Recent Advances in Visualization of Volumetric Data

Ken Brodlie and Jason Wood

School of Computer Studies, University of Leeds, Leeds, UK

Abstract

In the past few years, there have been key advances in the three main approaches to the visualization of volumetric data: isosurfacing, slicing and volume rendering, which together make up the field of volume visualization.
In this report we set the scene by describing the fundamental techniques for each of these approaches, using this to motivate the range of advances which have evolved over the past few years.
In isosurfacing, we see how the original marching cubes algorithm has matured, with improvements in robustness, topological consistency, accuracy and performance. In the performance area, we look in detail at pre-processing steps which help identify data which contributes to the particular isosurface required. In slicing too, there are performance gains from identifying active cells quickly.
In volume rendering, we describe the two main approaches of ray casting and projection. Both approaches have evolved technically over the past decade, and the holy grail of real-time volume rendering has arguably been reached.
The aim of this Eurographics 2000 STAR is to pull these developments together in a coherent review of recent advances in volume visualization.

1. Introduction

1.1. The problem and its applications

One of the enduring challenges in scientific visualization is the display of three-dimensional data on a two-dimensional display surface. By *three-dimensional data*, we mean data values obtained at sample locations within a three-dimensional space - such as temperature readings within an enclosed volume. We shall use the term *volume visualization* to describe this field of study.

The importance of the problem derives from the many applications in which this type of data occurs, and for which there is a need to gain insight through visualization. As the size of datasets continues to increase, so the importance of visualization as a tool grows, and so too the need to find more effective and efficient techniques. Volume datasets come from two main sources: firstly, from measurement or observation, such as in medical imaging through MRI, CT and other modalities, and from the use of high power microscopes; and secondly, from numerical simulations such as in Computational Fluid Dynamics where the aim is to understand (and predict) the behaviour of natural processes.

The field of volume visualization is now extremely large, and so any review is necessarily a selective process. Our hope is to be able to describe the fundamental techniques, and some of the developments from these that have occurred in recent years. For a more thorough study, the reader is encouraged to consult the original references listed in the bibliography at the end. There are useful overview papers also: the survey by Elvins [19], although now quite old, remains a very clear introductory exposition; and the chapters by Bajaj et al [3] and Yagel [85] together cover much of the field. The book by Lichtenbelt et al [43] provides a very readable introduction to the volume rendering part of volume visualization.

1.2. Reference Model

It is useful to pose the problem (and its solution) in terms of a reference model. We assume that we are given a set of data values at specified locations within a three-dimensional space. These data values are samples of some underlying phenomenon - which we might call reality - and it is the challenge of visualization to provide insight into this unknown reality. Mathematically, we are given values f_i at a set of points $(x_i, y_i, z_i), i = 1, 2, ..N$, which are sampled values from some underlying continuous function $f(x, y, z)$. There can be different arrangements of the data: *structured* or *unstruc-*

tured. An important distinction in structured grids is between *rectilinear* and *curvilinear*. Rectilinear grids are common in medical applications as the output from scanners; curvilinear and unstructured grids are common in numerical simulations. These grids are illustrated in Figure 1.

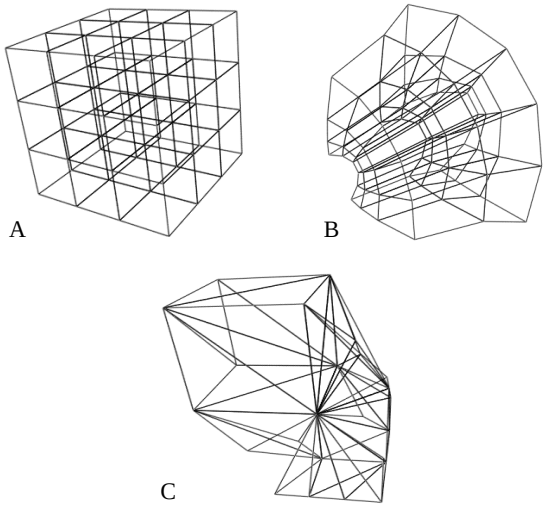

Figure 1: *Grids - A) Rectilinear, B) Curvilinear, C) Unstructured*

Note that in medical applications the data returned is not strictly point-based, but the average value over the grid cell - or *voxel*. This reflects the way in which scanners work. Thus papers which are aimed towards medical applications are often written in this context - sometimes called *voxel-based*. Although there is a single value per voxel, it is recognised that a volume feature may only occupy part of the voxel - this is known as the *partial volume effect*. In this report, we shall work in terms of a point model - sometimes called *cell-based* - where datavalues are given at the vertices of the grid and the behaviour inside a cell is assumed to be estimated through an interpolation process. (Unfortunately this is only one of many variations in the underlying assumptions and terminology of the subject, which can make it difficult to gain an overall picture.)

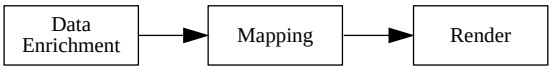

Figure 2: *Haber and McNabb reference model*

The process of transforming data into picture is fundamental to all scientific visualization. The classical paper by Haber and McNabb [27] remains the clearest exposition of this process. They model visualization as a sequence of three fundamental steps (see Figure 2):

Data Enrichment In this step, we reconstruct an estimate, $F(x,y,z)$, of the unknown $f(x,y,z)$. This step is present, explicitly or implicitly, in any visualization, and can be thought of as a modelling operation. In the case of three-dimensional data considered here, data enrichment is usually an interpolation process.

For rectilinear data, piecewise trilinear interpolation is much the most common - providing a useful compromise between the greater speed, but lesser accuracy, of nearest neighbour interpolation, and the greater accuracy, but less speed, of tricubic interpolation. Thus within each grid cell, a **trilinear interpolant** of the form

$$F(x,y,z) = a + bx + cy + dz + eyz + fzx + gxy + hxyz \quad (1)$$

is fitted to the eight data values at the cell vertices. An efficient way of computing trilinear interpolants is described by Hill [31]. An example of the use of tricubic interpolation in volume visualization can be found in [14], where the higher order interpolation is used to gain a smooth model of a binary dataset.

For unstructured data, it is usual to form a tetrahedral decomposition of the data points. Within each tetrahedron, a **linear interpolant** can be created, of the form:

$$F(x,y,z) = a + bx + cy + dz \quad (2)$$

fitted to the four data values at the tetrahedron vertices.

A general discussion on the research issues in volumetric modelling is given by Nielson [56]. An earlier paper by the same author [54] is also a very useful reference for unstructured data interpolation.

Mapping The next step is to choose some geometric interpretation of this function $F(x,y,z)$, that can provide some useful understanding of its behaviour. This geometric representation will typically be an object in three-dimensional space that can be rendered on a two-dimensional display surface. The capabilities of graphics hardware devices influence the choice of technique here, as it is important to generate representations that can be rendered fast.

In volume visualization, there are three distinct approaches to the mapping step. These are:

Surface extraction A surface shell is extracted from the data, containing points with a common value of F. Here we are visualizing the set of points (x,y,z) such that

$$F(x,y,z) = k \quad (3)$$

for some threshold value k. We can see this as a subset in the space of the **dependent** variable, and is commonly called isosurfacing. The surface is typically approximated as a triangular mesh which can be passed as geometry to a rendering process. For a piecewise trilinear interpolant, the isosurface proves to have interesting properties: it is a piecewise conic surface with often complex topology. For efficient rendering on con-

ventional graphics display devices, this has to be approximated by a triangular mesh - a problem which has proved a challenge throughout the last decade, as we shall see in section 2. If time is used as an additional display dimension, then we can visualize the entire dataset by sweeping the threshold through the range of the data.

Slice A two-dimensional slice is taken through the data (often parallel to one of the co- ordinate planes), allowing a two-dimensional technique such as coloured image or contouring to be applied. We can see this as a subset in the domain of the **independent** variables. Thus we are visualizing:

$$F(x,y,z) \mid (x,y,z) \in P(x,y,z) = 0 \qquad (4)$$

where $P(x,y,z) = 0$ is a plane. Again using time as an additional display dimension, we can sweep the slice plane through the data in order to gain a visualization of the full dataset. We look at slicing in section 3.

Volume rendering a 3D model of the data is created, using colour and opacity to reflect data values. The effect is to create a translucent gel material that can be passed to a renderer for display using computer graphics techniques. This is the most ambitious approach in that it aims to display all the data, not just a subset. We study volume rendering in section 4.

Rendering This is the final step in the pipeline, where the geometry created by the mapping step is realised as an image on the display surface, using standard computer graphics techniques. Conceptually, Haber and McNabb are correct to see this as a separate stage but as we shall see the mapping and rendering stages in practice are closely intertwined. For example, in surface extraction we have described how the mapping stage for rectilinear grids produces a triangular mesh which only approximates the isosurface of the true piecewise trilinear interpolant $F(x,y,z)$. In the rendering stage we can compensate for this approximation by clever shading which gives the illusion of a more accurate geometric representation.
In the slicing approach, the rendering is straightforward - the display of a coloured plane.
Volume rendering until recently has been regarded as significantly more expensive at the rendering step, but as we shall see later, this is changing - both through new approaches, and new hardware technologies.

The rest of the report is structured by the mapping step which is the key discriminator. We look at recent developments in each of the three areas in turn: surface extraction; slicing; and volume rendering. The majority of the work has been in the surface extraction and volume rendering areas, but slicing is included because it is a very commonly used approach. In each case we begin with a brief review of the classical approaches, to motivate and set the scene for the description of the new developments.

2. Surface Extraction

2.1. The classical approach

The classical approach to surface extraction is the **Marching Cubes** algorithm, proposed by Lorensen and Cline [48], with a similar suggestion from Wyvill et al [84]. This assumes data is on a rectilinear grid, and conceptually it processes each grid cell, or cube, independently, one after the other - hence the term *marching cubes*. The method is quite simple. Each vertex of a cube can be either greater than or less than the threshold value, k say, giving 256 different scenarios. An estimate $F(x,y,z)$ can be constructed as a trilinear interpolant of the values at the cube vertices. The intersections of the isosurface $F(x,y,z) = k$ with the edges of the cube are easily and accurately calculated by inverse linear interpolation. As mentioned earlier, the behaviour of $F(x,y,z) = k$ inside the cube is non-trivial and is a conic surface. However a simplistic estimate of F within the cube can be made by joining intersection points into a set of triangles. Lorensen and Cline argued that for reasons of symmetry and complementarity there are only 15 canonical configurations, and proposed corresponding triangulations of the isosurface (see Figure 3). For a given configuration (from the set of 256), they provide a look up table to give the corresponding canonical configuration and hence its triangulation.

This algorithm has been much used over the years since 1987, and has proved very effective in combination with fast triangle rendering hardware as provided on Silicon Graphics workstations, and more recently on PC graphics boards supporting OpenGL.

There are two major aspects of the algorithm which have received attention in recent years:

Surface representation The classical marching cubes algorithm has a naive approach to forming the interior representation. It was discovered quite quickly (see Durst [17]) that *holes* can appear when two adjacent cells have certain configurations. Much work has gone into making the algorithm more robust. In addition, there has been recent attention to the issue of accuracy, and gaining a more faithful representation of the true isosurface within each cell. We discuss this in section 2.2.

