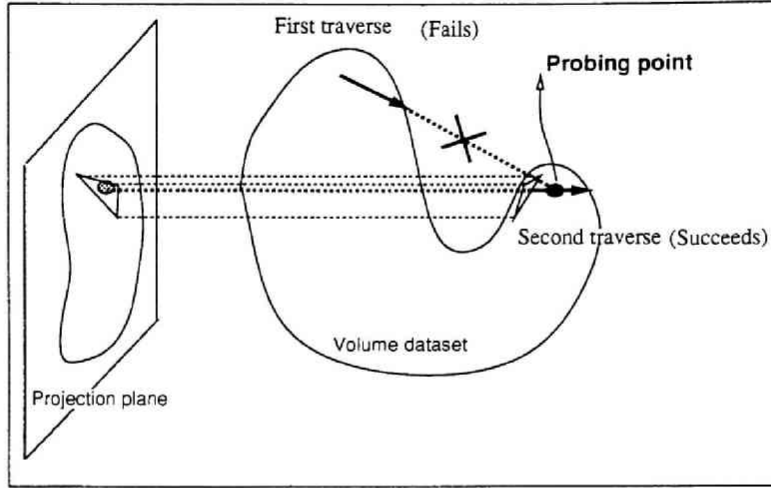


Figure 5.2: Re-traverse for non-convex volume

In step 3, the exit point ($\vec{p}_{out}$) is calculated as an intersection of a cell face and a half-line through the entry point ($\vec{p}_{in}$) parallel to the direction vector ($\vec{d}$). (We call the line a “traverse line”; see Figure 5.3.) The face on which the intersection is made can be determined by transforming it into the local coordinate system (volume coordinate system: $\vec{u}=(u,v,w,z)$) of the linear tetrahedral cell.² If the resulting coordinates satisfy $0 \leq u \leq 1$, $0 \leq v \leq 1$, $0 \leq w \leq 1$, and $0 \leq z \leq 1$, the intersection is made on the face. These local coordinates can be obtained by expressing the exit point in two ways:

$$\vec{p}_{out} = \sum_{i=0}^3 N_i(\vec{u}) \times \vec{p}_i \quad (5.1)$$

$$\vec{p}_{out} = \vec{p}_{in} + \Delta t \times \vec{d} \quad (\Delta t > 0), \quad (5.2)$$

where $\vec{p}_i$ ($i=0,1,2,3$) are the four vertices of the tetrahedral cell, Δt is defined as an increase in the parameter t , and $N_i(\vec{u})$ is an interpolation function of the i th node point.³ We can solve $\vec{u}$ and Δt as simultaneous equations after $\vec{p}_{out}$ has been eliminated from the two equations.

² Each value of the coordinates (u,v,w,z) denotes the ratio to the entire volume of the volume of the partial tetrahedron composed of a given point and its opposite triangular face. Therefore, one of the four coordinates is automatically determined from the other three coordinates, because the equation $u+v+w+z=1$ holds. See the appendix of Chapter 2.

³ When we check whether the intersection is made on the i th face, we set $N_i(\vec{u})$ to 0.

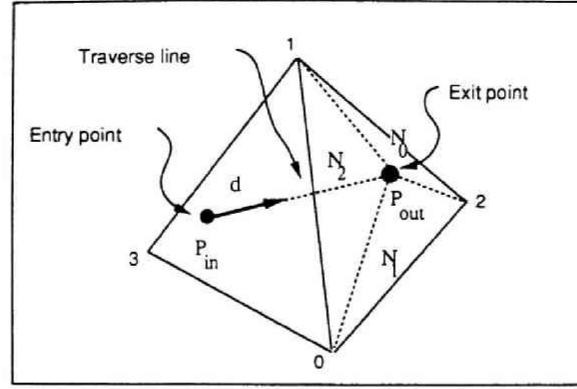


Figure 5.3: Calculation of an intersection at a cell face

5.2.2 Data interpolation

Once a tetrahedral cell that contains the probing point has been determined, it is necessary to estimate the data value at the point. In a tetrahedral cell, a data value is generally interpolated from nodal data S_i by using interpolation functions $N_i(\bar{u})$, which are expressed in the volume coordinate system

$$S(\bar{X}) = \sum_{i=0}^3 S_i \times N_i(\bar{u}), \quad (5.3)$$

where $\bar{u}$ denotes the set of local coordinates in the cell. Because the probing point is given in a world coordinate system, a coordinate transformation is necessary in order to calculate the data value. From Eq. 2.5, the coordinates of the probing point (x,y,z) can be given as follows:

$$\begin{aligned} \bar{X} &= \begin{pmatrix} x \\ y \\ z \end{pmatrix} \\ &= u \times \begin{pmatrix} x_0 \\ y_0 \\ z_0 \end{pmatrix} + v \times \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix} + w \times \begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix} + (1 - u - v - w) \times \begin{pmatrix} x_3 \\ y_3 \\ z_3 \end{pmatrix}. \end{aligned}$$

where (x_i, y_i, z_i) denotes the coordinates of the i th node point of the tetrahedral cell. Rearrangement based on the matrix formulation gives

$$\begin{pmatrix} x - x_3 \\ y - y_3 \\ z - z_3 \end{pmatrix} = \begin{bmatrix} x_0 - x_3 & x_1 - x_3 & x_2 - x_3 \\ y_0 - y_3 & y_1 - y_3 & y_2 - y_3 \\ z_0 - z_3 & z_1 - z_3 & z_2 - z_3 \end{bmatrix} \times \begin{pmatrix} u \\ v \\ w \end{pmatrix} \quad (5.4)$$

In 3-D FEM results that are successfully generated, a tetrahedral cell is never degenerated. Because the above matrix is not singular in this case, the local coordinates (u,v,w) can be obtained by multiplying

both members of the above equation by the inverse matrix:

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{bmatrix} x_0 - x_3 & x_1 - x_3 & x_2 - x_3 \\ y_0 - y_3 & y_1 - y_3 & y_2 - y_3 \\ z_0 - z_3 & z_1 - z_3 & z_2 - z_3 \end{bmatrix}^{-1} \times \begin{pmatrix} x - x_3 \\ y - y_3 \\ z - z_3 \end{pmatrix} \quad (5.5)$$

From this result, a data value can be evaluated by using Eq. 5.3.

5.3 Stream line display technique

A stream line can be defined by the following equation:

$$\frac{dx}{v_x} = \frac{dy}{v_y} = \frac{dz}{v_z} \quad (5.6)$$

where $\vec{v} = (v_x, v_y, v_z)$ is a velocity vector field. This line is calculated by integrating the instantaneous velocity field at each tetrahedral cell. The resulting exit point, at which the stream line leaves the cell, and the velocity vector at that point are used as the initial conditions for the adjacent cell.

On the assumption that (v_x, v_y, v_z) is constant during the time step Δt , the stream line is approximated by connecting points defined by

$$\vec{p}_{i+1} = \vec{p}_i + \vec{v}_i \times \Delta t, \quad (5.7)$$

where $\vec{p}_i$, $\vec{p}_{i+1}$, and $\vec{v}_i$ express the position vector of the i th point along the line in the tetrahedral cell, the position vector of the $(i+1)$ th point, and the velocity vector at the point of $\vec{p}_i$, respectively. The term $\vec{p}_0$ denotes the point at which the stream line enters the cell, and $\vec{v}_0$ denotes the velocity vector at that point. Δt is first determined as the time required for the traverse line passing through the entry point ($\vec{p}_0$) and parallel to the velocity vector ($\vec{v}_0$) to reach a face of the cell. The calculation is performed iteratively until further decrease of Δt leads to little change in the exit point, $\vec{p}_{out}$,⁴ for each tetrahedral cell, as shown in Fig 5.5.⁵

Kaneda et al. proposed a similar algorithm [Kane86]. However, the main feature of our algorithm is that it employs a data interpolation technique similar to the linear sampling method that we discussed in Section 4.3. To calculate the next position, $\vec{p}_{i+1}$, at each time step (Δt) from the current position, $\vec{p}_i$, we need to perform the following processes:

- Check whether the position is still included in the same tetrahedral cell.

⁴ $\vec{p}_{out}^{old}$ denotes the previous exit point in the iterative calculation. In the following algorithm, we also call $\vec{p}_{out}^{old}$ a reference exit point, because it is referred to in order to test for convergence.

⁵In this figure, we denote Δt as T.

1. Calculate the exit point $\vec{p}_{out}^{ld}$ and the time Δt required for the traverse line to reach the cell face on assumption that the velocity field is uniform in the volume cell.
2. Halve Δt and initialize the index value i .

$$\begin{aligned}\Delta t \times &= 0.5; \\ i &= 0;\end{aligned}$$

3. Interpolate a velocity vector $\vec{v}_i$ at the i th point $\vec{p}_i$ in the volume cell.
4. Calculate the exit point $\vec{p}_{out}$ and the required time t_i on the assumption that the velocity field is uniform in the volume cell.
5. Increment the index value i , move the current position forward by Δt , and return to step 3 when $t_i > \Delta t$.

$$\begin{aligned}\vec{v} &= \vec{v}_i; \\ \vec{p}_{i+1} &= \vec{p}_i + \Delta t \times \vec{v}; \\ i+ &= 1;\end{aligned}$$

6. Renew the reference exit point $\vec{p}_{out}^{ld}$ and return to step 2 when the previous exit point $\vec{p}_{out}^{ld}$ and the current one $\vec{p}_{out}$ are not close to each other.

$$\vec{p}_{out}^{ld} = \vec{p}_{out};$$

Step 5 can be changed as follows when the accuracy is considered more important than the performance:

$$\begin{aligned}\vec{v} &= \vec{v}_i; \\ \vec{p}_{i+1} &= \vec{p}_i + \Delta t \times \vec{v}; \\ \vec{v} &= \frac{1}{2} \times (\vec{v} + \vec{v}_{i+1}^*) \\ \vec{p}_{i+1} &= \vec{p}_i + \Delta t \times \vec{v}; \\ i+ &= 1;\end{aligned}$$

where $\vec{v}_{i+1}^*$ is a velocity vector at a position $\vec{p}_{i+1}$.

Figure 5.4: Integration of the velocity field within a cell

- Interpolate a velocity vector $\vec{v}_{i+1}$ at that position, if it is in the same cell.