Performance As computing power and measurement technology have increased, so has the size of datasets that users wish to analyse. The marching cubes algorithm can be rather slow - both in terms of locating cells which contain segments of the isosurface, and also in terms of rendering the large number of triangles which may result. Recent developments in performance are discussed in section 2.3.

When the data is on a tetrahedral mesh, the case is simpler in terms of robustness. The isosurface of the linear interpolant $F(x,y,z) = k$ is a plane and so there is no approximation involved in the triangulation. The method is known as Marching Tetrahedra - for obvious reasons (see Doi and Koide [16]).

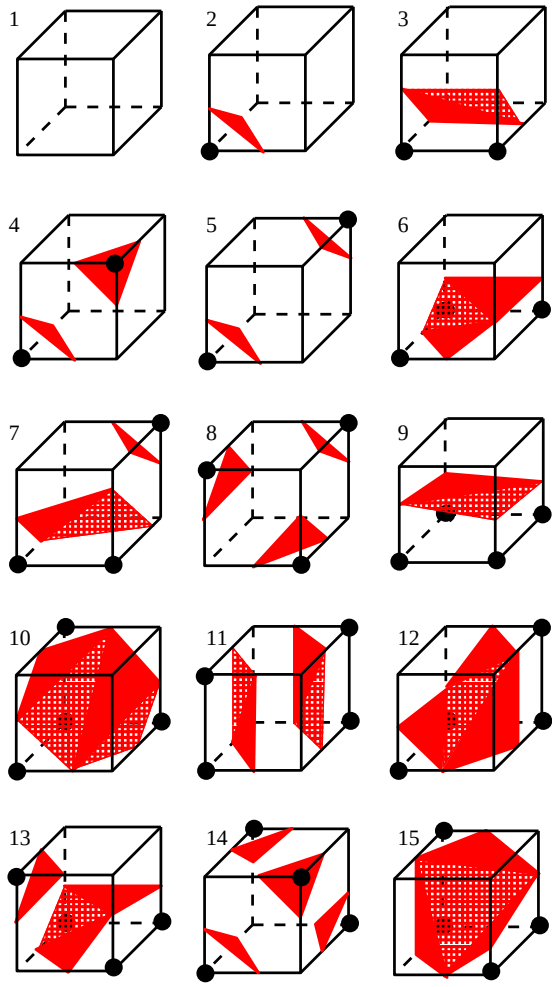

Figure 3: *The 15 Marching Cubes configurations*

2.2. Surface Representation

In this section we look at recent efforts to make the marching cubes isosurface algorithm give an improved representation of the surface - in terms of robustness, topological correctness and accuracy. We also look at how the rendering and representation are intertwined - see final section on rendering issues. For simplicity, we shall assume the isosurface value, k, is zero.

2.2.1. Robustness

The original marching cubes algorithm reduced the 256 possible cases to one of 15 canonical configurations. This enabled a small look up table and efficient coding, but caused inconsistent matching of surfaces between adjacent cells, so that 'holes' could appear. A remedy is to return to a full 256 case table of triangulations, and this is used for example in the vtk implementation of the algorithm (see Shroeder et al [67]), and discussed by Bartz et al [4].

2.2.2. Topological Correctness

The isosurface of a trilinear interpolant is the surface:

$$F(x,y,z) = a + bx + cy + dz + eyz + fzx + gxy + hxyz = 0 \quad (5)$$

F is linear along edges of the cube, bilinear across the faces of the cube and trilinear in the interior. The marching cubes algorithm in its basic form is happy to live with correctness along the edges. Nielson and Hamann [55] showed that faces where one pair of opposite vertices have datavalues of different sign from the other pair are ambiguous, in the sense that the contour line (marking intersection of isosurface with face) could be drawn so as to *cut off* either the positive-valued corners, or the negative-valued corners. As a way of resolving the ambiguity, they propose following the topology of the bilinear interpolant on the face - which is easily determined by looking at the asymptotes of the hyperbolic contours. Of the 15 canonical configurations in the marching cubes algorithm, six have a number of ambiguous faces. Nielson and Hamann show how subcases can be constructed to cover the alternative topologies that may result. The *asymptotic decider* approach is robust in the sense that no holes are left, in addition to being topologically consistent across faces. The interior triangulation strategy is one of simplicity, and does not concern itself with interior topology.

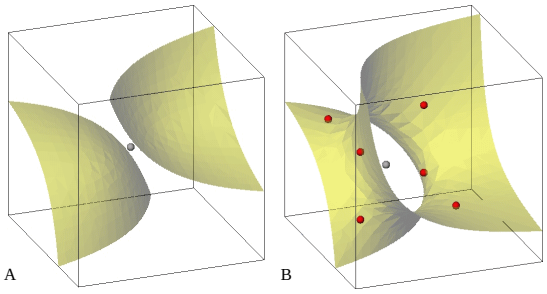

Figure 4: *Two internal configurations for the Marching Cubes configuration 5*

By contrast, Natarajan [53] studies the exact shape of the interior surface, and finds that further ambiguities can occur. For example, consider a cube where one pair of opposite vertices are positive, the remaining six vertices negative (this is configuration 5 in Figure 3. A simple triangulation will cut off the positive vertices with single triangular pieces, but these approximate a curved surface which bows out towards the centre of the cell. Imagine the data values all increasing uniformly so the zero isosurface pieces move towards each other - the simple minded triangular approximation remains separated as two pieces, but the true trilinear surface pieces

will come into contact with each other, and as that happens they merge into a single surface with a tunnel. This is an internal ambiguity, rather than the face ambiguity treated by Nielson and Hamann. This is illustrated in Figure 4 where we see the two situations (A and B) that can occur internally for a single vertex configuration. We show the exact isosurface of the trilinear interpolant. Natarajan shows that a key to identifying tunnels is the value at the *body saddle point* which is the 3D equivalent of the saddle point whose value was exploited by Nielson and Hamann to determine face ambiguities. The body saddle is located where the transition from one to two pieces occurs. In the figure, A shows the situation with two pieces, and the small sphere marks the body saddle point. In B, the pieces have merged with a tunnel appearing. (The body saddle is still shown - the other small spheres will be explained later).

A definitive treatment of topological correctness in isosurfacing on rectilinear grids is presented by Chernyaev [9]. He identifies some 33 canonical configurations, covering both face and interior ambiguities. This is a little known, but significant, paper which subsumes the previous work by Nielson and Hamann, and Natarajan. Chernyaev takes each of the fifteen original canonical configurations, which are based on vertex values. The cases which involve face ambiguity are then subdivided into subcases; and of these, those that also involve internal ambiguities are further subdivided.

More recently, Cignoni et al [12] have made a similar study, working from the Natarajan paper. They show that ambiguities extend the 256 cases to some 798 different cases, but only 88 of these are distinct configurations. Work is needed to unify the Chernyaev and Cignoni papers, and then this matter may (possibly) be put to rest.

2.2.3. Accuracy

Some recent work has attempted to develop beyond the robustness and topological correctness, in order to increase the accuracy of the internal representation of the isosurface within the cell. If the cell size is very small relative to the display size, then simple internal triangulation is quite sufficient. However as we zoom into the data, or if the data itself is of lower resolution, then more care in representing the interior is justified. The expense of course is that more triangles are created.

One approach is by Hamann et al [28]. They approximate the surface of the trilinear interpolant by triangular rational-quadratic Bezier patches. While this better reflects the curved interior of the true isosurface, there is no guarantee that any interior points of the approximating surface lie on the true isosurface.

The starting point for Lopes [46] is 2D visualization by contouring. Lopes and Brodlie [47] propose a more accurate contouring method for 2D visualization. Within a grid cell, the unknown $f(x,y)$ can be approximated by a bilinear function

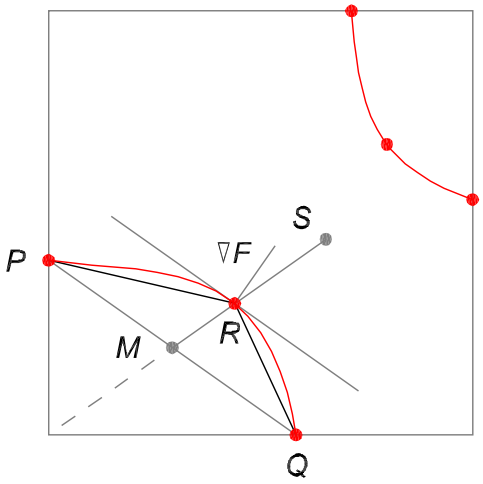

Figure 5: *Shoulder Point*

$F(x,y)$. The contour of a bilinear function within a grid cell is an hyperbolic arc, but it is usual to approximate this with a single straight line joining the intersection points of the contour with the cell edges (the equivalent of using triangles in isosurfacing). They show that a better approximation to the hyperbolic arc can be obtained by calculating the *shoulder point* of the conic section - this is the point on the conic parallel to the chord joining the end-points. This gives two-piece linear approximation to the conic arc. The shoulder point is an optimal point to choose in forming this approximation, as it is the furthest point on the arc from the chord. This is shown in Figure 5: P and Q are the end-points of the hyperbolic arc, and R is the shoulder point of the arc. R is quite easy to calculate as it lies on the line joining M, the mid-point of the chord PQ, and S, the saddle point of the bilinear interpolant.

In his thesis [46], Lopes extends this to isosurfacing. He takes the intersection points of the isosurface with cell edges to form an initial polygonal outline of the isosurface - this is exactly as done in the classical approach. The edges of the polygon lie on the cell faces, and are approximations to the isocontour lines on the faces. We can extend this polygon by adding shoulder points, exactly as in the 2D contouring case. This improves the accuracy on the faces. In the interior Lopes defines two special classes of points which help define the internal behaviour of the surface. One class is called *bi-shoulder points* which are 3D analogues of shoulder points; the other is called *inflection points* which are generalisations of the Natarajan body saddle point. Lopes is able to use these points to generate efficient triangulations that correctly represent the interior topology, as well as increasing the accuracy. In Figure 4 the Lopes inflection points are shown as small

spheres surrounding the body saddle in B. As can be seen, they nicely delineate the shape of the tunnel.

Lopes' work aims to create a good internal representation by adding a minimal number of carefully chosen extra points - so that a minimal number of extra triangles are created. Recent work by Cignoni et al [12] and Bailey [2] are both based on the idea of progressive mesh refinement: the mesh is continually subdivided until it approximates the isosurface to a specified tolerance. Cignoni et al propose the following refinement strategy. Each triangle is considered in turn: firstly, the midpoints of each side are evaluated for their distance from the true isosurface; and secondly, the centre point of the triangle is similarly evaluated. The distance is measured in the direction of the estimated gradient vector at the points. If this distance exceeds a tolerance, then the mesh is refined.