For the first process, we compare Δt with the time (t_i) required for the half-line through the point $\vec{p}_i$ and parallel to the velocity vector $\vec{v}_i$ to reach a face of the cell. If Δt is larger than t_i , the next position is outside the cell, and the intersection of the half-line and the face becomes an exit point. If not, the position is inside the cell or on a face of the cell. For the second process, we must develop an efficient method of interpolation, because it is generally performed at many positions in the cell. Here, we employ the strategy of linear interpolation by using the data values at the two ends of a line segment as in the linear sampling method. Consequently, the velocity vector ($\vec{v}_{i+1}$) at the $(i+1)$ th point is calculated by linearly interpolating the velocity vectors at the i th point and the exit point in the ratio of Δt to $(t_i - \Delta t)$ as (see Figure 5.5)

$$\vec{v}_{i+1} = \frac{\vec{v}_i \times (t_i - \Delta t) + \vec{v}_{out} \times \Delta t}{t_i}. \quad (5.8)$$

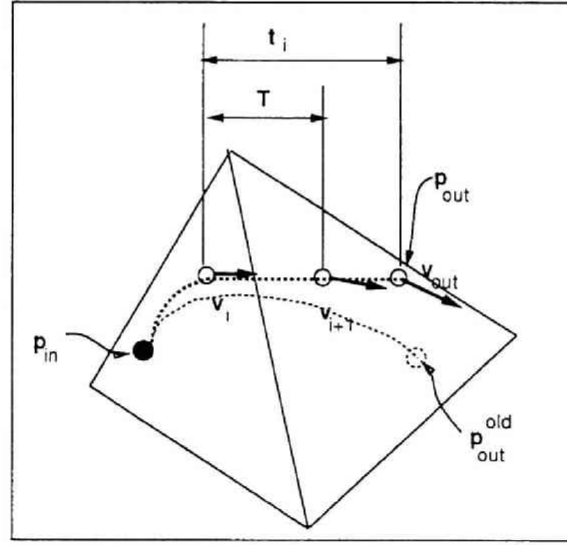


Figure 5.5: Calculation of a velocity vector in a cell

5.4 Performance evaluation

To evaluate the efficiency of the proposed algorithms, we performed tests with the following features:

- An irregular volume dataset consisting of 60,680 cells was used.

- For data probing, we moved a probing point on a basis of a unit translation vector $\vec{d}^6$ by pushing a mouse button.
- Pushing a mouse button corresponds to a unit translation vector.
- A stream line was spawned off after a probing point had been fixed by probing operations.

We divide the total cost (T_p) of the probing algorithm into two parts. The first, t_{pc} , is the cost of calculating the intersections of a traverse line and a cell face. The second, t_m , is the cost of displaying the coordinates and the data values each time a mouse button is pushed. We also divide the total cost (T_s) of the stream line algorithm into two parts. The first, t_{sc} , is the cost of the overhead in each cell. The second, t_i , is the cost of interpolating a velocity vector at each iteration. Consequently, the total cost functions (T_p, T_s) are estimated as

$$T_p = N_{pc} \times t_{pc} + N_m \times t_m \quad (5.9)$$

$$T_s = N_{sc} \times t_{sc} + N_i \times t_i, \quad (5.10)$$

where

- N_{pc} is the total number of times that a traverse line visits cells.
- N_m is the total number of times that a mouse button is pushed.
- N_{sc} is the total number of cells through which a stream line passes through.
- N_i is the total number of iterations in the calculation of velocity vectors.

We measured the CPU time (T_p, T_s) in milliseconds and the parameters (N_{pc}, N_m, N_{sc}, N_i). Table 5.1 shows the statistics on the results. From the values of T_p and T_s , the cost components of the total cost function are calculated as

$$t_{pc} = 2.43 \times 10^{-1} \quad (5.11)$$

$$t_m = 2.89 \quad (5.12)$$

$$t_{sc} = 3.80 \times 10^{-2} \quad (5.13)$$

$$t_i = 6.10 \times 10^{-2} \quad (5.14)$$

⁶The components of the vector were $(\pm\Delta x, 0, 0)$, $(0, \pm\Delta y, 0)$, or $(0, 0, pm\Delta z)$, where Δx is defined as $\frac{1}{20}$ of the volume extent in the x direction, and Δy and Δz are defined similarly.

Table 5.1: Statistics on tests for the data probing technique and the stream line display technique

Test no.	no. 1	no. 2	no. 3	no. 4	no. 5
N_{pc}	51	36	77	28	65
N_m	10	7	20	10	10
T_p (msec)	41.7	30.2	77.6	38.5	44.9
N_{sc}	9	9	79	88	32
N_i	333	262	2307	2803	994
T_s (msec)	20.5	16.2	141.8	170.7	60.9

The validity of these costs can be confirmed by using other datasets. It is apparent that these cost components are independent of the number of cells (N). However, the number of times that a traverse line visits cells on a basis of a single translation vector ($\frac{N_{pc}}{N_m}$) and the number of times that a stream line passes through cells (N_{sc}) are both proportional to $N^{1/3}$. This means that the total costs, T_p and T_s , are dependent on N . In this measurement, the above two parameters ($\frac{N_{pc}}{N_m}$ and N_{sc}) are 4.51 and 43.4 on average, respectively. The number of iterations at each tetrahedral cell ($\frac{N_i}{N_{sc}}$) is dependent on the variation of velocity vectors. In this measurement, this parameter is 30.9 on average. On the assumption that we have a volume dataset whose number of cells is one thousand times that of the volume dataset used for this evaluation and whose number of iterations is the same, we can estimate the performance of the data probing technique and the stream line display technique as

$$T_p = 1000^{1/3} \times 4.51 \times t_{pc} + t_m \times 1 \quad (5.15)$$

$$= 13.8 \text{ msec} \quad (5.16)$$

$$T_s = 1000^{1/3} \times 43.4 \times (t_{sc} + 30.9 \times t_i) \quad (5.17)$$

$$= 834.5 \text{ msec.} \quad (5.18)$$

This evaluation shows that our algorithm performs well even for a large volume dataset ($N \approx 60,000,000$ cells).

5.5 Summary of this chapter

We developed two visualization techniques, a data probing technique and a stream line display technique, based on an efficient traverse of tetrahedral cells, which play the role of measurement devices for

understanding 3-D FEM results. The efficient traverse makes it possible to move freely around a given irregular volume dataset and to spawn off stream lines.

Chapter 6

System implementation and applications

Many engineering problems have recently been solved by using 3-D FEM techniques. We have been working on ways of calculating the thermal stress inside a semiconductor chip, the heat flux around the chip, and the air flow in the clean room where the semiconductor chip is manufactured, by using 3-D FEM techniques. We realized when we started that there was no visualization software for viewing these results effectively. We therefore began to develop an isosurface display program, a volume rendering program, and a stream line display program, and integrated them into a single package called the Integrated Volume visualization System (IVS) at the request of engineers who use numerical simulation technologies. In this chapter, we describe IVS and its applications.

6.1 Goal of IVS

IVS has been designed so that mechanical engineers can visualize the results of 3-D numerical simulation (FEM and FDM) to evaluate a product design, especially in the upstream design phase. There are two situations in which numerical simulation techniques are applied to design phases. One is in the upstream design phase, and the other is in the downstream design phase. This classification is based on whether or not trial manufacturing has been done. In general, numerical simulation techniques are initially installed in the downstream design phase, mainly to analyze the causes of design defects found by testing a pilot product. Since malfunctions have been discovered during the testing, the end users model only a small portion of the product. The requirement for visualization is therefore relatively low and may be satisfied by using a conventional post-processor. In the next step, the numerical simulation techniques are applied

in the upstream design phase. In this situation, a large portion or the whole portion of the product is numerically modeled, because there is no good reference document such as a test report based on a pilot product. The need for visualization becomes very high, because the product's performance must be foreseen in the numerical model. The requirements include both extensive and intensive exploration of simulation results. To meet them, IVS has two goals: to provide an overall viewer and to provide a detailed viewer.

6.1.1 Overall viewer

In the upstream design phase, IVS should be able to show the overall features of the simulation results. A volume rendering tool can meet this requirement naturally, because it produces an image of density clouds that emit light according to data values in 3-D space. The data distribution on a slice surface may be a good abstraction of the volume dataset, but it confines 3-D information to a 2-D plane. To realize an overall viewer, it is desirable that global data values should contribute to the final data representation. When a volume slice is used for the realization, it must move interactively in the direction of its normal.

6.1.2 Detailed viewer

Detailed viewing capability allows the value of the data at an arbitrary point to be interactively displayed by interpolation of simulation results defined at discrete points. It is indispensable that the final evaluation of a product design should use measurements based on a real model in the downstream design phase, even if numerical simulation techniques are applied in the upstream design phase. In this respect, numerical simulation technique should be useful in preparing an effective plan for the real measurement. With this capability, engineers could decide where and what kinds of data values should be measured in the downstream design phase. Thus it would be good if engineers could virtually measure the data values in the same way as they do in their laboratory. To realize such a capability, IVS provides a data probing tool followed by visualization tools.

6.2 User interface of IVS

6.2.1 Overall viewer

In the overall viewer, end users can invoke visualization tools such as volume rendering, isosurface, volume slice, and stream line display tools, by selecting from menus (see Figure 6.2.1). The menus are designed

according to the following considerations with respect to the user's context:

- **Which parts of volume cells are used for visualization?**

In an IVS grouping menu, engineers can specify a set of volume cells that they wish to activate for their visualization. For example, this grouping function will be useful when

- Exterior faces of solid regions are visualized in a model where both solid cells and fluid cells exist.
- Isothermal surfaces are visualized only in solid volume cells.

IVS provides a method for grouping volume cells by specifying the maximum and minimum volume cell identifiers and specifying a volume cell attribute such as fluid, void, or solid.

- **Which function data are used for visualization?**

In general, even a single volume dataset contains a variety of independent variables, such as temperature, pressure, and velocity vector. A selection box is provided to allow a variable name to be specified. IVS makes useful integrated visualization possible by providing space for storing two scalar variable data and one vector variable data item. For example, it can display isothermal surfaces that are colored to represent pressure data, and can combine stream lines on them.

- **How are the data visualized?**

To visualize data values, IVS visualization tools map them onto some adequate geometry such as a slice surface or an isosurface. Each visualization tool needs the following parameters:

- Geometry parameter
- Color parameter
- Seed parameter.

The first parameter specifies what geometric primitive is used for the display. For volume slice display, isolines or a shaded contour face can be selected by making a trade-off between beauty and efficiency. The second parameter specifies how the primitives are colored. Either constant color or color mapping can be selected. The third parameter specifies a seed for generating geometries that

is related to each visualization tool. For example, in an isosurface tool, the seed is the value of an isosurface.

Each tool can be invoked and can generate related geometries after these parameters have been set. If no parameters are explicitly specified, IVS uses default values. The visualization tool can be executed interactively, since engineers can change the third parameter by using a mouse device.