The work of Bailey is motivated by the application of isosurfaces in a manufacturing process, where accuracy is everything and efficiency (in terms of number of triangles) is of less importance. The approach is similar to that of Cignoni: each midpoint of the sides of a triangle, and its centroid, are checked for their difference to the true isosurface, and if necessary, these points are moved along a gradient direction to lie on the isosurface and used in a refined mesh. A difference to Cignoni is that the 'distance' measurement is made in the dependent variable space - ie the value of the interpolant at a candidate point is compared with the isosurface value - rather than being measured in the independent variable space. Cignoni discusses the merits of the two approaches.

There is probably scope for some future research in combining the Lopes approach of choosing optimal refinement points at the first level, and the Bailey and Cignoni progressive refinement approaches at subsequent levels (indeed Lopes hints at this in his thesis).

2.2.4. Rendering Issues

In isosurfacing usually (but not always) the interface to the renderer is a triangular mesh, often in the form of triangle strips for efficiency. In the simplest form, these triangles are passed on to the renderer without specific normal vectors at the vertices. The renderer will then apply either flat, Gouraud or Phong shading - the latter two techniques having a visual smoothing effect on the surface.

However it is possible to incorporate *visual accuracy* in the rendering by supplying normal vectors at the triangle vertices that reflect the normal direction of the true isosurface. This can be seen as a *trompe l'oeil* of the same nature as bump mapping in computer graphics. The normal is manipulated so as to deceive the eye into thinking the geometric representation is (in this case) more accurate than it really is - or more positively, so as to economise on the number of triangles rendered while still reflecting the true isosurface.

The normal is equal to the gradient vector of the isosurface. For rectilinear meshes the gradient vector is easily calculated by central differences at each vertex, and then linear interpolation gives the gradient, ie normal, vector at the triangle vertices. For unstructured meshes, the gradient vector at a mesh point is typically calculated by looking at the differences along all edges connected to that mesh point, and carrying out a least squares estimate. Note that this process is quite sensitive to the relative distances to neighbouring points, and some weighting in the least squares estimation can be beneficial.

There is a quite different approach to isosurface rendering in which there is no intermediate approximation of the isosurface by a triangular mesh. Instead the isosurface is directly rendered using a ray casting approach. This was proposed by Jones and Chen [34] - who termed it *direct surface rendering* - and later by Parker et al [59], [60] - who term it *interactive ray tracing for volume visualization*. The quality of image is very high since the exact isosurface is rendered, not an approximation, and Parker and colleagues show that this technique can be competitive in performance because it parallelises readily. With a number of intelligent optimizations they are able to render data from the Visible Human Project at interactive rates (10 frames per second, for a 512x512 image on 64 processor SGI Reality Monster, on a dataset of 1734 slices of 512x512 16bit data). There are strong arguments in favour of this approach if you have the compute power available - for example, all the topological issues discussed earlier are irrelevant. The only possible drawback is that there is no intermediate triangular mesh representation available for further computation, but this is probably a relatively uncommon requirement.

2.3. Performance

Algorithm performance has become increasingly important as our ability to capture or create data has grown at a rate that has outstripped the rate of improvement of computing technology. A number of strategies exist for improving the performance of isosurfacing extraction from large structured and unstructured data sets.

2.3.1. Presorting - Introduction

The classic algorithms for isosurface extraction, Marching Cubes and derivatives such as Marching Tetrahedra, operate by inspecting every cell of the data set looking for cells that contain the isosurface. We call these the "active cells". Generally a selected isosurface will only intersect a small subset of the overall dataset and so being able to locate these active cells efficiently will give improved performance. Presorting methods create extra data structures to allow efficient searching of the data set to find active cells. A number of presorting approaches have been published which can be classified based on whether they sort by data value or by spatial location.

2.3.2. Presorting - Value Partitioning

Value partitioning methods are generally used for unstructured data sets where connectivity between nodes is not implicity provided by location but must be specified. This means that the memory requirements for this type of data representation are large when compared to structured grids and hence the number of nodes represented is typically smaller. The relative overhead of the associated search structure compared to the memory requirements of the actual data set is less for an unstructured grid than for a structured grid and hence these methods are generally used in conjunction with unstructured data. Cignoni et al [13] however do offer a useful technique for applying value partitioning to structured data, see interval trees below.

Extrema Graphs

Itoh and Koyamada [33] proposed an algorithm for accelerated isosurface extraction which first preprocessed the data set to a list of extremum points (local minima and maxima) connected by a graph whose arcs contain the IDs of cells intersected by the graph. Additionally, two boundary cell lists, sorted according to the minimum and maximum data values of the cells, are generated. The cells in the boundary lists are intersected by an open isosurface, while the cells contained within the arcs of the extrema graph are intersected by closed isosurfaces. Using the cells contained within the lists as seed points for a surface growing algorithm, all possible isosurfaces can be generated.

The extrema graph is constructed by searching the data for minima and maxima, but since this would be overly expensive the extremum points are approximated to the nearest grid point. This reduces the search to examining the datavalue at each grid point and comparing it with all its neighbours and marking it as either a local maxima, local minima or neither. Once this is done all the grid cells marked as neither are discarded. Any local clouds of extremum points, caused by neighbours having the same value, are reduced to a single point. This final set of points are then connected to form the graph.

The graph is generated by selecting one extrema as a start point and a number of extrema points closeby as goal points. The nearest goal point is selected and the vector between the start and goal points generated. The arc between the two points along the vector is traversed by moving between neighbouring cells that lie on this vector. If the goal point is reached without leaving the volume then the IDs of all the visited cells are placed in the cell list associated with the arc, and the overall maximum and minimum values of all the cells in the arc are stored. If the vector between the two selected points leaves the volume, then another goal point is chosen and traversal starts again. If there are no goal points that can be reached by traversing along the vector, then a polygonal arc is used to join two points. Polygonal arcs are able to move off the straight vector and hence avoid leaving the volume, but they are more expensive to compute since the distance value of each face of a visited cell to the goal cell must be calculated.

Once all the extremum points are connected, the boundary cell lists are generated as two sorted lists using the maximum and minimum data values of the boundary cells. A boundary cell is defined as a cell which has at least one unshared face.

To find an isosurface of a given value, starting cells are searched for by traversing the arcs of the extrema graph. The given value is compared to the maximum and minimum value of the overall arc: if it lies within that range then each cell in the list is visted in turn, otherwise the next arc is tested. Once all the arcs have been tested, the boundary cell lists are traversed. All active cells that are found are used as seed point cells for a surface propagation algorithm such as that of Speray and Kennon [70].

Performance is estimated at $\mathcal{O}(n^{2/3})$ at best, but can be $\mathcal{O}(n)$ in worst case for noisy data.

Kd-Trees

Livnat, Shen and Johnson [44] propose an algorithm that utilises the kd-tree designed by Bentley [5] as a data structure for efficient associative searching. Essentially, the kd-tree acts as a multidimensional binary tree with the nodes at each level holding one of the data values and a connection to two subtrees. The two subtrees are constructed such that the left subtree holds values that are less than the value at the parent, the right subtree holds values that are greater. Unlike a binary tree, however, the next level of the tree down holds a different data value, with its children being partitioned relative to that data value. This continues cyclically down the tree, swapping data values at each level.

This data structure is ideal for creating a search tree for the cells of an unstructured mesh where each cell has an associated minimum and maximum data value. The tree is constructed by examining each cell in the data set to calculate its minimum and maximum data values. The first node in the tree then contains the cell ID and the min and and max value of the median cell as calculated based on the minimum values of all the cells. This can easily be done by partially sorting the cells on the minimum value using a median sort algorithm. The head of the tree then has pointers to the two subtrees made up from the remaining cells to the left and right of the median cell. For each subtree the above process is repeated, but this time the cells are partially sorted by their maximum value. Using the median value cell causes the creation of a balanced tree, and because one cell is left at each node the size of the tree is directly related to the size of the dataset.

Once the tree has been constructed it can then be searched to quickly find active cells. Given an isosurface value *k*, the active cells are found by first comparing *k* to the minimum value of the first node in the tree. If it is less than the minimum value, then the cell at that node is ignored and we can guarantee that all cells in the right subtree are not required and hence we have already eliminated half the cells in the

dataset. We move on to compare k to the maximum value of the left child of the top node only. If k lies between the minimum and maximum values of the cell at the node then we know that it is an active cell and is used in generating the isosurface; we must then go on to test both sub-trees. If k is greater than the maximum then we must test both subtrees but we do not have an active cell. When we compare k to a node that is sorted by maximum values, if it is greater then the maximum value at that node then we can discard all values to the left and just test cells to the right. This process is repeated until the bottom of the tree is reached.

Livnat et al go on to suggest some further searching optimisations, it can be noted that (considering a node sorted by minimum value) when k is greater then the minimum value we must search both subtrees and seemingly have gained nothing. This is true for the right subtree, but for the left subtree, we know that we have satisfied the minimum condition and hence may skip that test.

Performance is $\mathcal{O}(\sqrt{n}+K)$ for search, with preprocessing estimated at $\mathcal{O}(n\log n)$. Here K is the size of the output, ie number of cells selected.

Interval Trees

This method is designed to answer the following query "given a set $I = \{I_1, ..I_m\}$ of intervals of the form $[a_i, b_i]$, with $a_i \leq b_i$ on the real line, and a query value k, find all intervals of I that contain k" as a way of finding a set of active cells from which to construct an isosurface. Each cell can be reduced to an interval of the form $[a_i, b_i]$ by examining the data values at each vertex to find the min and max. Cignoni et al [13] use the interval tree defined by Edelsbrunner [18] as an optimally efficient data structure when solving this query, and the algorithm for constructing and searching is outlined below.

For each $i = 1, 2, ..m$ where m is the number of cells, consider the sorted sequence of values $X = (x_1, .., x_h)$ corresponding to the unique set of intervals (i.e. each range a_i, b_i is equal to some x_j). The interval tree for I consists of a balanced binary search tree T whose nodes correspond to values of X, plus a structure of lists of intervals attached to nonleaf nodes of T. The interval tree is defined recursively as follows. The root of the tree T has a discriminant $\delta_r = x_r = x_{[h/2]}$, and I is partitioned into three subsets as follows:

$$I_l = \{I_i \in I \mid b_i < \delta_r\} \tag{6}$$

$$I_r = \{I_i \in I \mid a_i > \delta_r\} \tag{7}$$

$$I_{\delta_r} = \{I_i \in I \mid a_i \leq \delta_r \leq b_i\} \tag{8}$$

The intervals that fall into the node are sorted into two lists, AL and DR as follows:

AL contains all elements of I_{δ_r} sorted into ascending order by their left extremes (that is by their minimum value)

DR contains all elements of I_{δ_r} sorted into descending order by their right extremes (that is their maximum value)

The left and right subtrees are defined recursively.

To perform a search on the tree T, given a query k, is a recursive process starting from the head of the tree

if $k < \delta_r$ then list AL is scanned until an interval I_i is found such that $a_i > k$; all scanned intervals are reported; the left subtree is visited recursively;

if $k > \delta_r$ then list DR is scanned until an interval I_i is found such that $b_i < k$; all scanned intervals are reported; the right subtree is visited recursively;

if $k = \delta_r$ then the whole list is reported.

This alternative and more optimal approach to the kd-tree is calculated to have a worst case time complexity of $\mathcal{O}(K + \log h)$ where K is the output size and h is the number of nodes in the tree.

As mentioned above, these search structures are generally used with unstructured data due to the relatively high overhead associated with using them for structured grids. Cignoni et al, however present a useful observation that potentially makes their overheads with uniform grids no worse than that of octree subdivision (discussed in Space Partitioning). They observe that rather than building an interval tree with an interval for every cell in the data set, it is possible to reach almost the entire dataset by encoding just approximately one in four carefully chosen cells. The cells chosen form a 3D chessboard where the black cells are encoded into the tree (A 2D example can be shown in Figure 6). The neighbouring white cells will be found if required since all of their edges touch a black square. For example, the square marked A touches 2 white squares, one to the right and one in front, cell B touches four, one behind, one to the right, one to the left and one to the front.

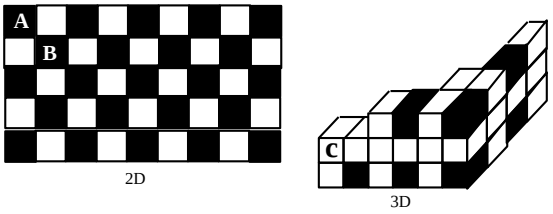

Figure 6: *Using Interval Trees for regular gridded data.*

This can be extended to a 3D volume (see Figure 6). When searching, performance can be gained by considering each layer with black cells as a separate interval tree since shared edge intersections and normals that are normally accumulated and stored over the whole data set can be discarded once a single tree has been searched. When a tree is searched and a black cell detected as active it is a simple matter to find cells that share the intersected edges. Once all the trees have been

searched a small number of cells around the edge of the that may not be linked to a black cell, cell C for example in Figure 6, need to be tested.

2D Lattice Subdivision of the Span Space

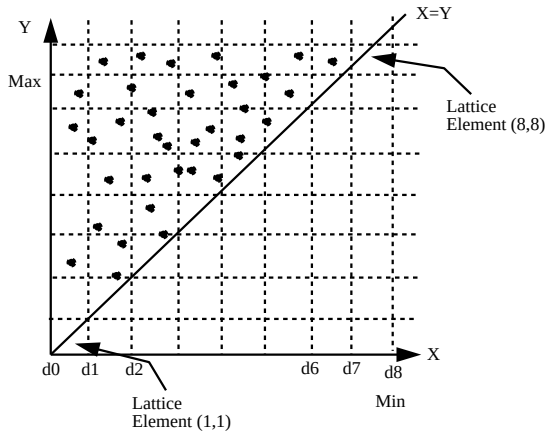

Figure 7: *Partitioning the Span Space*

Further work by Shen et al [66] looks at using the span space but instead of using a kd-tree as the search structure, they decompose the space into a 2D LxL lattice of cells (see Figure 7). The lattice elements are spread across the whole range of the data set, and the division points are chosen such that an even number of cells falls into each division. This obviously requires the elements to be of varying size.

Searching for a particular value k where k lies in the element (p, p) , it is possible to classify the lattice elements to one of five cases (see Figure 8) based on their indices (i, j) as follows :

- case 1: if $i > p$ or $j < p$ then there are no active cells in these elements as they have either a minimum value above k or a maximum value below k;
- case 2: if $i < p$ and $j > p$ then all cells in these elements are active cells.
- case 3: if $i < p$ and $j = p$ then all elements in this region have the potential to contain active cells and must be searched, but we know that their minimum values are below k so need only test their max values.
- case 4: if $i = p$ and $j > p$ the inverse of case 3
- case 5: $i = p, j = p$ We must test both min and max values of all cells in this element

The authors report that the search phase has an average performance of $\mathcal{O}(\log(n/L) + \sqrt{n}/L + K)$ where K is the number of active cells and L is the number of chosen subdivisions. The value of L is reported to be best between 200 and 500.

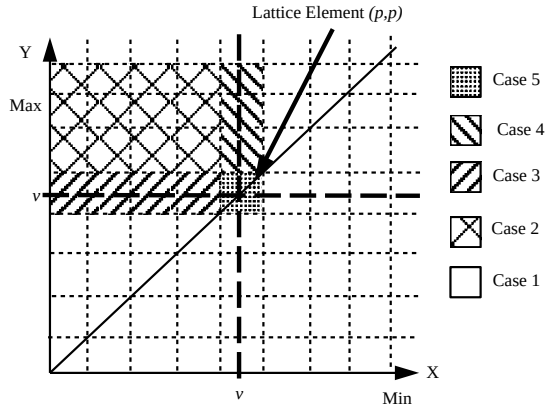

Figure 8: *Searching the Span Space*

2.3.3. Space Partitioning

This approach offers the simplest method to gain improved performance, but unfortunately is only suitable for structured data. Wilhelms and Van Gelder [80] use a branch-on-need octree which allows the creation of uneven sized sub-volumes unlike the standard even-octree subdivision. Using an octree (of any sort) allows the search algorithm to skip unrequired sub-volumes when fitting an isosurface based on the range of each sub-volume. If the required isovalue does not lie in the range of a high level volume, then all subvolumes within it can be skipped. An alternative approach based on a pyramid data structure is proposed by Criscione et al [15] which has similar efficiency and overhead to Wilhelms et al but is easier to implement. Space partitioning techniques cannot easily be applied to unstructured data as they rely on the regular structure of the underlying data set.

2.3.4. Multiresolution

Algorithms such as marching cubes inevitably produce a large number of polygons to represent a surface since every active cell in the data set contributes one or more triangles. The advantage of the multiresolution approach is that it preprocesses the data into a hierarchy of lower resolution volumes which means that triangle reduction effectively occurs at isosurface extraction time (rather than after when using mesh simplification). The resolution of the surface is not uniform across the volume and can be controlled by an error estimate which forces higher resolution where the data changes rapidly [87]. Alternatively, the resolution can be controlled by a user selected point and radius of focus [21].

There is a growing literature on this area - the paper by Gerstner and Rumpf [24] includes a good review of the field.

2.3.5. Exploiting parallelism and distributed processing

The marching cubes algorithm operates on individual cells within a data set without reference to or altering of any global

information. For this reason it is an ideal candidate for use as a parallel algorithm to speed up isosurface extraction from large data sets as long as the data can be suitably partitioned. One example of this type of work is that presented in a case study paper by Painter et al [58] from the Mantle project which uses a parallel implementation of the kd-tree of Livnat et al from above. Other more recent work has been presented by Lombeyda and Rajan [45] where they have taken the vtk marching cubes code and created a parallel implementation using p-threads, with IRIS Explorer as a front end for rendering and user interface. This has been applied to a data set generated by a numerical simulation of the Rayleigh-Taylor instability. Shen et al (2D Lattice Subdivision of the Span Space above) give results of using their algorithm on a parallel machine. They suggest passing each lattice element from the half space above the $x = y$ line, column by column, to a different processor using a round-robin method to give an even distribution of cells of all values. Their results give a load imbalance of just 2%. Other work in this area has been by Gerstner et al [24].

Another way of isosurfacing large datasets while only having access to a desktop workstation is to make use of remote resources through distributed processing. Work done by Engel et al [21] has looked at the idea of visualizing data across the network with particular focus on using the world wide web. They propose a 6 stage model for isosurface extraction, starting at data storage and ending with the displayed image, that places a different number of stages on the client and server machines. They then go on to briefly evaluate the relative strengths and weaknesses of different placement strategies finally focussing on 3 scenarios. Two scenarios are rejected because they are view dependent and would require data transmission whenever the viewpoint is altered, and they reject the option of simply using the remote resource as a data server since the size of the data sets are considered too large. Having identified the three remaining scenarios, sending triangles, sending interpolation values and the Marching Cubes cases with client side triangle setup or sending active cells for local triangulation, as possibilities, they then offer some mechanisms to help reduce network usage. These include combining triangles in individual cells into triangle strips to reduce the number of indices sent to the client, as well as a multiresolution approach taking into account a user specified point and radius of interest. Other work by Engel et al [20] has looked at progressive isosurface transmission across the web from a server calculation of the isosurface.

2.3.6. Out of core isosurface extraction

The methods described above in presorting seek to gain performance improvements by providing efficient search structures to use alongside the data set to quickly find active cells. The construction time and memory requirements of these additional data structures needs to be considered when implementing such systems. The time factor may effectively be ignored as a "run-time" cost since many of the above data structures can be pre-generated and stored to disk as a one off processing step and loaded along with the data when isosurface extraction is to be performed. This, however, still leaves the issue of memory requirement.

Out of core algorithms of all types are designed for situations where the amount of data to be processed is simply too large to fit in main memory. These algorithms avoid the time wasted by disk thrashing by employing unique data structures on disk that offer optimal I/O performance and also seek to exploit locality. Work done by Arge and Vitter [1] provided an optimal external memory data structure for the stabbing query problem which can be used for 2D range searching. Chiang and Silva [10] use this and ideas from the interval trees of Cignoni (above) to create an I/O optimal interval tree which they then go on to demonstrate using vtk. Results from their paper show that the time taken to generate the isosurface from the active cells is actually greater than the time taken to search for these cells. These same authors then go on to provide improved data structures to reduce the amount of disk space required to build and store the search structures. This new work (presented in [11]) uses meta cells which contain a group of neighbouring cells, and it is these metacells that are then built into an interval tree. Depending on the number of cells in a meta cell there is a direct tradeoff of disk overhead (the more meta cells the larger the disk overhead) to query time (the more meta cells the quicker the query time). Work by Sulatycke et al [71] takes note of the fact that the seach for active cells is quicker than the time taken to generate the isosurface. To take advantage of this they have developed a multithreaded system that uses a single I/O thread for finding and reading the active cells and a number of computational threads for isosurface generation. They work as a producer-consumer system with the I/O thread placing active cells into a buffer and the computational thread removing cells from the buffer when they have finished their current processing. If no data is ready then the computational thread blocks until data arrives. The authors report speedups over the vtk out-of-core implementation by an order of magnitude or more. Other work on parallel out-of-core visualization has been done by Bajaj et al [3].

2.4. Time Varying Isosurfaces

There is increasing interest in finding efficient methods for the construction of isosurfaces from time varying data. An early paper by Weigle and Banks [76] constructed an isovolume of the set of isosurfaces of level k over a period of time; a second pass can be used to extract the isosurface at a particular time.

Papers by Shen [65], and by Sutton and Hansen [72], introduce some of the presorting ideas described in the previous section, in order to increase efficiency. In particular, Sutton and Hansen extend the Wilhelms and van Gelder branch-on-need octree method of spatial partitioning.

3. Slicing

Slicing is the least glamorous of the three fundamental approaches, perhaps because it reduces the problem to a 2D visualization problem, or sequence of such. Thus it makes use of techniques that have been studied from the early days of scientific visualization - in the 1960s. However it remains a very valuable technique and many users are better able to comprehend 2D information than 3D. Every visualization package will include a slicing module!