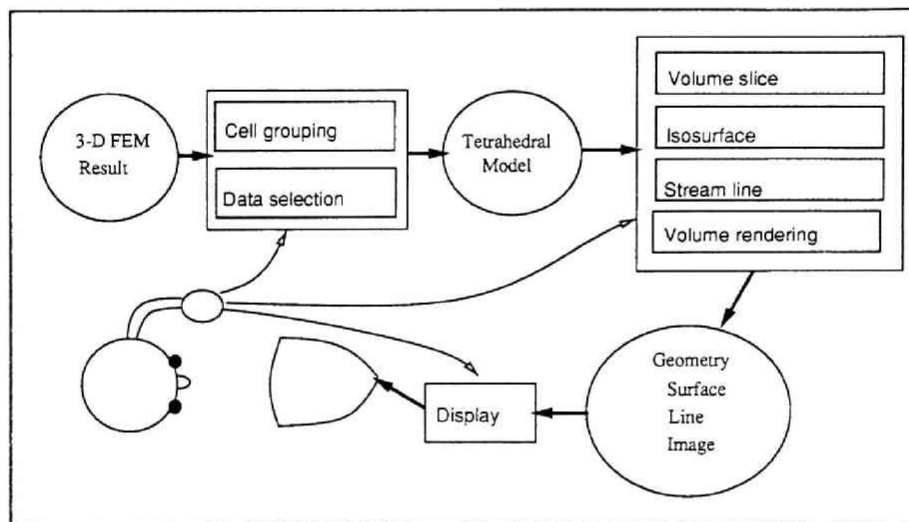


Figure 6.1: User interface of IVS overall viewer

6.2.2 Detailed viewer

Unlike the overall viewer, the detailed viewer has no menus. It offers a “virtual laboratory” so that end users can “measure” the data values of a given volume dataset. This viewer’s features are as follows:

- To specify an arbitrary position (a probing position) in 3-D space by using the peripheral devices with which engineering workstations (EWS) are normally equipped

Currently, various kinds of device have been developed for specifying spatial locations, and are utilized to implement ‘Virtual Reality’ systems. However, end-users cannot readily purchase these devices, since they are still costly. Therefore, we decided to use a mouse, which is a standard EWS device, for specifying a probing position. For effective specification, the following three modes are defined:

- Direct input of coordinates (point mode)

- Restriction of the probing position to a user-specified line (line mode)
- Restriction of the probing position to a user-specified plane (plane mode).

The probing point is always inside the 3-D region of a given volume dataset. If it reaches the boundary of the region, it stops moving. This simulates a real situation in data measurement. For example, when we measure temperature values in a room, we cannot move the measurement device through the wall of the room.

- **To interactively display (scalar or vector) data values evaluated at a user-specified point**

The data values must always be shown at the probing point. In a real situation, measured physical quantities are dynamically indicated once the measurement device has been turned on. In the original volume dataset, data values are defined at the dispersed points (node points). On the other hand, the probing point may not correspond to one of these node points. Efficient evaluation of the data values at an arbitrary point is indispensable.

- **To interactively display a stream line, a volume slice, or an isosurface according to the movement of the probing point**

A stream line, a volume slice, or an isosurface that passes through a probing point is generated by a simple action (pushing one of the buttons on the mouse device). When a stream line, a volume slice, or an isosurface is displayed, it is continuously re-generated according to the movement of the probing point. Interactive generation makes dragging possible. When a flow field is observed in a room, smoke is used as a tracer. In the observation, it is important to see how a change in the source position of the smoke (e.g. cigarette smoke) influences the resulting trajectory. A laser light sheet method is often used to illuminate particles of the smoke and show their distribution on a slice plane. In a real situation, establishment and modification of the plane are elaborate tasks.

6.3 Stereoscopic display capability

When 3-D simulation results are displayed, especially by using the capabilities of the IVS overall viewer, such as volume rendering and isosurface display, much information tends to be overlapped in the depth direction. In this case, it is very difficult to understand which information is at the front and which is at the

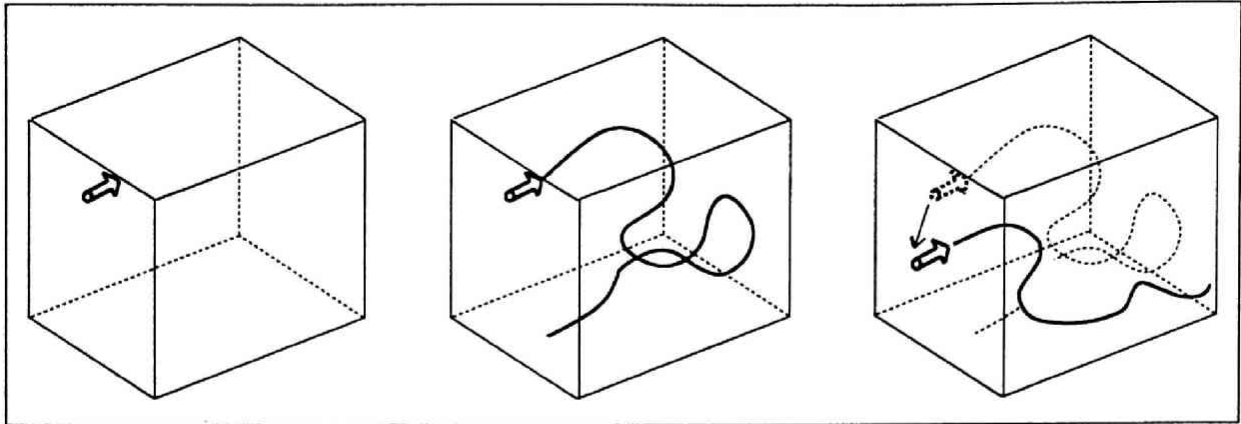


Figure 6.2: User interface of IVS detailed viewer

back. Stereoscopic display is a promising solution to this problem. Moreover, when an engineer specifies a point in the IVS detailed viewer, it is very easy to recognize its location in 3-D space. For interactive visualization, 120-Hz display systems are used in IVS. In the systems, two vertically compressed images (a right-left pair) are displayed in the upper and lower parts of the display (see Figure 6.4.1).

6.4 Application of IVS to engineering problems

6.4.1 Design of semiconductor chips

Realization of a high packaging density in a substrate on which semiconductor chips are located has become a very important issue in the development of miniaturized computer systems such as notebook personal computers. As a trade-off for this high density, many engineering problems must be considered. Some of these problems have been solved by the use of numerical simulation techniques. In this section, we describe two applications of IVS, thermal stress calculation and thermal performance verification.

Thermal stress visualization

To increase the packaging density, semiconductor chips are often attached directly to a substrate through solder bumps. In this case, the solder bumps can cause mechanical failure, although they are encapsulated between the chip and the substrate with epoxy resin in order to guard against this eventuality. Fractures may result from the thermal stress that is introduced by the difference between the coefficient of thermal expansion of the chip and that of the substrate. Engineers try to disperse the stress by optimizing the

locations of the solder bumps on the substrate, the material for the substrate, and the material for the encapsulation. 3-D FEM plays an important role, because it is very difficult to measure the stress in a real semiconductor chip.

For simplicity, only one chip is treated as our finite element model; in fact the model can be actually described as a quarter of a chip, because the location of the solder joints is symmetrical (see Figure 6.4).

Heat sources have been placed under the surface of the chip. Convection surfaces have been imposed on top of the chip and the card, and under the card. The side surfaces are assumed to be adiabatic. The solder joint has material properties that are highly dependent on temperature and time. Therefore, in the FEM calculation, the material properties of the solder joint are assumed to be plastic and dependent on temperature. Those of the other portion are assumed to be elastic and dependent on temperature.

The thermal stress calculation generates stress tensor data in each volume cell. Equivalent stress values (scalar values) rather than tensor data values are required, because they are often used to evaluate the mechanical strength. Each tensor is converted into an equivalent stress value at each volume cell in the IVS environment. Because engineers wish to understand the stress distribution in the encapsulation surrounding solder bumps, the contour map on exterior faces does not provide sufficient information (see Figure 6.5). Figure 6.6 shows iso-stress surfaces obtained by IVS. Note particularly that there is an isolated region with a high stress value in the upper left corner. Naturally, it can be expected that there will be a high possibility of a fracture at the solder joint nearest to this region. This expectation is justified qualitatively by experimental results.

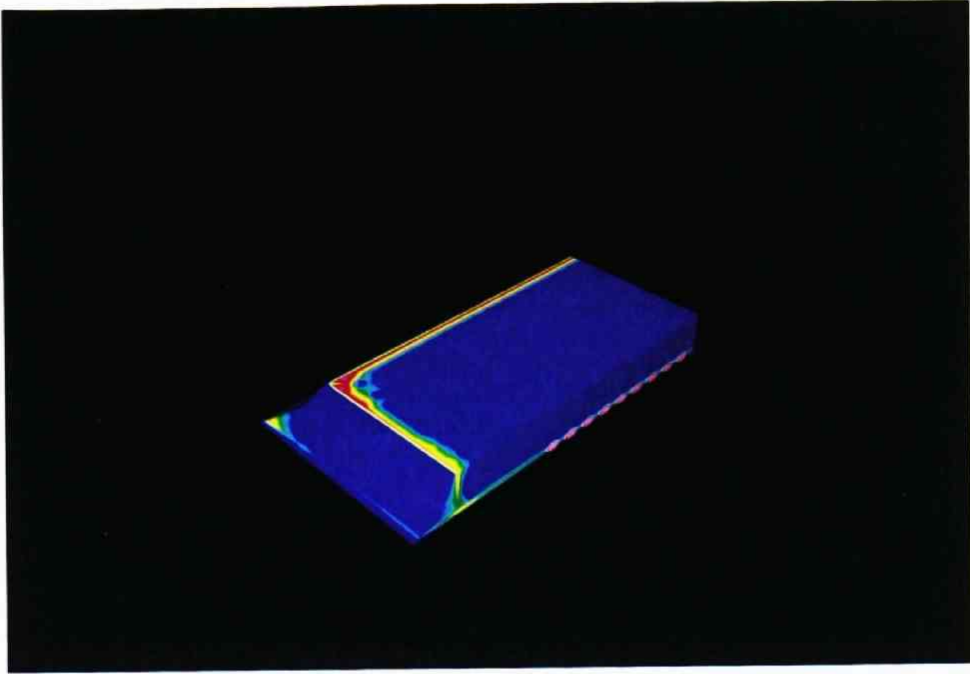


Figure 6.5: Thermal stress contour on the exterior faces

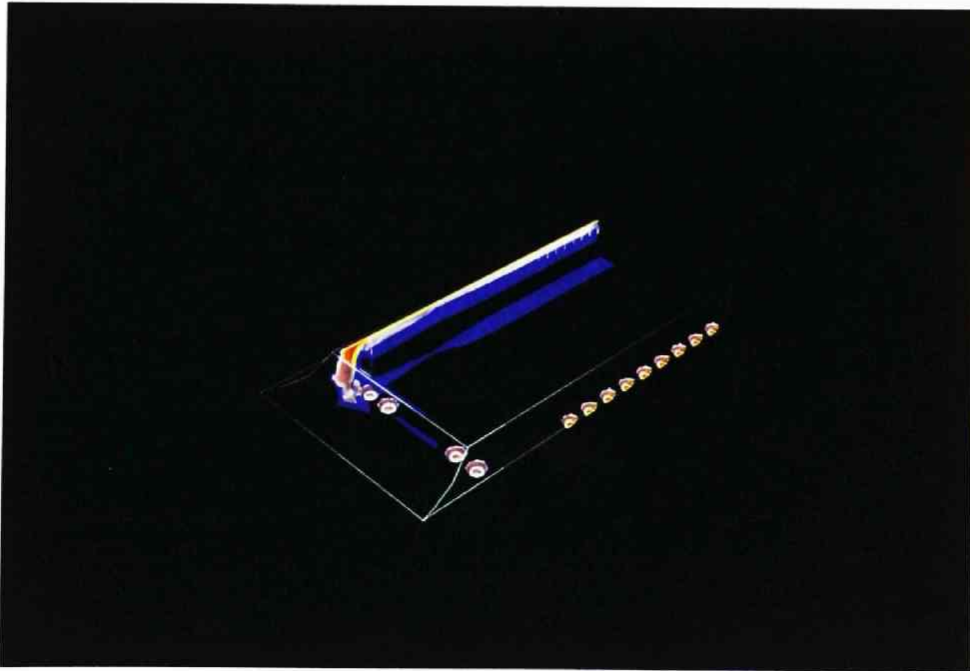


Figure 6.6: Iso-stress surfaces within the chip

Thermal performance verification

The trend toward high-density packaging also makes the cooling of electrical equipment an active area of heat transfer investigation. This cooling is motivated by reliability considerations, by the desire to decrease the transmission time of electric signals, and by the advent of new computer configurations and devices. The reliability of the packaging depends on the control of the temperature level of critical components, while the transmission time is diminished by shorter transmission distances, giving rise to closely positioned components and high densities of dissipated electrical power.

An enormous variety of specific geometries are encountered in the cooling of electronic equipment. Furthermore, the flow passages are frequently of irregular shape, often being bounded by components of various sizes and shapes. As a consequence, the approach to cooling electronic equipment has included elements of art as well as science. In particular, owing to the diversity and irregularity of flow passages, there have been few fundamental research studies on electronic equipment cooling.

Despite the above-mentioned complexities in the arrangement of electronic components and the diversity in the flow passages, technological progress in 3-D FEM has made it possible to identify generic cooling problems and related flow passage configurations. These generic problems invite systematic research, with the promise that broadly applicable results will be forthcoming.

The work to be presented here focuses on one of the most commonly recurring generic configurations in electronic cooling. This is the case of forced convection air cooling of an array of nine rectangular heat-generating modules deployed along one wall of a flat rectangular duct. Figures 6.7, 6.8 and 6.9 show the isothermal surfaces in the fluid region, the thermal distribution outside the modules, and the isothermal surfaces in the heat sink attached to a module, respectively. These isothermal surfaces are very useful for conveying the overall features of the results, especially when they are displayed by changing the thermal value interactively. On the basis of these results, some modules are removed and barriers that protrude above the plane of the modules are installed in order to increase the heat transfer coefficient in modules that have particularly high heat dissipation and/or a crucial temperature limitation.

6.4.2 Control of air flow in a clean room

Clean rooms are extensively used in manufacturing lines for computer components such as memories and disks. Conventionally, engineers have tried to achieve uniformly high cleanliness within a clean room;

although the required level of cleanliness is not the same at every point. This results in a huge and unnecessary running cost. The area requiring high cleanliness should be localized, if other areas do not share this need. In a clean room of the vertical laminar flow type, which is used in computer makers' manufacturing lines, very clean air is supplied from the ceiling in order to blow particles generated in the clean room smoothly out through the return ducts.

Design of a rectifying board

Manufacturing lines for hard disks often use a small clean room in which a workbench and a worker are located at fixed positions. The area in which high cleanliness is required is above the workbench on which the worker is assembling mechanical components. One promising idea is to compel the air that in the current approach goes out uselessly through the return ducts to contribute to the achievement of local high cleanliness above the workbench. To realize this idea, we attached a rectifying board to the side of the workbench. This board is a squared plate with air-guides to magnify the force that controls the air flow. The board is installed on the opposite side of the workbench to a contamination source such as a worker. The performance of the board is evaluated by using the following parameters above the workbench:

- Flow direction
- Cleanliness.

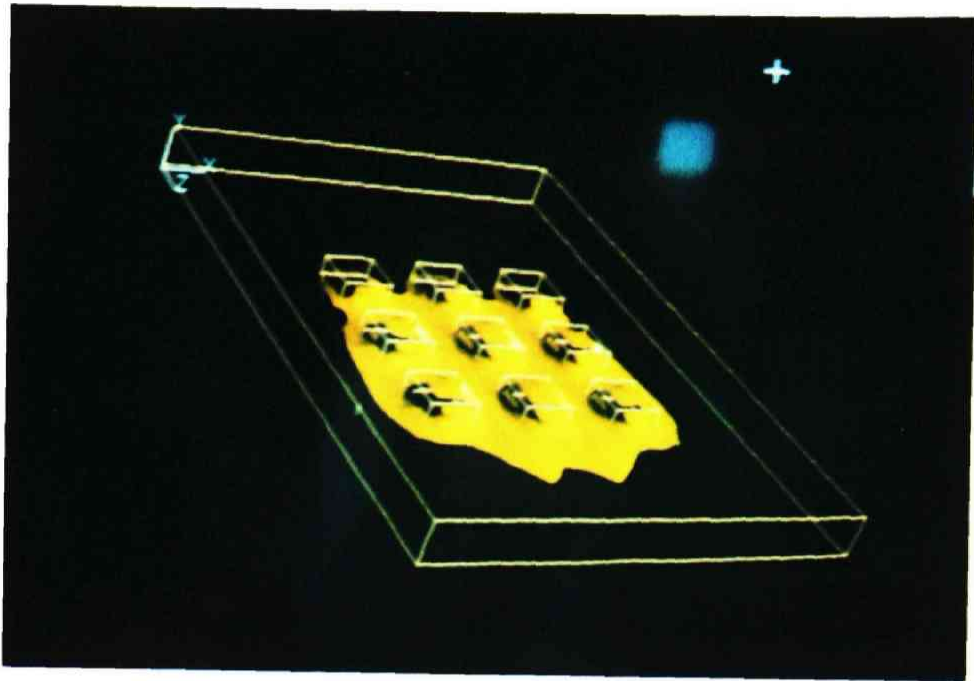


Figure 6.7: Isothermal surface in the fluid region (55,800 cells)

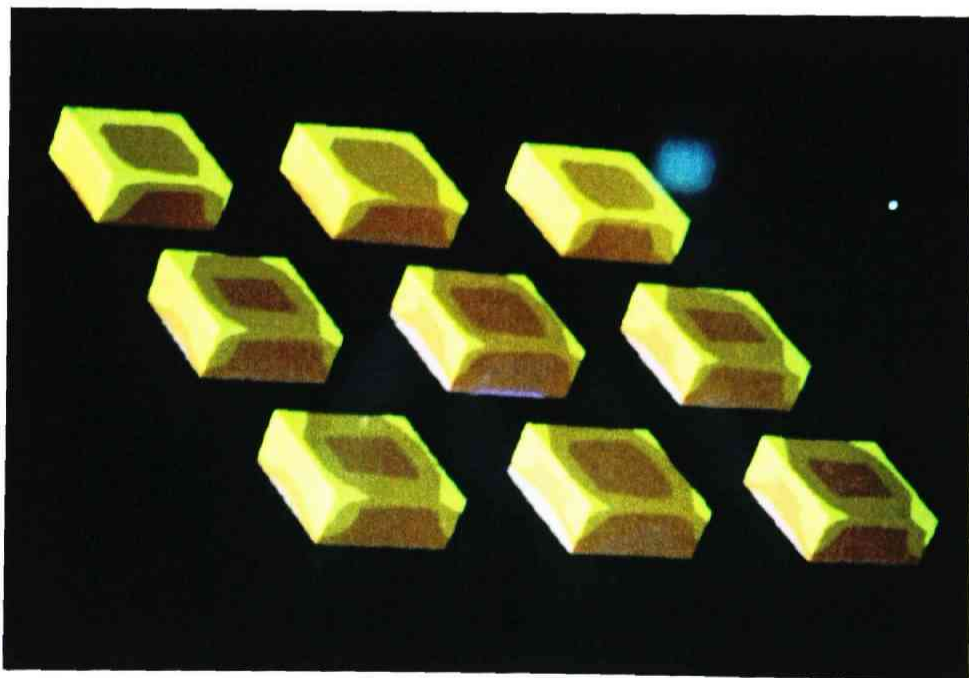


Figure 6.8: Thermal distribution outside the heat-generating modules

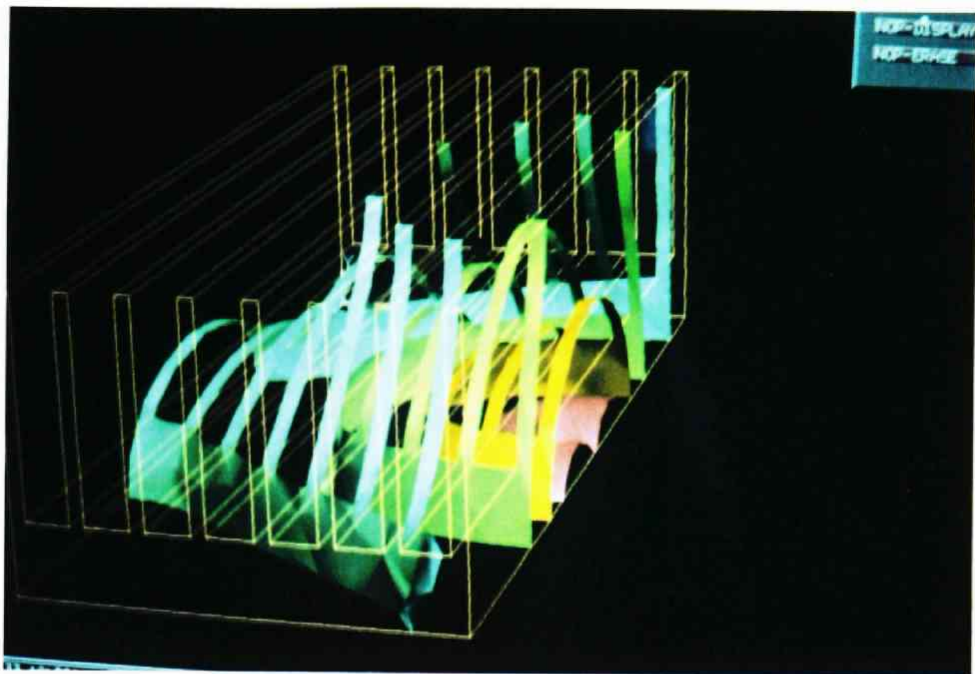


Figure 6.9: Isothermal surfaces in the heat sink (9,800 cells)