The naive approach to slicing is to do an exhaustive examination of all the cells to test for intersection with the slicing plane. Some obvious speedups can be achieved for structured gridded data when the slicing plane is orthogonal to one of the principle axes. For unstructured grids this is not possible, but an approach similar to those used above in presorting can be used. A number of the presorting methods reduce the problem to a 2D range search, but unstructured grids are made up of a range of element types from tetrahedra to hexahedra all of which are 3 dimensional and hence produce a 6D range search. It is, however, possible to reduce the problem to a 2D range search if the orientation of the slice plane is known. The rotational component of the slice plane from the *xy*-plane is calculate and its inverse is applied to each cell in turn and its minimum and maximum *z*-component recorded along with the cell ID. Once all cells have been inspected a search tree can be constructed as described above, e.g a kd-tree, based on the minimum/ maximum *z*-values of the cells. The query value, *k*, then is simply the distance from the origin to the slicing plane. The tree can be used to quickly find slices at any distance from the origin as long as they keep the original orientation. This allows planes to be rapidly swept through the data to give a good impression of the structures within. This method can be used for planes of arbitrary orientation on any unstructured grid.

4. Volume Rendering

4.1. Introduction

Volume rendering offers a more complete solution to the volume visualization, in that it aims to picture the entire volume rather than a subset. It has been traditionally thought of as more computationally expensive than surface extraction, but this view has recently been challenged by exciting new hardware developments.

The technique is based on modelling the data as a translucent gel, and so a fundamental first step is to assign material properties to correspond to the data values. *Classification* is the process by which we assign a colour and opacity value to a given data value. The *opacity transfer function* will take as input certainly the data value, but perhaps also other information such as gradient estimates, and return an opacity value. The gradient value is used when interior structure (such as anatomical features in medical imaging) is to be highlighted: the opacity is scaled down in areas of low gradient, and up where the gradient is high - thus emphasising boundaries between features. *Colour transfer functions* do a similar job in assigning RGB values to data values. The transfer functions may also use information from a prior *segmentation* process which has labelled data as belonging to a particular feature in the volume. Lichtenbelt et al [43] discuss segmentation in relation to volume rendering.

Classification remains something of an art. Kindlmann et al [37] suggest ways of automating the process.

4.2. Classical Approaches

4.2.1. Volume Rendering Integral

The basis of most volume rendering techniques is the volume rendering integral in its low-albedo form, as derived by Kajiya and von Hertzen [36] and by Max [49]. A very clear exposition is given by Mueller et al [51] and we follow this here. We imagine the volume as a set of particles with certain densities μ, and fire rays through each pixel on the image plane into this volume. For any ray, the amount of light of wavelength λ received at the image plane is given by:

$$I_\lambda = \int_0^L C_\lambda(s)\mu(s)e^{-\int_0^s \mu(t)dt}ds \qquad (9)$$

where L is the length of the ray, and $C_\lambda(s)$ is the light of wavelength λ reflected at s in the direction of the ray. The calculation of $C_\lambda(s)$ can be based on the standard Phong reflection model, given specification of light sources, the material colour from the classification process and the normal direction (which as we have seen is the gradient vector). The weighting by $\mu(s)$ reflects the density at the point - the greater the density, the greater the intensity of reflected light. The integral accumulates this intensity over the length of the ray, but attenuates it according to the density of material through which it passes. This attenuation is represented by the exponential term. Max [49] calls μ the light extinction coefficient; it defines the rate at which light is occluded per unit length due to scattering or extinction.

In practice, the integral given by equation 9 has to be evaluated numerically. Using a very simple Riemann sum approximation, we have

$$I_\lambda = \sum_{i=0}^{n} C_\lambda(i\Delta s)\mu(i\Delta s)\Delta s \prod_{j=0}^{i-1} exp(-\mu(j\Delta s)\Delta s)) \qquad (10)$$

where n is the number of steps along the ray at which sample values are taken.

We can simplify this by a number of approximations. The exponential term in equation 10 can be replaced by the first two terms of its Taylor expansion, ie

$$exp(-\mu(i\Delta s)\Delta s)) = 1 - \mu(i\Delta s)\Delta s \qquad (11)$$

Define the transparency $t(i\Delta s)$ as:

$$t(i\Delta s) = exp(-\mu(i\Delta s)\Delta s)) \tag{12}$$

so as to give:

$$\mu(i\Delta s)\Delta s = 1 - t(i\Delta s) = \alpha(i\Delta s) \tag{13}$$

where $\alpha = 1 - t$ is opacity.

This approximation converts equation 10 into the compositing formula that is commonly used in volume rendering, namely

$$I_\lambda = \sum_{i=0}^{n} C_\lambda(i\Delta s)\alpha(i\Delta s)\prod_{j=0}^{i-1}(1-\alpha(j\Delta s)) \tag{14}$$

The values of C and α are only known at data points and so an interpolation process is required in order to calculate the values at the sample points $i\Delta s$. It is possible to interpolate data values and then classify, but the above derivation assumes that the classification and shading are done at the data points, and the resulting colours and opacities used for interpolation.

If we use unit spacing, equation 14 simplifies further to:

$$I_\lambda = \sum_{i=0}^{n} C_\lambda(i)\alpha(i)\prod_{j=0}^{i-1}(1-\alpha(j)) \tag{15}$$

In practice, the computation is done for R, G, B separately, and from now on we remove the λ suffix. Basically, it is a sum over intensities of individual samples, each intensity attenuated by the product of transparencies accumulated as the light passes from sample to observer.

The calculation can be done recursively by processing one sample at a time, accumulating colour and opacity separately:

$$C_{out} = C_{in} + (1-\alpha_{in})\alpha_i C_i \tag{16}$$

for each sample i, and

$$\alpha_{out} = \alpha_{in} + \alpha_i(1-\alpha_{in}) \tag{17}$$

This corresponds to the Porter and Duff image composition operator **over** [63]. This is a front-to-back ordering. In fact, the order can be reversed to work back-to-front, in which case only the colour needs to be accumulated:

$$C_{out} = C_i\alpha_i + C_{in}(1-\alpha_i) \tag{18}$$

It is worth noting that the compositing steps are associative but not commutative. This has two important implications: associativity means that we can composite groups of samples, then composite the groups, as long as we retain the order - this is important in developing parallel applications. The lack of commutativity means that the order of compositing is important, and we shall see that this problem has proved a significant computational geometry challenge in volume rendering.

4.2.2. Different Approaches

Volume rendering techniques can be broadly classified into two approaches: image order and object order. In the *image order* approach (also called *backward rendering*), we process from the image plane to the volume. In the *object order* approach (also called *forward rendering*), we process from volume to image. Note however that as the subject has developed, this broad classification is now less distinct, as hybrid methods have evolved to take advantage of both approaches.

The classical image order method is ray casting, and the seminal paper is that of Levoy [41]. The Levoy paper is a realisation of the volume rendering integration described in the previous section. The order of compositing is back-to-front. There are attractions however in doing the compositing front-to-back, even though there is the extra work of accumulating opacity. This extra work proves very useful because once the accumulated opacity value reaches a threshold there is no point in continuing and so work is saved - this is known as *early ray termination*.

In section 4.3 we look at recent improvements to image order methods. These fall into a number of categories:

Volume Rendering Equation The classical approach computes the volume rendering integral as a Riemann sum. Greater *accuracy* can be achieved by working harder on the integration; greater *speed* can be achieved by ignoring the composition of samples entirely, and simply locating the maximum intensity along the ray and using that value for I - this is known as *maximum intensity projection*.

Interpolation The classical approach requires the calculation of colour and opacity values at sample locations that will generally not coincide with data points - so interpolation is needed. Recent work has explored the options of interpolating before classifying, against interpolating after classifying.

Curvilinear and Unstructured Meshes The classical approach of Levoy was targetted at medical data which occurs typically on rectilinear grids. Recent work has looked at approaches for curvilinear grids and unstructured grids, both of which occur routinely in CFD applications. Indeed unstructured data is of growing importance in medical applications through hand-held ultrasound scanners.

Fast Traversal A time consuming aspect of the Levoy method is the traversal of rays through the volume dataset, and recent work has looked to optimize this traversal.

Hardware Advances A major advance in the last year has been the development of commercial hardware for volume rendering, allowing real time volume rendering on a PC platform. This has suddenly made volume rendering affordable.

The object order approach is characterised by the *splatting* technique proposed by Westover [78], [79]. This essentially projects voxels onto the image plane, forming so-called splats, and composites the splats in the image plane. As Mueller et al [51] explain, splatting essentially re-orders the volume rendering integral so that each voxel's contribution is separated out.

The algorithm works as follows. Find the face of the volume nearest the observer, and consider the volume as a set of slices parallel to that face. Order the voxels within a slice in terms of distance to observer, nearest first. Classify and shade each voxel. Now project each voxel in turn into image space, using a circular Gaussian filter to determine the coverage of the splat in the image plane. The projection of the kernel into the image plane can be pre-calculated and this gives the method its speed - the projection is called a *footprint*. The colour and opacity values are blended into the image buffer at every pixel that falls within the footprint - the values scaled by the value of the Gaussian at the particular pixel. The blending is done using the usual compositing rules.

In section 4.4 we look at recent improvements to object order methods. These fall into a number of categories:

Better Splatting In a series of papers, researchers at Ohio have steadily improved the original Westover splatting method.

Shear Warp Rendering A hybrid approach, known as shear-warp rendering, has become increasingly popular. It achieves its speed by first aligning (using a shear) the volume and the viewing direction so that a line of voxels can project directly to a pixel, and secondly compensating for the first transformation by an image warp transformation.

Unstructured Grids As in the image order approach, researchers have looked at efficient techniques for unstructured grids.

Texture Mapping The emergence of texture mapping hardware has fostered a new object order approach in which the volume is seen as a 3D texture, and slices are projected and composited using this hardware.

4.3. Advances in Image-based Techniques

4.3.1. Volume Rendering Equation

The classical volume rendering equation 9 is a complex integral which cannot be evaluated analytically. The usual approach is to compute numerically using Riemann sums, leading to the simple formulation of equation 10. Novins and Arvo [57] experimented with other numerical integration methods, including the trapezoidal and Simpson's rule. More recently, Jung et al [35] showed how a semi-analytical solution can be found. They use a numerical approximation to the exponential term in equation 9 and replace the term $g(s)$ by the trilinear interpolant. Within any cell this is a polynomial and so can be integrated exactly. They are able to show that this approach enables fine detail to be observed that is not apparent in the classical Levoy discrete approach.

For some applications, the effort of compositing samples is simply not worth it. For example, in angiography, the visualization requirement is to highlight blood vessels in the volume and perfectly good results can be obtained by locating the maximum intensity along the ray and using that intensity value as I. This Maximum Intensity Projection (MIP) approach has been studied by a number of authors recently, with a key aim being to traverse the data quickly to reach the significant cells. For example, if the datavalues at the vertices of a cell are each less than the current maximum, then there is no need to calculate any sample inside the cell using trilinear interpolation - the resulting value will be automatically bounded by the vertex values. There is a nice description of this technique in Heidrich et al [29]. Parker et al [60] extend their fast ray casting of isosurfaces to provide a fast MIP algorithm for shared memory architectures.