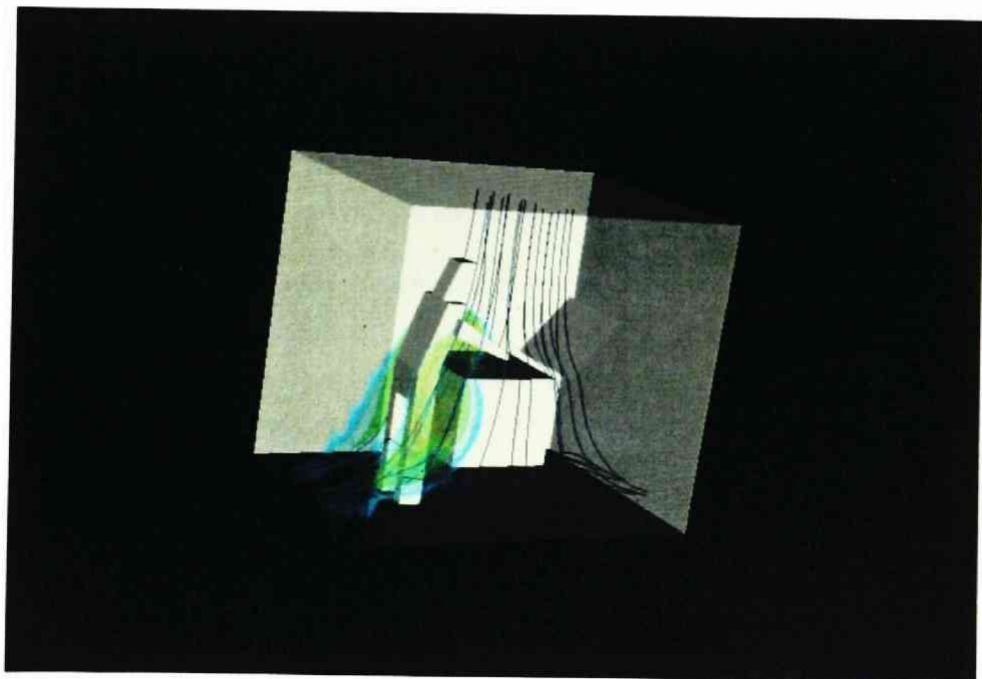


Figure 6.10: Volume rendering of particle density (61,680 cells)

The flow direction is recognized by using a tuft of cloth or smoke. There is a rule of thumb for optimizing the air flow that it should be directed from the working area toward the contaminant source (worker). In order to measure the cleanliness, we use a particle counter to monitor the number of particles that a particle generator discharges from a diffuser, which is located as a contaminant source. If no particles are registered, optimum cleanliness has been achieved. These parameters could be measured in an actual clean room. However, there are inevitable difficulties in the measurement:

- It is difficult to make the measurements in a clean room during operation.
- The measurement devices will inevitably change the flow field in a clean room.

Therefore, we obtained the parameters by numerical simulation. The air flow field can be obtained by solving Navier-Stokes equations. Particle behavior can be numerically realized by calculating the transportation equation of its density. The FEM has been employed to calculate the air flow and the particle density, because a real environment in a clean room includes irregularly shaped components.

Once the calculation is complete, we first use a volume rendering technique to understand the total distribution of the density of the particles generated from a worker, because it gives us an image similar to what we see in flow visualization. A generated image can suggest which areas should be visualized in detail (see Figure 6.10).

Then, we investigate the localized values of the density and the velocity around the board and the workbench by using the IVS's detailed viewer. For such an investigation, the user interface is important. In order to specify the seed point in a graphics display, IVS provides an probing icon composed of

- A line segment parallel to the velocity vector ($\vec{v}_s$), pseudo-colored according to the vector's magnitude, and labeled with strings showing the magnitude
- A point pseudo-colored according to the scalar data (s_s) there, such as pressure or temperature, and labeled with strings showing the values of the data.

In this localized visualization, the probing icon plays the role of a tuft and a particle counter used in an actual clean room. We specify an arbitrary spatial point effectively by using a stereoscopic display. By moving the icon interactively in the area above the workbench and spawning off stream lines, we confirm the effectiveness of the board with respect to the contaminant density above the workbench and the flow direction (see Figure 6.11).

Optimization of a length of a separation panel

In the manufacturing line for semiconductor chips that we studied, there is a large clean room, which is divided into two areas: a processing area and an operator aisle area. There is a dividing panel suspended vertically from the ceiling between the two areas. This panel does not separate the two areas completely: there is some space around the floor so that facility engineers can enter the processing area to maintain processing tools. In the processing area, a very clean air flow is supplied at high speed from the ceiling so that a high level of cleanliness is maintained. Although the speed and the cleanliness of the supplied air flow in the aisle area are lower, their levels are high relative to the cleanliness needed. To reduce the running cost, it is desirable to decrease the velocity of the air flow provided in the aisle area. With a given decrease in velocity, we optimize the length of the separation panel so that the resulting air flow field satisfies the following conditions:

- There is no air flow directed from the aisle area to the process area.
- There is no vorticity, especially near the ceiling.

When we used the conventional method, we observed the trajectory of water mist after spawning off it in order to confirm that the conditions were satisfied. Currently, numerical simulation techniques are employed for this observation. In the IVS's detailed viewer, it can be performed easily, because no interaction has any influence on the air flow field, and furthermore, the placement of a seed point from which a particle (air mist) is spawned off is more flexible. By interactively displaying stream lines according to the movement of the probing point, we can investigate a bad case in which the above conditions are not satisfied. Figure 6.12 shows such a case in which vortical flow exists near the ceiling.

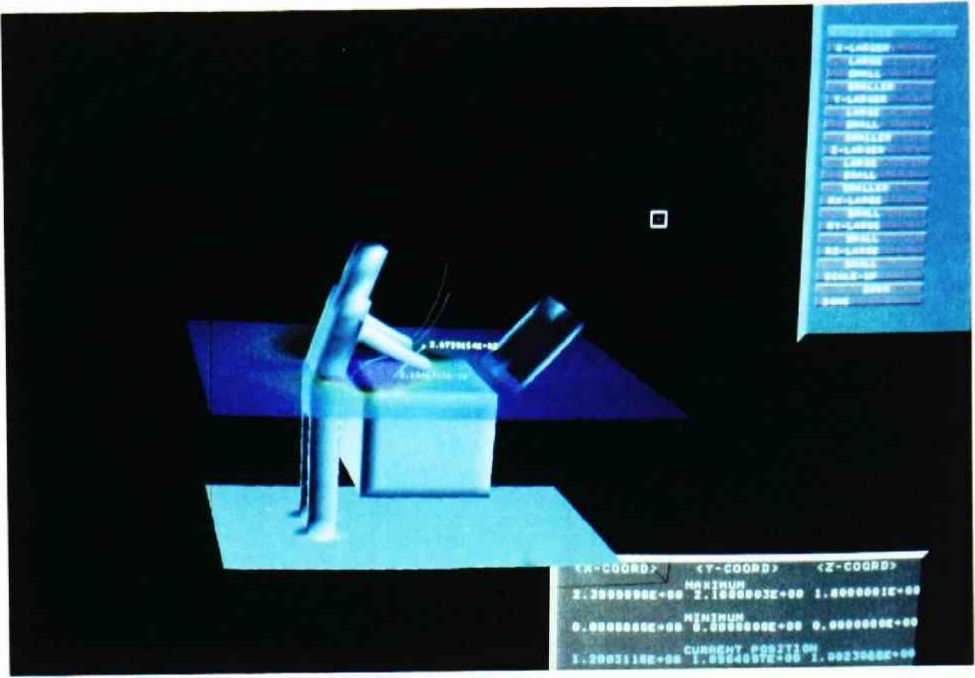


Figure 6.11: Localized visualization above the workbench

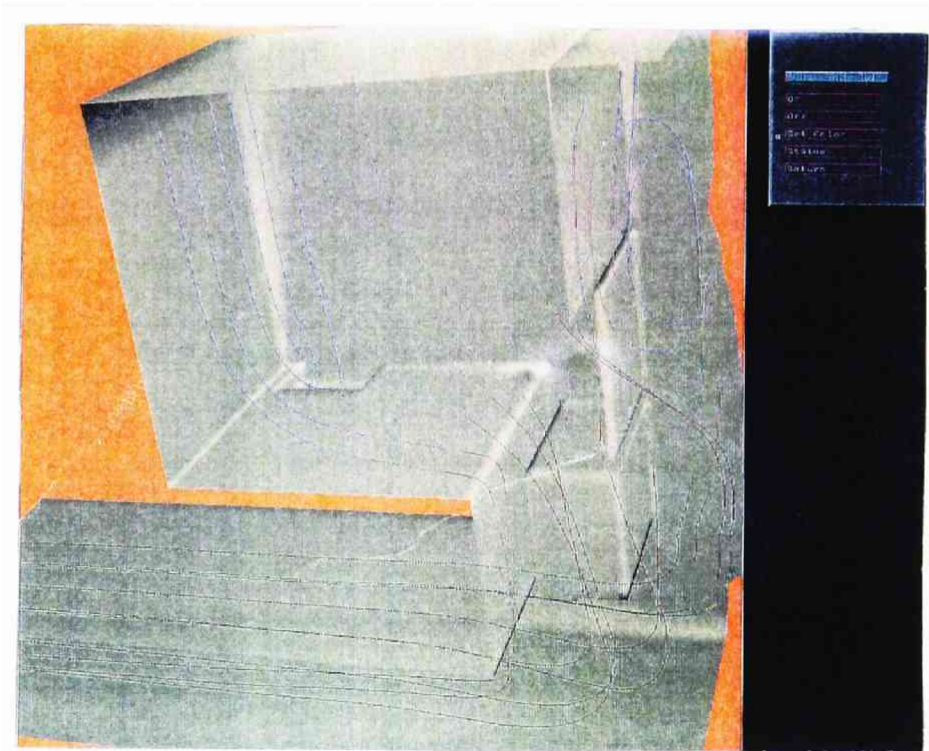


Figure 6.12: Visualization of vortical flow near the ceiling

6.4.3 Design of reciprocating engines

New fuel consumption requirements and pollution reduction regulations for automotive engines have boosted engine research and design activities, and created pressure for the use of new types of fuel, and for new optimized engine design. Combustion in engines is governed by local in-cylinder flow characteristics. The understanding, optimization, and control of the combustion process during the engine cycle depend on similar operations for the in-cylinder flow. Experimental investigations of flow motion in cylinders are costly and time-consuming, and are also incomplete. Numerical simulation technologies are complementing or replacing experimental investigations, and are able to evaluate performance at the design stage, without prototype casting and detailed measurements.

The engine used in this work is a four-valve spark-ignition four-stroke engine, with two direct inlet ports. The modeled fluid is air in ambient conditions, considered as a perfect gas. The inlet boundary condition is a specified stagnation pressure at the port pipe limit, and the cylinder outlet boundary condition is a specified static pressure. A subgrid scale eddy viscosity is used for bulk turbulence modelling, and a law of wall for the turbulent boundary layer. Figure 6.13 shows the cloudy representation of the calculated pressure field in the cylinder. The clouds make it easy for us to identify the region in which the circumferential swirls, whose evolution has an important influence on combustion, are produced. Figure 6.14 shows that the stream lines could revolve under the valve in the cylinder. The interactive point location capability provided by IVS allows us to specify the seed points in the complex region of this engine model. Moreover, the data probing capability, in which data values such as pressure and velocity are displayed on the fly, can provide a “virtual measurement” tool.



Figure 6.13: Cloudy representation of the calculated pressure fields (372,500 cells)



Figure 6.14: Identification of low-pressure region under the valve

Chapter 7

Conclusion

7.1 Summary

This dissertation has proposed a method for visualizing 3-D FEM results both globally and locally. Tetrahedral cells were used as elements in order to allow the method to efficiently interpolate complex data distribution within a volume cell. The method was applied to real engineering problems and its effectiveness was confirmed. Here, we summarize the dissertation by considering the following goals, which we outlined in Chapter 1:

1. It should use a flexible data model in which an irregular volume dataset can be handled efficiently.
2. It should include a capability for giving overall features of the results.
3. It should include a capability for interactively displaying the value of the data at an arbitrary point by interpolating a simulation result.
4. It should be applicable to real engineering problems that are solved by using 3-D FEM techniques, and its effectiveness should be confirmed.

The original 3-D FEM results were decomposed into tetrahedral cells to allow efficient visualization. In Chapter 2, it was pointed out that a list of adjacent volume cells should be generated and that each volume cell should be decomposed into a set of tetrahedral cells after the data interpolation techniques for 3-D FEM results have been discussed. Generally, techniques for interpolating a data value at an arbitrary position in a volume cell include Newton iteration, which requires a good initial value. For data interpolation in a tetrahedral cell, the calculation is straightforward. Finally, we proposed a tetrahedral

model composed of a nodal data component, a cell topology component, a cell adjacency component, and an exterior face component. The model achieves the first goal.

In Chapter 3, we described an efficient technique for displaying isosurfaces, based on the tetrahedral model proposed in Chapter 2. Once four data values have been provided at node points of a tetrahedral cell, topological information on the triangular facets generated within a tetrahedral cell can be determined without any ambiguity, unlike those generated within a cubic cell. This allows the technique to be developed on the basis of a fully table-driven approach. A previous bottleneck was that the cells that never intersected the isosurface had to be visited when a series of isosurfaces were displayed interactively. Our technique solved the problem by searching for intersected cells from a seed cell on the basis of cell adjacency information. The seed cell is selected from a list of extrema points within the volume. We confirmed the efficiency of our technique by performing a benchmark test in which it was compared with a brute-force technique.

In Chapter 4, we proposed techniques for comprehending the overall features of a 3-D FEM result as density clouds. First, we discussed an equation that describes the scattering of radiation in the clouds. Because it is very difficult to solve the equation, we then used a simplified form of it in which the cloud particles themselves emit light with no scattering for our purpose of visualization, and provided it in discrete systems. We developed techniques based on the image order approach and the object order approach. In order to relax the bottlenecks of the image order approach, in testing for intersections of a ray and an irregular volume and in integrating brightness along rays, we developed a new cell traverse method. In the object order approach, we approximate volume rendering images with a stack of concentric spheres, that is, iso-distance surfaces. To generate a series of these surfaces efficiently, we employed the extrema-based marching tetrahedra technique proposed in Chapter 3. Finally, we used the results of benchmark tests in the two approaches and discussed a criterion for their proper use with respect to performance. The techniques outlined in Chapters 3 and 4 achieve the second goal.

In Chapter 5, we focused on two techniques for displaying more localized information from a given irregular volume data set: a data probing technique and a stream line display technique. A bottleneck of these techniques was that it takes considerable CPU time to find a cell that contains a given point (the spatial point location problem). We developed an efficient point location technique that assumes the existence of a starting point in any cell. It traverses cells along a line that is a function of a

parameter t , where $t=0$ is the starting point and $t=1$ is the target. Data interpolation in the target cell is straightforward, because the data distribution is linear along a line segment within a tetrahedral cell. The idea of the cell traverse was applied to a technique for displaying stream lines on the basis of this idea. These techniques achieve the third goal.

In the final chapter, we described the Integrated Volume visualization System (IVS), in which the techniques proposed in the previous chapters are integrated into a single package, and considered the application of IVS to engineering problems. First, we stated the two major goals of IVS to provide overall viewers and detailed viewers and described its user-interface.

Then, we applied IVS to the design of semiconductor chips, the control of air flow in a clean room, and the design of reciprocating engines. As a trade-off for high packaging density in a substrate, many engineering problems must be considered, by using numerical simulation techniques. We used an IVS isosurface display technique for thermal stress visualization within a semiconductor chip and for thermal performance verification in a heat-generating module and in a heat sink. In order to maintain high cleanliness locally by changing the layout of the room instead of increasing the ventilation, numerical simulation technologies are often employed for this control. To explore the results, we used as IVS visualization techniques a data probing technique, a stream line display technique, and a volume rendering technique, which are related to measurement techniques that have been used in actual clean rooms. IVS volume rendering techniques were also used to visualize the pressure distribution in the calculated in-cylinder flow of reciprocating engines. The density clouds showed the regions that give rise to circumferential swirls, whose evolution has an important influence on combustion.

7.2 Plan for extending this study

On the basis of the research presented in this dissertation, the following extensions can be considered:

1. Most actual physical phenomena have unsteady characteristics. The first obvious extension to this work is therefore to adapt it for unsteady numerical simulation results.
2. A basic strategy needed to enable our technique to cope with p-type FEM results is described in Chapter 2. The second extension is to implement a tetrahedrization technique for high-order volume cells.

3. We proposed seed-based visualization techniques: extrema-based marching tetrahedra and data probing based on cell traverse. Virtual reality technologies are welcome in that they allow users to specify a seed cell more easily.
4. A possible and useful extension of volume rendering is opacity animation in an interactive environment. Once concentric spheres have been generated in our splatting technique (Method-2) and stored in a temporary memory space, an interactive modification of the opacity lookup table changes the resulting image dramatically. The realization of this capability requires high-speed image compositing and a large memory space.
5. Steering of on-going numerical simulation is becoming a trend in the scientific visualization community. Volume rendering, or isosurface display, in residual data values of the simulation process is a promising application of IVS. After exploration of a region in which re-meshing operations are required, an interactive session for mesh generation will open and users can specify the positions in a virtual reality environment as if they were in a "jungle gym."

Bibliography

- [Artz81] E Artzy, G Frieder, and G T Herman. "The Theory, Design, Implementation and Evaluation of a Three-Dimensional Surface Detection Algorithm," *Computer Graphics and Image Processing*, Vol. 15, No. 1, pp. 1-24, 1981.
- [Blin82] J F Blinn, "Light Reflection functions for Simulation of Clouds and Dusty Surfaces." *Computer Graphics*, Vol. 16, No. 3, pp. 21-29.
- [Brys92] S Bryson and C Levit, "The Virtual Wind Tunnel," *IEEE Computer Graphics and Applications*, Vol. 12, No. 4, pp. 25-34, 1992.
- [Bruc92] L Bruce, "A Scientific Visualization Renderer," *IEEE Visualization '92*, pp. 227-223, 1992.
- [Buni88] P G Buning, "Sources of Error in the Graphical Analysis of CFD Results," *J. Sci. Comp.*, Vol. 3, No. 2, 1988.
- [Chan91] L W Chang, H W Chen, and J R Ho. "Reconstruction of 3D Medical Images: A Nonlinear Interpolation Technique for Reconstruction of 3D Medical Images." *CVGIP: Graphical Models and Image Processing*, Vol. 53, No. 4, pp. 382-391, 1991.
- [Chen85] L S Chen, G T Herman, R A Reynolds, and J K Udupa, "Surface Shading in the Cuberille Environment," *IEEE Computer Graphics and Applications*, Vol. 5, No. 12, pp. 33-43, 1985.
- [Chon90] M S Chong, A E Perry, and B J Cantwell. "A General Classification of Three-Dimensional Flow Fields," *Physics of Fluids A*, Vol. 2, No. 5, pp. 765-777, 1990.
- [Chri78] H N Christiansen and T W Sederberg. "Conversion of Complex Contour Line Definitions into Polygonal Element Mosaics." *Computer Graphics*, Vol. 12, No. 2, pp. 187-192, 1978.