4.3.2. Interpolation

There are many trade-offs in volume rendering that are often hard to resolve. An interesting issue involves whether to classify then interpolate, or to interpolate then classify, when calculating the colour and opacity at a sample point. This is discussed in detail by Gasparakis [23], following on from discussion in Lichtenbelt et al [43]. Gasparakis concludes that a smoother image results from first classification then interpolation. However care is needed in the interpolation. The classical approach is to trilinearly interpolate colour values in order to get the colour of a sample to use in equation 16. Wittenbrink et al [83] show that this can produce unexpected effects: suppose one vertex value has red colour, but zero opacity (ie is transparent). This is included in the colour interpolation process, even though its contribution is null when a ray passes directly through it. This is corrected if the colours used in the trilinear interpolation are weighted by their opacity - this is known as *opacity-weighted colour interpolation*. Gasparakis [23] gives a rigorous proof that this is the correct way to carry out the interpolation.

Other authors favour interpolation then classification, see for example Lichtenbelt et al [43]. The argument for this order is that fine detail within a voxel can sometimes be picked out. On the other hand, since the sample points are view-dependent (points along a set of rays in direction of view), the classification itself is then view dependent. Indeed the classification process moves from being a pre-processing step to being a necessary step in each rendering. Furthermore the shading calculation is carried out at the sample point, not the

vertex - thus the approach is analogous to Phong shading, rather than Gouraud. Note that an interpolation of the gradient is also needed. For a discussion of how to do this, see the papers by Moller et al [50] and Bentum et al [6].

Wittenbrink et al [83] give a thorough comparison of the above issue.

Some of the highest quality medical volume rendering has been carried out in Hamburg by Hohne and his research group. They have created detailed segmentation of medical images to create an anatomical atlas. We include mention of them in this section through their recent work on accurate identification of partial volume effects in medical volume rendering. See the paper by Tiede et al [73] for this work, and for pointers to other work by this group.

4.3.3. Curvilinear and Unstructured Grids

Many problems in computational science involve non-rectilinear grids, and there has been much recent work on extending the classical ray casting approach to handle these grids.

The two common types are: curvilinear grids and unstructured grids. One option of course is to resample the data onto a rectilinear grid and use the classical approach. This is unsatisfactory for many reasons: the grid will reflect the nature of the problem and a very fine rectilinear grid may be needed to capture all the detail. Curvilinear grids can be converted to unstructured, by decomposing each hexahedral cell into five tetrahedra, but this then destroys the connectivity implicit in the curvilinear grid. Hence special methods both for curvilinear and unstructured grids have emerged.

For curvilinear grids, there is a mapping from the curvilinear grid in *physical* space (P-space) to a corresponding rectilinear grid in *computational* space (C-space). Fruhauf [22] shows how it is possible to traverse and interpolate along rays in C-space and transform back to P-space for rendering using the Jacobian matrix. Hong and Kaufman [32] argue that there is a loss of accuracy in this process.

Hong and Kaufman [32] themselves propose a different approach. The basic algorithm for ray casting into a curvilinear volume is presented as follows:

1. Cast ray from pixel (x,y)
2. Find first intersection with a cell-face
3. Repeat
   a. Find exit cell face, and exit point
   b. Interpolate to get value s at exit point
   c. Accumulate colour and opacity using s and depth of cell

The repetition terminates when the opacity is 1 or the ray leaves grid. The key to Hong and Kaufman's work is to project cell faces onto the image plane in order to reduce the complexity of the algorithm. The exterior faces are projected and bucket-sorted to get a depth ordering. Each hexahedral cell is described as a set of 12 bounding triangles, two per face - given the entry triangle, the determination of the exit triangle is also accelerated by projection of the 11 candidates onto the image plane. We shall see later in this section other instances of complexity of a 3D problem being handled by reduction to 2D.

For unstructured grids, the major issue is again computational complexity. Whereas for rectilinear grids we know the ordering of cells along a ray and thus can process them without sorting, this is not the case for unstructured grids. The challenge for ray casting unstructured grids is to identify the cells which are intersected by a ray, and order these front-to-back so that compositing can be carried out. The naive approach is to compute (for an *NxN* image and a mesh with n edges) the intersection of all N^2 rays with all $\mathcal{O}(n)$ facets, and sort the intersections along each ray. This has complexity upper bound of $\mathcal{O}(N^2 n \log n)$. There has been significant progress in recent years on lowering this complexity bound.

The seminal paper was by Giertsen [25] who introduced the idea of a *sweep plane*. Imagine a viewing co-ordinate system in which the *xy*-plane is the display, with the *y*-axis in an up direction, and rays being fired into the volume parallel to the *z*-axis direction. Now imagine a scanline on the display, with a given *y*-value, and consider a plane through this scanline and in the viewing direction (ie orthogonal to the display plane). This is a sweep plane, and for a parallel projection will contain all the rays through the scan line. If we can find the intersection of the cells with this sweep plane, then we have reduced the sorting problem from 3D to 2D, with corresponding saving in complexity. Giertsen transforms the vertices of the mesh to the standard viewing system, orders them by *y*-value and then proceeds scanline-by-scanline, maintaining an active set of cells intersected by the current sweep plane. By exploiting coherence between scanlines, Giertsen is able to make efficiency gains.

This basic idea has been developed throughout the last decade with steady improvements in efficiency. In a series of papers, Silva and colleagues have worked to improve the complexity measure mentioned earlier. The *Lazy Sweep Ray Casting Algorithm* [68] avoids some of the transformation and sorting required by the original Giertsen method. This achieves a worst case upper bound of $\mathcal{O}(k + n + n\log n + Nn\log n)$ where $k = \mathcal{O}(N^2 n)$ is the size of output (ie number of facets crossed by all rays).

Westermann and Ertl [77] exploit polygon rendering hardware to construct the projection of cells onto the sweep plane (taking a view from above the volume and using clipping to isolate the correct cells). This is stored in a buffer, with the cells in line of sight order. In a second pass, the buffer is traversed in order to carry out the volume rendering integration.

4.3.4. Fast Traversal

Performance of volume rendering by ray casting has been an active area of research throughout the last decade as the search has gone on to bring the process down to interactive speed.

An early development was *template-based* ray traversal proposed by Yagel et al [86]. This work is based on the observation that, with parallel rays, the path through the voxel structure is common to all rays, ie a template for the path can be used.

Another approach is to pre-process the volume so that the regions of significance are identified, and volume rendering integration is begun from the intersection with the first object rather than the intersection with the volume boundary. This can be done by using a hierarchical data structure such as octrees (used by Parker et al [59], [60] and by Levoy [42]). An alternative, suggested by Wan et al [74], is to record the boundary voxels in the volume, and to project these cells onto the image plane, storing information in two projection buffers - one giving the nearest distance to a boundary cell, the other the furthest distance, for each pixel. In the ray casting process, only rays which intersect boundaries need be considered, and for those which do intersect, the traversal need only proceed from nearest boundary to furthest boundary. In a further paper [75], the authors show how the algorithm may be easily parallelised and they report 20 frames per second rendering on a 16 processor SGI Challenge, for a 256x256x225 dataset and 256x256 grey scale image.

4.3.5. Hardware Advances

The holy grail of volume rendering for many years has been real-time, interactive visualization of moderate-sized datasets. Kaufman over many years has led the research into hardware architectures which seek this prize. His series of Cube architectures progressively got closer to the holy grail, culminating in the Cube-4 architecture described by Pfister and Kaufman [62]. Research on volume rendering hardware has also been very active in Germany, where two machines have been implemented: VIRIM described by Guenther et al [26] and VIZARD described by Knittel and Strasser [38]. All follow the ray casting approach. For a general overview, see the paper by Ray et al [64].

There has been a very exciting development in the last year, showing the holy grail to be fully in sight. Mitsubishi Electric have enhanced the Cube-4 architecture, and created a commercial product, as a PC-board which can fit into a standard PC. The *VolumePro* system is fully described in a landmark paper by Pfister et al [61]. The system is based on a ray casting approach but with many important acceleration techniques, such as the use of shear-warp transformations to improve speed of data access (see later).

4.4. Advances in Object-based Techniques

4.4.1. Better Splatting

The research group at Ohio, and others, have continued throughout the last decade to improve the Westover splatting algorithm. Recent work has addressed a perceived weakness of splatting in that the image appears rather blurred. This problem can be traced to the issue of shading before or after interpolation. The original splatting algorithm shades, then interpolates, which has a smoothing effect. Mueller et al [51] re-order the splatting computation so that shading takes place after interpolation. In this way edges are more clearly picked out.

In another recent paper, Mueller et al [52] continue development of a sheet-buffer approach, in which all the splats from one slice are added into a colour and opacity sheet buffer - and then the resulting sheets are composited either back-to-front, or front-to-back. This gives a smoother appearance in animated viewing. The original sheet buffer proposed by Westover [79] used axis-aligned sheets, but this new work shows improved results are achieved by working with sheets aligned with the image plane.

4.4.2. Shear Warp Rendering

The idea of shear warp rendering is to pre-transform the data into an orientation from which rendering can be fast. Based on earlier work by, among others, Cameron and Undrill [8], the idea was developed into a very effective algorithm by Lacroute and Levoy [40]. It is illustrated in Figure 9. Instead of projecting voxels from the volume at an angle to the image plane, we shear the volume by translating each slice, and resampling along the direction shown. Projection is now trivial: the slices are composited front-to-back using the standard operation, creating an intermediate image. This intermediate image is then corrected by an affine 2D warp. The algorithm exploits coherence in the volume data by using run-length encoding, and the overall algorithm has proved highly competitive.

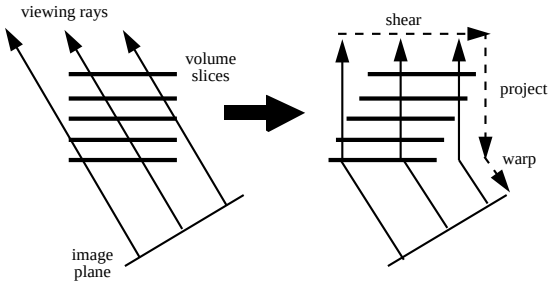

Figure 9: *Shear Warp*

The algorithm can be parallelised and analysis of its performance is covered in the paper by Lacroute [39].

4.4.3. Unstructured Grids

As mentioned earlier, there are many applications where the data is on an unstructured grid. A common approach is to project cells on to the image plane, and for all pixels covered by the projection, a compositing operation is performed to build the image. Because compositing is not commutative, the cells need to be ordered before projection and this is a challenging computational geometry problem. A series of papers have developed progressively better sorting methods. A seminal paper was published by Williams [82]. An important recent paper by Williams et al [81] describes a highly accurate rendering system based on this approach, and using a detailed treatment of optical models within the volume rendering integral.

Research continues to try to improve the original Williams algorithm. For example, see the paper by Silva et al [69].