- [Clea88] J G Cleary and G Wyvill, "Analysis of an Algorithm for Fast Ray Tracing Using Uniform Space Subdivision," *The Visual Computer*, Vol. 4, No. 2, pp. 65-83, 1988.
- [Clin88] H E Cline, W E Lorensen, S Ludke, C R Crawford, and B C Teeter, "Two Algorithms for the Three-Dimensional Reconstruction of Tomograms", *Medical Physics*, Vol. 15, No. 3, pp. 320-327.
- [Clin90] H E Cline, S Ludke, W E Lorensen, and B C Teeter, "A 3D Medical Imaging Research Workstation," *Volume Visualization Algorithms and Architectures, ACM SIGGRAPH '90 Course Notes*, Course Number 11, ACM Press, pp. 243-255, 1990.
- [Dayh63] M O Dayhoff, "A Contour Map Program from X-Ray Crystallography," *Communications of the ACM*, Vol. 6, No. 10, pp. 620-622.
- [Delm92] T Delmarcelle and L Hesselink, "Visualization of Second-Order Tensor Fields and Matrix Data," *IEEE Visualization '92*, pp. 316-323, 1992.
- [Dick91a] R R Dickinson, "Interactive Analysis of the Topology of 4D Vector Fields," *IBM Journal of Research and Development*, Vol. 35, No. 1/2, pp. 59-66.
- [Dick91b] R R Dickinson, "Interactive Analysis of Transient Field Data," *SPIE Proceedings*, Vol. 1083, pp. 173-180, 1989.
- [Dreb88] R Drebin, L Carpenter, and P Hanrahan, "Volume Rendering," *Computer Graphics*, Vol. 22, No. 4, pp. 65-74, 1988.
- [Durs88] M J Durst, "Letters: Additional Reference to Marching Cubes," *Computer Graphics*, Vol. 22, No. 2, pp. 72-73, 1988.
- [Eber90] D S Ebert and R E Parent, "Rendering and Animation of Gaseous Phenomena by Combining Fast Volume and Scanline A-Buffer Techniques," *Computer Graphics*, Vol. 24, No. 4, pp. 357-366, 1990.
- [Eiho84] S Eiho, "Reconstruction of the Left Ventricle from X-Ray Cine-Angiocardiograms with a Rotating Arm," *Computers in Cardiology 1983*, pp. 63-67, 1984.

- [Ekou91] A B Ekoule, F C Peyrin, and C L Odet, "A Triangulation Algorithm from Arbitrary Shaped Multiple Planar Contours," *ACM Transaction on Graphics*, Vol. 10, No. 2. pp. 182-199, 1991.
- [Elia89] P Eliasson, J Oppelstrup, and A Rizzi, "STREAM 3-D: Computer Graphic Program for Streamline Visualization," *Advances in Engineering Software*, Vol. 11, No. 4, 1989.
- [Forr69] A R Forrest, "Coordinates, Transformations and Visualization techniques," Document 23, University of Cambridge CAD Group, 1969.
- [Fuch77] H Fuchs, Z M Kedem, and S P Uzelton, "Optimal Surface Reconstruction from Planar Contours," *Communications of the ACM*, Vol. 20, No. 10, pp. 693-702, 1977.
- [Fuji85] A Fujimoto and I Kansei, "Accelerated Ray Tracing," *Proceedings of Computer Graphics Tokyo '85*. 1985.
- [Fuji86] A Fujimoto, T Tanaka, and K Iwata. "ARTS: Accelerated Ray-Tracing System." *IEEE Computer Graphics and Applications*, Vol. 6, No. 4, pp. 16-26. 1986.
- [Gall89] R S Gallagher and J C Nagtegaal, "An Efficient 3D Visualization Technique for Finite Element Models and Other Coarse Volumes," *Computer Graphics*. Vol. 23, No. 3. 1989.
- [Gall91] R S Gallagher, "Span Filtering: An Optimization Scheme For Volume Visualization of a Large Finite Element Model," *IEEE Visualization '91*, pp. 68-75, 1991.
- [Garr90] M P Garrity, "Raytracing Irregular Volume Data," *Computer Graphics*. Vol. 24, No. 5. pp. 35-40. 1990.
- [Gile90] M Giles and R Haimes, "Advanced Interactive Visualization for CFD," *Computing Systems in Engineering*. Vol. 1, No. 1, pp. 51-62, 1990.
- [Gier90] C Giertsen, A Halvorsen, and P R Flood. "Graph-Directed Modelling from Serial Sections." *Visual Computer*, Vol. 6, No. 5, pp. 284-290, 1990.
- [Glas84] A S Glassner, "Space Subdivision for Fast Ray Tracing," *IEEE Computer Graphics and Applications*. Vol. 4, No. 1, pp. 15-22. 1984.

- [Glob91] A C Globus and T Lasinski, "A Tool for Visualizing the Topology of Three Dimensional Vector Fields," *IEEE Visualization '91*, pp. 33-40, 1991.
- [Good89] D S Goodsell, S Mian, and A J Olson, "Rendering of Volumetric Data in Molecular Systems," *Journal of Molecular Graphics*, Vol. 7, No. 1, pp. 41-47, 1989.
- [Habe90] R B Haber, "Visualization Techniques for Engineering Mechanics," *Computing Systems in Engineering*, Vol. 1, No. 1, pp. 37-50, 1990.
- [Haim91a] R Haimes, M Giles, and D Darmofal, "Visual3 A Software Environment for Flow Visualization," *VKI Lecture Series on Computer Graphics and Flow Visualization on CFD*, 1991.
- [Haim91b] R Haimes and D Darmofal, "Visualization in Computational Fluid Dynamics: A Case Study," *IEEE Visualization '91*, pp. 392-397, 1991.
- [Hall86] E Halley, "An Historical Account of the Trade Winds, and Monsoons, Observable in the Seas between and near the Tropicks; with an Attempt to Assign the Phisical Cause of Said Winds," *Philosophical Transactions*, Vol. 183, pp. 153-168, 1686.
- [Hanr90] P Hanrahan, "Three-Pass Affine Transformation for Volume Rendering," *Computer Graphics*, Vol. 24, No. 5, pp. 71-78, 1990.
- [Herm79] G T Herman and H K Liu, "Three Dimensional Display of Human Organs from Computed Tomograms," *Computer Graphics and Image Processing*, Vol. 9, No. 1, pp. 1-21, 1979.
- [Heap69] B R Heap and M G Pink, "Three Contouring Algorithms," *DNAM Report 81*, National Physical Laboratory, Teddington, UK, 1969.
- [Helm89] J Helman and L Hesselink, "Representation and Display of Vector Field Topology in Fluid Flow Data Sets," *IEEE Computer*, Vol. 22, No. 8, pp. 27-36, 1989.
- [Helm90] J Helman and L Hesselink, "Surface Representations of Two- and Three-Dimensional Fluid Flow Topology," *IEEE Visualization '90*, pp. 6-13.
- [Helm91] J Helman and L Hesselink, "Visualizing Vector Field Topology in Fluid Flows," *IEEE Computer Graphics and Applications*, Vol. 11, No. 3, pp. 36-46.

- [Hess87] L Hesselink and J Helman, "Evaluation of Flow Topology from Numerical Data." Invited AIAA-paper, 87-1181-CP, 1987.
- [Hess88] L Hesselink, J Helman, and K Wu, "Visualization and Interpretation of 3-D Scientific Data Sets," ICALEO '88 Conference, 1988.
- [Hols70] P H Holst, "A Theoretical and Experimental Investigation of Natural Convection in a Porous Medium," Ph.D. Thesis, Univ. of Calgary, 1970.
- [John89] E R Johnson and C E Mosher, "Integration of Volume Rendering and Geometric Graphics." State of the Art in Data Visualization, SIGGRAPH '89 tutorial, 1989.
- [Kaji84] J T Kajiya and B P Herzen, "Ray Tracing Volume Densities," Computer Graphics, Vol. 18, No. 3, pp. 165-174.
- [Kane86] K Kaneda, K Harada, Y Sato, and E Nakamae, "A Method of Visualizing Three-Dimensional Vector-Valued Data," Transactions of ISPJ, Vol. 27, No. 6, pp. 567-574, 1986.
- [Kauf87a] A Kaufman, "An Algorithm for 3D Scan Conversion of Polygons." Proceedings of Eurographics '87, pp. 197-208, 1987.
- [Kauf87b] A Kaufman, "Efficient Algorithms for 3D Scan-Conversion of Parametric Curves, Surfaces, and Volumes," Computer Graphics, Vol. 21, No. 4, pp. 171-179, 1987.
- [Kauf90] A Kaufman, R Yagel, and D Cohen, "Intermixing Surface and Volume Rendering," 3D Imaging in Medicine, NATO ASI Series, Vol. F60, pp. 217-227, 1990.
- [Kepp75] E Keppel, "Approximating Complex Surfaces by Triangulation of Contours," IBM Journal of Research and Development, Vol. 19, No. 1, pp. 2-11, 1975.
- [Kerl90] G D Kerlick, "Moving Iconic Objects in Scientific Visualization," IEEE Visualization '90, pp. 124-130, 1990.
- [Klas91] R V Klassen and S J Harrington, "Shadowed Hedgehogs: A Technique for Visualizing 2D Slices of 3D Vector Fields," IEEE Visualization '91, pp. 148-153, 1991.

- [Koid86] A Koide, A Doi, and K Kajioka, "Polyhedral Approximation Approach to Molecular Orbital Graphics," *Journal of Molecular Graphics*, Vol. 4, No. 3, pp. 149-155, 1986.
- [Koya88] K Koyamada, "Visualization of Iso-value Surfaces and Stream Lines," 1988 IDEAS/CAEDS User's Conference, pp. 87-98, 1988.
- [Koya90] K Koyamada, "Volume Visualization for the Unstructured Grid Data," *SPIE/SPSE Symposium on Electronic Imaging Conference Proceedings*, Vol. 1259, pp. 14-25, 1990.
- [Koya91a] K Koyamada and T Miyazawa, "Volume Rendering Method for Finite Element Results Air-Flow Visualization in a Clean Room," *Trans. of IPSJ*, Vol. 32, No. 5, pp. 560-569, 1991.
- [Koya91b] K Koyamada and T Nishio, "Volume Visualization of 3-D FEM Results," *IBM Journal of Research and Development*, Vol. 35, No. 1, pp. 12-25, 1991.
- [Koya92a] K Koyamada, "Fast Traverse of Irregular Volumes," *CG International '92 Proceedings*, pp. 295-312.
- [Koya92b] K Koyamada, S Uno, A Doi, and T Miyazawa, "Fast Volume Rendering by Polygonal Approximation," *Journal of Information Processing*, Vol. 15, No. 4, pp. 535-544.
- [Koya92c] K Koyamada, "Visualization of Simulated Airflow in a Clean Room," *IEEE Visualization '92 Proceedings*, pp. 156-163, 1992.
- [Laur91] D Laur and P Hanrahan, "Hierarchical Splatting: A Progressive Refinement Algorithm for Volume Rendering," *Computer Graphics*, Vol. 25, No. 4, pp. 285-288, 1991.
- [Lavi87] S Lavin and R Cherveney, "Unit-Vector Density Mapping," *The Cartographic Journal*, Vol. 24, pp. 131-141, 1987.
- [Levo88] M Levoy, "Display of Surfaces from Volume Data," *IEEE Computer Graphic and Applications*, Vol. 8, No. 3, pp. 29-37, 1988.
- [Levo90a] M Levoy, "Efficient Ray Tracing of Volume Data," *ACM Transactions on Graphics*, Vol. 9, No. 3, 1990.

- [Levo90b] M Levoy, "Volume Rendering by Adaptive Refinement," *The Visual Computer*, Vol. 6, No. 1, pp. 2-7, 1990.
- [Levo90c] M Levoy, "A Hybrid Ray Tracer for Rendering Polygon and Volume Data," *IEEE Computer Graphics and Applications*, Vol. 10, No. 2, pp. 33-40, 1990.
- [Lore87] W Lorensen and H Cline, "Marching Cubes: A High-Resolution 3D Surface Construction Algorithm," *Computer Graphics*, Vol. 21, No. 4, pp. 163-169, 1987.
- [Mall76] G D Mallinson and G V Davis, "Three-Dimensional Natural Convection in a Box: A Numerical Study," *ARL Report, Struc/361*, 1976.
- [Max83] N Max, "Panel on the Simulation of Natural Phenomena," *Computer Graphics*, Vol. 17, No. 3, pp. 137-139.
- [Max90] N Max, P Hanrahan, and R Crawfis, "Area and Volume Coherence for Efficient Visualization of 3D Scalar Functions," *Computer Graphics*, Vol. 24, No. 5, pp. 27-34, 1990.
- [Miya89] T Miyazawa and K Sugimoto, "A Rendering Algorithm for Visualizing Voxel and Surface Data," *SIGCG Note of IPSJ*, 89-CG-41-2, 1989 (in Japanese).
- [Miya92] T Miyazawa and K Koyamada, "A High-Speed Integrated Renderer for Interpreting Multiple 3D Volume Data," *Journal of Visualization and Computer Animation*, Vol. 3, No. 2, pp. 65-83, 1992.
- [Murm88] E Murman and K Powell, "Trajectory Integration in Vortical Flows," *AIAA Journal*, Vol. 27, No. 7, pp. 982-984, 1988.
- [Neem90] H Neeman, "A Decomposition Algorithm For Visualizing Irregular Grids," *Computer Graphics*, Vol. 24, No. 5, pp. 49-56, 1990.
- [Newe72] M E Newell, R G Newell, and T L Sancha, "A New Approach to the Shaded Picture Problem," *Proc. ACM National Conf.*, 1972.
- [Pari89] P Parikh, R Lohner, D Gumbert, and S Pirzadeh, "Numerical Solution on a PATH-FINDER and Other Configurations Using Unstructured Grids and Finite Element Solver," *AIAA Paper 89-0362*, 1989.

- [Phai82] N V Phai, "Automatic Mesh Generation with Tetrahedron Elements," *Int.J.num.Meth.Engng*, Vol. 18, pp. 273-289, 1982.
- [Phon75] B T Phong, "Illumination for Computer-Generated Pictures," *Communications of ACM*, Vol. 18, No. 6, pp. 311-317, 1975.
- [Preparata 89] P Preparata and T Roberto, "Efficient Point Location in a Convex Spatial Cell Complex," Technical Report 89-47, Dept. of Computer Science, Brown University, 1989.
- [Purv86] D G Purvis and C Culberson, "On the Graphical Display of Molecular Electrostatic Force-Fields and Gradients of the Electron Density," *Journal of Molecular Graphics*, Vol. 4, No. 2, pp. 89-92, 1986.
- [Robe70] K V Roberts and D E Potter, "Magnetohydrodynamic Calculations," *Methods in Computational Physics*, Vol. 9, Plasma Physics, pp. 402, 1970.
- [Rubi80] S M Rubin and T Whitted, "A Three-Dimensional Representation for Fast Rendering of Complex Schemes," *Proceedings of SIGGRAPH '80*, pp. 110-116, 1980.
- [Sabe88] P Sabella, "A Rendering Algorithm for Visualizing 3D Scalar Fields," *Computer Graphics*, Vol. 22, No. 4, pp. 51-58, 1988.
- [Schr91] P Schroder and J B Salem, "Fast Rotation of Volume Data on Data Parallel Architecture," *IEEE Visualization '91*, pp. 50-57, 1991.
- [Sewe88] G E Sewell, "Plotting Contour Surfaces of a Function of Three Variables," *ACM Transactions on Mathematical Software*, Vol. 14, No. 1, pp. 33-44, 1988.
- [Shen85] D Shenton and Z Cendes, "Three-Dimensional Finite Element Mesh Generation Using Delaunay Tessellation," *IEEE Trans.*, Vol. MAG-21, pp. 2535-2538, 1985.
- [Shin91a] Y Shinagawa and T L Kunii, "The Homotopy Model: A Generalized Model for Smooth Surface Generation from Cross-Sectional Data," *Visual Computer*, Vol. 7, No. 2/3, pp. 72-86, 1991.

- [Shin91b] Y Shinagawa and T L Kunii, "Constructing a Reeb Graph Automatically from Cross Sections," *IEEE Computer Graphics and Applications*, Vol. 11, No. 6, pp. 44-51, 1991.
- [Shir90] P Shiley and A Tuchman, "A Polygonal Approximation to Direct Scalar Volume Rendering," *Computer Graphics*, Vol. 24, No. 5, pp. 63-70, 1990.
- [Slui82] M Sluiter and D Hansen, "A General Purpose Automatic Mesh Generator for Shell and Solid Elements," *Computers in Engineering*, Vol. 3, Book No. G00217, ASME, pp. 29-34, 1982.
- [Sper90] D Speray and S Kennon, "Volume Probe: Interactive Data Exploration on Arbitrary Grids," *Computer Graphics*, Vol. 24, No. 5, pp. 5-12, 1990.
- [Stel84] J F Stelzer, "A Simple but Effective Method to Produce Color FEM Result Presentations," *Engineering Computation*, Vol. 1, pp. 227-231, 1984.
- [Stri89] T Strid and A Rizzi, "Development and Use of Some Flow Visualization Algorithm," VKI Lecture Series on Computer Graphics and Flow Visualization on CFD, 1989.
- [Sutc76] D C Sutcliffe, "An Algorithm for Drawing the Curve $f(x,y)=0$," *Computer Journal*, Vol. 19, pp. 246-249, 1976.
- [Suth74] I E Sutherland, R F Sproull, and R A Schumacker, "A Characterization of Ten Hidden-Surface Algorithms," *Computer Surveys*, Vol. 6, No. 1, 1974.
- [Ullm76] J Ullman, "Fundamental Concepts of Programming Systems," pp. 108-114, Addison-Wesley Pub.Co., 1976.
- [Upso86] C Upson, "The Visual Simulation of Amorphous Phenomena," *Visual Computer*, Vol. 2, No. 5, pp. 321-326, 1986.
- [Upso88] C Upson and M Keeler, "The V-Buffer: Visible Volume Rendering," *Computer Graphics*, Vol. 22, No. 4, pp. 59-64, 1988.
- [Veen77] A Veen and L D Peachey, "TLOTS: A Computer Graphics System for Three-Dimensional Reconstruction from Serial Sections," *Computers and Graphics*, Vol. 2, pp. 135-150, 1977.

- [Volp87] G Volpe, "Streamlines and Streamribbons in Aerodynamics," AIAA Paper 87-1182, 1987.
- [Voss83] R Voss, "Fourier Synthesis of Gaussian Fractals: 1/f Noises, Landscapes, and Flakes," Tutorial on State of the Art Image Synthesis v.10, SIGGRAPH '83, 1983.
- [Wall91] A Wallin, "Constructing Isosurfaces from CT Data," IEEE Computer Graphics and Applications, Vol. 11, No. 6, pp. 28-33, 1991.
- [Wegh84] H Weghorst, G Hooper, and D P Greenberg, "Improved Computational Methods for Ray Tracing," ACM Transactions on Graphics, Vol. 3, No. 1, pp. 52-69, 1984.
- [West89] L Westover, "Interactive Volume Rendering," Proceedings of the Chapel Hill Workshop on Volume Visualization, pp. 9-16, 1989.
- [West90] L Westover, "Footprint Evaluation for Volume Rendering," Computer Graphics. Vol. 24, No. 4, pp. 367-376, 1990.
- [Wilh90a] J Wilhelms, J Challinger, and A Vaziri, "Direct Volume Rendering of Curvilinear Volumes," Computer Graphics, Vol. 24, No. 5, pp. 41-47, 1990.
- [Wilh90b] J Wilhelms and A V Gelder, "Topological Considerations in Isosurface Generation." Computer Graphics, Vol. 24, No. 5, pp. 79-86, 1990.
- [Wilh91] J Wilhelms and A V Gelder, "A Coherent Projection Approach for Direct Volume Rendering," Computer Graphics, Vol. 25, No. 4, pp. 275-284, 1991.
- [WilC92] C Williams, J Rasure, and C Hansen, "The State of The Art of Visual Languages for Visualization," IEEE Visualization '92 Proceedings, pp. 202-209, 1992.
- [Will92a] P L Williams, "Interactive Splatting of Nonrectilinear Volumes," IEEE Visualization '92 Proceedings, pp. 37-44, 1992.
- [Will92b] P L Williams, "Visibility Ordering Meshed Polyhedra," ACM Trans. on Graphics. Vol. 11, No. 2, pp. 103-126, 1992.
- [Wrig79] T Wright and J Humbrecht, "ISOSRF: An Algorithm for Plotting Iso-Valued Surfaces of a Function of Three Variables," Computer Graphics, Vol. 13, No. 2, pp. 182-189, 1979.

- [Wyvi86] G Wyvill, C McPheeters, and B Wyvill, "Data Structure for Soft Objects," *Visual Computer*, Vol. 2, No. 4, pp. 227-234, 1986.
- [Yate91] L A Yates and G T Chapman, "Streamlines, Vorticity Lines, and Vortices," Paper AIAA-91-0731, 1991.
- [Yeo84] M F Yeo, "An Interactive Contour Plotting Program," *Engineering Computation*, Vol. 1, pp. 273-279, September 1984.
- [Yerr84] M Yerry and M Shephard, "Automatic Three-Dimensional Mesh Generation by the Modified-Octree Technique," *Int.J.num.Meth.Engng*, Vol. 20, pp. 1965-1990, 1984.
- [Yeun88] P K Yeun and S B Pope, "An Algorithm for Tracking Fluid Particles in Numerical Simulations of Homogeneous Turbulence," *J. Comp. Physics*, Vol. 79, pp. 373, 1988.