4.4.4. Texture Mapping Techniques

Traditional approaches to volume visualization are very expensive to perform with respect to cpu usage. To improve this situation specialist hardware has been developed to accelerate many of the standard operations, but it is expensive and not generally found on the desktop. For this reason an alternative approach has been devised that attemps to make use of more generic computer equipment, using hardware acceleration where possible. Modern graphics workstations and PCs now come with texture mapping hardware, for acceleration of 2D and 3D texture operations, as part of the basic graphics system. Work has been presented by Cabral et al [7] that represents the volume using 2D or 3D texture maps exploiting any available hardware.

2D Texture Mapping

If a machine has only 2D texture mapping hardware then this can be used for volume visualization of regular structured data if the data is prepared in the right manner. Slices are taken through the volume orthogonal to each of the principal axes and the resulting information for each slice is represented as a 2D texture which is then pasted onto a rectangle of the same size (see Figure 10). The texture hardware can then be used to quickly manipulate the 2D slices, doing the appropriate bilinear interpolations, when the viewpoint is changed. The reason for making 3 sets of slices is that if just one set is taken, say in the *xy*-plane, as the viewpoint is moved around to the *yz*-plane then the user will be looking directly along the slice and hence it will seem to disappear. Using the 3 sets of slices means that this cannot happen since, in the example above, the *yz*-plane would be directly in the viewing direction. There is one further consideration, however, with the three sets of orthogonal slices visible, the ends of slices not aligned to the view direction cause visual clutter. To remedy this only the set of slices most aligned to the viewing direction are "turned on", the other two sets are invisible. As the viewpoint moves the underlying system selects the appropriate slices to be visible. Transparency is used to make internal information visible.

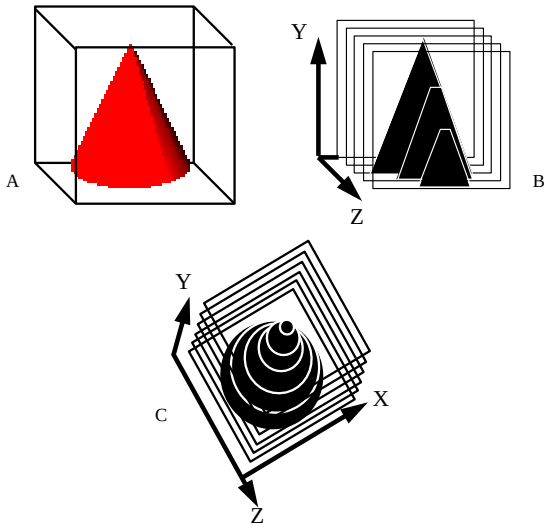

Figure 10: *Using 2D textures for Volume Rendering: A) Represents a volume of binary data containing a solid cone; b) shows the volume being sliced in the XY plane with a texture map being pasted onto each rectangle; c) shows the same process, but in the XZ plane*

A nice application of this approach to volume rendering is given by Hendin et al [30]. They describe a web-based visualization system for medical imaging,using a combination of VRML and Java. It also allows the incorporation of isosurface geometry within the displayed volume.

OpenGL Volumizer

Silicon Graphics have developed an API for volume visualization, called Volumizer, which uses the techniques descibed above. It provides a set of classes that allow an application to query the underlying hardware at run time to get optimal parameters for best performance. The user's data is stored as a set of 3D tiles called bricks which are sized to a power of 2 depending on the amount and type of texture memory available. If the data is not naturally a power of 2 in size then the user can pad the data to fit before loading, or the system offers 3 options (some of which are only available on certain hardware). It will either truncate the data to the nearest power of 2 below the data size (e.g a 34^3 volume would be clipped to a 32^3 volume), augment the data to the next size up (64^3) or create bricks of different sizes (32^3 plus a number of 2^3). This allows for maximum performance when paging them into memory. The system represents the volume as a series of texture mapped polygons, the advantage being that it allows the combination of both volume data and geometricaly represented objects in the same scene. Volumizer uses a 3D geometric representation made up from

tetrahedra and/or pyramids to define the region of the volume that will be visible. The system clips the texture polygons to the bounds of the tetrahedra (and/or pyramids). For example, to volume render a cube of data, 5 tetrahedra could be defined that cover the space of the entire volume and the system would clip the textured polygons to the 5 tetrahedra. Using a geometric description allows applications to create shapes that are not necessarily cubic, it can create doughnut shapes for instance that leaves a hole through the centre of the volume due to the textured polygons being clipped. Material properties such as colour and opacity can be added before or after the texture volume has been loaded using lookup tables. It is more interactive to use a post lookup table, but its availability is hardware dependent.

5. Conclusions

The subject of volume visualization has come a long way over the past fifteen years. The main approaches of isosurface rendering through marching cubes, and volume rendering through ray casting and splatting were all crystallised during the late 1980s, but the past decade has seen these approaches develop a maturity - in terms of robustness, accuracy and performance.

There are strong advocates of the surface extraction approach, arguing that it gives excellent definition of features within a dataset, and exploits polygon rendering hardware to give fast performance. There are equally strong voices arguing for the volume rendering approach, and their case is strengthened by the new hardware developments. There is a useful comparison of the two approaches by Bartz and Meissner [4]. The truth is probably that both approaches are useful and the winner in all the competition between researchers is the user - who now has a battery of very powerful techniques to apply to any volume visualization problem.

Acknowledgements

Thanks to Adriano Lopes for Figure 5 and Figure 4.

References

1. L. Arge and J.S. Vitter. Optimal interval management in external memory. In *Proceedings of IEEE Foundations of Computer Science*, pages 560–569. 1996.
2. Michael Bailey. Manufacturing isovolumes. In Min Chen, Arie E. Kaufman, and Roni Yagel, editors, *Volume Graphics*, pages 79–93. Springer, 2000.
3. C.L. Bajaj, V. Pascucci, and D.R. Schikore. Accelerated isocontouring of scalar fields. In Chandrajit Bajaj, editor, *Data Visualization Techniques*, pages 31–48. Wiley, 1999.
4. Dirk Bartz and Michael Meissner. Voxels versus polygons: A comparative study for volume graphics. In Min Chen, Arie E. Kaufman, and Roni Yagel, editors, *Volume Graphics*, pages 171–184. Springer, 2000.
5. J. Bentley. Multidimensional binary search trees used for associative search. *Communications of the ACM*, 18(9):509–516, 1975.
6. M.J. Bentum, B.B.A. Lichtenbelt, and T. Malzbender. Frequency analysis of gradient estimators in volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 2(3):242–254, 1996.
7. B. Cabral, N. Cam, and J. Foran. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *Proceedings of the ACM/IEEE Symposium on Volume Visualization*, pages 91–98. IEEE Press, 1994.
8. G.G. Cameron and P.E. Undrill. Rendering volumetric medical image data on a SIMD- architecture computer. In *Proceedings of 3rd Eurographics workshop on rendering*, pages 135–145. 1992.
9. E.V. Chernyaev. Marching cubes 33 : Construction of topologically correct isosurfaces. Technical Report CN/95-17, CERN, 1995. Available as http://wwwinfo.cern.ch/asdoc/psdir/mc.ps.gz.
10. Y. Chiang and C. Silva. I/o optimal isosurface extraction. In *Proceedings IEEE Visualization 97*, pages 293–300. IEEE Press, 1997.
11. Y. Chiang, C. Silva, and W. Schroeder. Interactive out-of-core isosurface extraction. In *Proceedings IEEE Visualization 98*, pages 167–174. IEEE Press, 1998.
12. P. Cignoni, F. Ganovelli, C. Montani, and R. Scopigno. Reconstruction of topologically correct and adaptive trilinear surfaces. *Computers and Graphics*, 24(3):399–418, 2000.
13. P. Cignoni, P. Marino, C. Montani, E.Puppo, and R.Scopigno. Speeding up isosurface extraction using interval trees. *IEEE Transactions on Visualization and Computer Graphics*, 3(2):158–170, 1997.
14. Daniel Cohen-Or, Arie Kadosh, David Levin, and Roni Yagel. Smooth boundary surfaces from 3d datasets. In Min Chen, Arie E. Kaufman, and Roni Yagel, editors, *Volume Graphics*, pages 71–78. Springer, 2000.
15. P. Criscione, C. Montani, R. Scateni, and R. Scopigno. Discmc: An interactive system for fast fitting isosurfaces on volume data. In *Proceedings of Virtual Environments and Scientific Visualization '96*, pages 178–190. Springer Wien, 1996.
16. A. Doi and A. Koide. An efficient method of triangulating equi-valued surfaces by using tetrahedral cells. *IEICE Trans Commun. Elec. Inf. Syst.*, E-74(1):214–224, 1991.
17. Martin Durst. Letters: Additional reference to marching cubes. *Computer Graphics*, 22(2):72–73, 1988.

18. H. Edelsbrunner. Dynamic data structures for orthogonal intersection queries. Technical Report F59, Inst Informationsverarb, Tech. Univ. Graz, 1980.

19. T. T. Elvins. A survey of algorithms for volume visualization. *Computer Graphics*, 26(3):194–201, 1992.

20. K. Engel, R. Grosso, and Th. Ertl. Progressive isosurfaces on the web. In *Late Breaking Hot Topics, IEEE Visualization 98*. 1998.

21. K. Engel, R. Westermann, and T. Ertl. Isosurface extraction techniques for web-based volume visualization. In *Proceedings of IEEE Visualization 99*, pages 139–146. 1999.

22. T. Fruhauf. Ray casting of nonregularly structured volume data. *Computer Graphics Forum*, 13(3):294–303, 1994.

23. C. Gasparakis. Multi-resolution multi-field ray tracing: A mathematical overview. In *Proceedings of IEEE Visualization99*. ACM Press, 1999.

24. Thomas Gerstner and Martin Rumpf. Multiresolutional parallel isosurface extraction based on tetrahedral bisection. In Min Chen, Arie E. Kaufman, and Roni Yagel, editors, *Volume Graphics*, pages 267–278. Springer, 2000.

25. C. Giertsen. Volume visualization of sparse irregular meshes. *IEEE Computer Graphics and Applications*, 12(2):40–48, 1992.

26. T. Guenther, C. Poliwoda, C. Reinhard, J. Hesser, R. Maenner, H-P. Meinzer, and H.-J. Bauer. VIRIM- a massively parallel processor for real-time volume visualization in medicine. In *Proceedings of the 9th Eurographics Workshop on Graphics Hardware*, pages 103–108. 1994.

27. R.B. Haber and D.A. McNabb. Visualization idioms : A conceptual model for scientific visualization systems. In B. Shriver G.M. Nielson and L.J. Rosenblum, editors, *Visualization in Scientific Computing*, pages 74–93. IEEE, 1990.

28. Bernd Hamann, Isaac J. Trotts, and Gerald E. Farin. On approximating contours of the piecewise trilinear interpolant using triangular rational-quadratic Bezier patches. *IEEE Transactions on Visualization and Computer Graphics*, 3(3):215–227, 1997.

29. Wolfgang Heidrich, Michael McCool, and John Stevens. Interactive maximum projection volume rendering. In Gregory M. Nielson and Deborah Silver, editors, *Proceedings of IEEE Visualization95*, pages 11–18. IEEE Computer Society Press, 1995.

30. Ofer Hendin, Nigel W. John, and Ofer Shochet. Medical volume rendering over the WWW using VRML and JAVA. In *Proceedings of Medicine Meets Virtual Reality 1998*. 1998.

31. S. Hill. Tri-linear interpolation. In P. S. Heckbert, editor, *Graphics Gems IV*, pages 521–525. AP Professional, 1994.

32. Lichan Hong and Arie E. Kaufman. Accelerated ray casting for curvilinear volumes. In *Proceedings of IEEE Visualization 98*, pages 247–254. 1998.

33. T. Itoh and K. Koyamada. Automatic isosurface propagation using an extrema graph and sorted boundary cells lists. *IEEE Transactions on Visualization and Computer Graphics*, 1(4):319–327, 1995.

34. M.W. Jones and M. Chen. Fast cutting operations on three dimensional volume datasets. In *Visualization in Scientific Computing*, pages 1–8. Springer-Verlag, 1995.

35. Moon-Ryul Jung, Hyunwoo Park, and Doowon Paik. An analytical ray casting of volume data. In *Pacific Graphics 98*, pages 79–86. 1998.

36. J.T. Kajiya and B.P. Von Herzen. Ray tracing volume densities. In *Proceedings of SIGGRAPH 84*, pages 165–174. 1984.

37. Gordon Kindlmann and James W. Durkin. Semi-automatic generation of transfer functions for direct volume rendering. In *Proceedings of 1998 Symposium on Volume Visualization*, pages 79–86. ACM SIGGRAPH, 1998.

38. G. Knittel and W. Strasser. VIZARD - visualization accelerator for real-time display. In *Proceedings of the SIGGRAPH/Eurographics Workshop on Graphics Hardware*, pages 139–146. 1997.

39. P. Lacroute. Analysis of a parallel volume rendering system based on the shear-warp factorisation. *IEEE Transactions on Visualization and Computer Graphics*, 2(3):218–231, 1996.

40. P. Lacroute and M. Levoy. Fast volume rendering using a shear-warp transformation of the viewing transformation. In *Proceedings of SIGGRAPH 94*, pages 451–458. 1994.

41. M. Levoy. Display of surfaces from volume data. *IEEE Computer Graphics and Applications*, 8(3):29–37, 1988.

42. M. Levoy. Efficient ray tracing of volume data. *ACM Transactions on Graphics*, 9(3):245–261, 1990.

43. Barthold Lichtenbelt, Randy Crane, and Shaz Naqvi. *Introduction to Volume Rendering*. Hewlett-Packard Professional Books, 1998.

44. Y. Livnat, H. Shen, and C. Johnson. A near optimal isosurface extraction algorithm using the span space. *IEEE Transactions on Visualization and Computer Graphics*, 2(1):73–84, 1996.

45. S. Lombeyda and M. Rajan. Simple parallel isosurface calculation and rendering of large data sets. In *HPC users group conference March 99*. 1999. See http://www.erpnews.com/conference/iworks99/iwrkhpc.html.

46. Adriano Lopes. *Accuracy in scientific visualization*. PhD thesis, University of Leeds, Leeds, UK, 1999.

47. Adriano Lopes and Ken Brodlie. Accuracy in contour drawing. In *Eurographics UK 98 Conference Proceedings*. 1998.

48. W.E. Lorensen and H.E. Cline. Marching cubes : A high resolution 3d surface reconstruction algorithm. *Computer Graphics*, 21(4):163–169, 1987.

49. Nelson Max. Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 1(2):99–108, 1995.

50. T. Moller, R.K. Machiraju, K. Mueller, and R. Yagel. A comparison of normal estimation schemes. In *Proceedings of IEEE Visualization 97*, pages 19–26. 1997.

51. Klaus Mueller, Torsten Moller, and Roger Crawfis. Splatting without the blur. In David Ebert, Markus Gross, and Bernd Hamann, editors, *Proceedings of IEEE Visualization 99*, pages 363–370. ACM Press, 1999.

52. Klaus Mueller, Naeem Shareef, Jian Huang, and Roger Crawfis. High-quality splatting on rectilinear grids with efficient culling of occluded voxels. *IEEE Transactions on Visualization and Computer Graphics*, 5(2):116–134, 1999.

53. B Narajan. On generating topologically consistent isosurfaces from uniform samples. *The Visual Computer*, 11:52–62, 1994.

54. G.M. Nielson. Scattered data modelling. *IEEE Computer Graphics and Applications*, 13(1):60–70, 1993.

55. G.M. Nielson and B. Hamann. The asymptotic decider: Resolving the ambiguity in marching cubes. In *Proceedings of IEEE Visualization92*, pages 83–91. IEEE, 1991.

56. Gregory M. Nielson. Volume modelling. In Min Chen, Arie E. Kaufman, and Roni Yagel, editors, *Volume Graphics*, pages 29–48. Springer, 2000.

57. Kevin Novins and James Arvo. Controlled precision volume integration. In *1992 Workshop on Volume Visualization*, pages 83–89. ACM SIGGRAPH, 1992.

58. J. Painter, H. Bunge, and Y. Livnat. Case study: Mantle convection visualization on the cray t3d. In *Proceedings of Visualization 96*, pages 409–412. 1996.

59. S. Parker, P. Shirley, Y. Livnat, C. Hansen, and P-P. Sloan. Interactive ray tracing for isosurface rendering. In David Ebert, Hans Hagen, and Holly Rushmeier, editors, *Proceedings of Visualization '98*, pages 233–238. ACM Press, 1998.

60. Steven Parker, Michael Parker, Yarden Livnat, Peter-Pike Sloan, Charles Hansen, and Peter Shirley. Interactive ray tracing for volume visualization. *IEEE Transactions on Visualization and Computer Graphics*, 5(3):238–250, 1999.

61. Hanspeter Pfister, Jan Hardenbergh, Jim Knittel, Hugh Lauer, and Larry Seiler. The VolumePro real-time ray casting system. In *Proceddings of SIGGRAPH99*, pages 251–260. ACM Press, 1999.

62. Hanspeter Pfister and Arie Kaufman. Cube 4 - a scalable architecture for real-time volume rendering. In *1996 ACM/IEEE Symposium on Volume Visualization*, pages 47–54. 1996.

63. T. Porter and T. Duff. Compositing digital images. *Computer Graphics*, 18(3):253–259, 1984.

64. Harvey Ray, Hanspeter Pfister, Deborah Silver, and Todd A. Cook. Ray casting architectures for volume visualization. *IEEE Transactions on Visualization and Computer Graphics*, 5(3):210–223, 1999.

65. Han-Wei Shen. Isosurface extraction in time-varying fields using a temporal hierarchical index tree. In David Ebert, Hans Hagen, and Holly Rushmeier, editors, *Proceedings of Visualization '98*, pages 159–164. ACM Press, 1998.

66. H.W. Shen, C.D. Hansen, Y. Livnat, and C.R. Johnson. Isosurfacing in span space with utmost efficiency. In *Proceedings IEEE Visualization 96*, pages 287–294. IEEE Press, 1996.

67. Will Shroeder, Ken Martin, and Bill Lorensen. *The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics*. Prentice-Hall, 1996. See also the vtk website: www.kitware.com.

68. Claudio T. Silva and Joseph S.B. Mitchell. The lazy sweep ray casting algorithm for rendering irregular grids. *IEEE Transactions on Visualization and Computer Graphics*, 3(2):142–157, 1997.

69. Claudio T. Silva, Joseph S.B. Mitchell, and Peter L. Williams. An exact interactive time visibility ordering algorithm for polyhedral cell complexes. In *Proceedings of 1998 Symposium on Volume Visualization*, pages 87–94. ACM SIGGRAPH, 1998.

70. D. Speray and S. Kennon. Volume probe: Interactive data exploration on arbitrary grids. *Computer Graphics*, 24(5):5–12, 1990.

71. P. Sulatycke and K. Ghose. A fast multithreaded out-of-core visualization technique. In *IEEE Symposium on Parallel and Distributed Processing 1999*, pages 569–575. 1999.

72. Philip Sutton and Charles D. Hansen. Isosurface extraction in time-varying fields using a temporal branch-on-need tree (T-BON). In David Ebert, Markus Gross, and Bernd Hamann, editors, *Proceedings of IEEE Visualization 99*, pages 147–153. ACM Press, 1999.

73. Ulf Tiede, Thomas Schiemann, and Karl Heinz Hohne. High quality rendering of attributed volume data. In David Ebert, Hans Hagen, and Holly Rushmeier, editors, *Proceedings of Visualization '98*, pages 255–262. ACM Press, 1998.

74. M. Wan, S. Bryson, and A. Kaufman. Boundary cell-based acceleration for ray casting. *Computers and Graphics*, 22(6):715–721, 1998.

75. Ming Wan, Arie Kaufman, and Steve Bryson. High performance presence-accelerated ray casting. In *Proceedings of IEEE Visualization 99*, pages 379–386. 1999.

76. Chris Weigle and David C. Banks. Extracting iso-valued features in 4-dimensional scalar fields. In *Proceedings of 1998 Symposium on Volume Visualization*, pages 103–110. ACM SIGGRAPH, 1998.

77. Rudiger Westermann and Thomas Ertl. The VSBUFFER: Visibility ordering of unstructured volume primitives by polygon drawing. In *Proceedings of IEEE Visualization 97*, pages 35–42. 1997.

78. Lee Westover. Interactive volume rendering. In *1989 Chapel Hill Volume Visualization Workshop*, pages 9–16. 1989.

79. Lee Westover. Footprint evaluation for volume rendering. In *Proceedings of SIGGRAPH 90*, pages 367–376. 1990.

80. J. Wilhelms and A. Van Gelder. Octrees for faster isosurface generation. *ACM Transactions on Graphics*, 11(3):201–227, 1992.

81. Peter L. Williams, Nelson L. Max, and Clifford M. Stein. A high accuracy volume renderer for unstructured data. *IEEE Transactions on Visualization and Computer Graphics*, 4(1):37–54, 1998.

82. P.L. Williams. Visibility ordering meshed polyhedra. *ACM Transactions on Graphics*, 11(2):103–126, 1992.

83. Craig M. Wittenbrink, Thomas Malzbender, and Michael E. Goss. Opacity-weighted color interpolation for volume sampling. In *Proceedings of 1998 Symposium on Volume Visualization*, pages 135–142. ACM SIGGRAPH, 1998.

84. Geoff Wyvill, Craig McPheeters, and Brian Wyvill. Data structure for soft objects. *The Visual Computer*, 2:227–234, 1986.

85. R. Yagel. Efficient techniques for volume rendering of scalar fields. In Chandrajit Bajaj, editor, *Data Visualization Techniques*, pages 15–30. Wiley, 1999.

86. R. Yagel, D. Cohen, and A. Kaufman. Discrete ray tracing. *IEEE Computer Graphics and Applications*, 12(5):19–28, 1992.

87. Y. Zhou, B. Chen, and A. Kaufman. Multiresolution tetrahedral framework for visualizing regular volume data. In *Proceedings of Virtual Environments and Scientific Visualization '97*, pages 135–142. Springer Wien, 1997.