Medical Image Analysis 13 (2009) 245–256

Contents lists available at ScienceDirect

Medical Image Analysis

journal homepage: www.elsevier.com/locate/media

VS: A surface-based system for topological analysis, quantization and visualization of voxel data ☆

Itay Cohen [a], Dan Gordon [b,*]

[a] Intel Development Center, Matam Park, Haifa 31015, Israel
[b] Dept. of Computer Science, University of Haifa, Mt. Carmel, Haifa 31905, Israel

ARTICLE INFO

Article history:
Received 18 February 2008
Received in revised form 3 October 2008
Accepted 13 October 2008
Available online 29 October 2008

Keywords:
Connected components
Isosurface extraction
Marching cubes
Surface connectivity
Topological information
Virtually-solid display
Weaving wall
Voxel-sweep
VS system

ABSTRACT

VS is a simple system consisting of several techniques for various volumetric problems. Based on the marching cubes algorithm, it operates with one space sweep through the voxels and extracts all topological information: detection of all isosurfaces, partitioning the data into connected components on the basis of surface connectivity, and association of surfaces with any internal surfaces to arbitrary levels of nesting. VS extends Baker's "Weaving Wall" method by associating topological cavities with their outer surface, and by using efficient data structures for the voxel traversal and for the connected component detection. Its runtime, on average, is only about 2% more than the speeded-up marching cubes algorithm. VS operates on the original voxels without using the contour tree required by other approaches. Using linear-time preprocessing, VS constructs a data structure which can be utilized for any isovalue or interval of isovalues. An accurate estimate of each component's volume is based on the volume enclosed by the outer surface, minus the volume of any internal cavities. VS enables noise reduction by eliminating components (and cavities) with a small volume. Different components can be rendered with different visual attributes, and cutaway views by arbitrary cut-planes are displayed as if the objects were solid, without adding any new surface patches to match the intersection.

1. Introduction and background

Volume visualization is of prime importance in many industrial, scientific and medical applications. The raw data is usually available as a 3D matrix of numbers which are the discrete values of a density function $F(x,y,z)$, representing some physical property. For example, in computerized tomography, the property is that of X-ray attenuation. In such an application, a user wishes to specify threshold values for some type of material (e.g., bone) and then visualize the shape of the object(s).

Another important objective is to obtain all the topological information from the 3D data. This includes the partitioning of the data into connected components, i.e., the user wishes to distinguish between disconnected parts of the same type. Other topological information that may exist in the data is the presence of cavities inside the objects, and the possible presence of internal objects inside the cavities, etc., to arbitrary levels. Interaction is also important, e.g., the ability to isolate disconnected parts and display them separately or in any combination of parts, or with different visual attributes (such as color and opacity). It is also desirable to have the ability to slice the objects with arbitrary cut-planes in order to better visualize the internals of complex parts. Another useful feature is quantization, i.e., the ability to measure the surface area of each component, as well as the volume enclosed by the surface. The topological data enables an accurate estimate of the volume, since the volume of internal cavities can be subtracted. Such quantization also allows the user to eliminate noise by ignoring "small" components and cavities.

There exist two fundamentally different approaches to visualizing volumetric data: volume-based and surface-based. Volume-based methods operate on the entire set of voxels and display the objects directly according to some method (see Cai and Sakas, 1998; Cohen-Or and Sheffer, 1994; Cohen-Or and Fleishman, 1995; Frieder et al., 1985; Lacroute and Levoy, 1994; Levoy, 1988, 1990; Meagher, 1984; Reynolds et al., 1987; Sobierajski et al., 1993; Yla-Jääski et al., 1991). These methods have a disadvantage of requiring the entire dataset to be in some suitable format, but, on the other hand, the original data is available at all times for various purposes, such as "volume rendering" (Levoy, 1988).

Surface-based methods first extract the surfaces in a preprocessing step, and then display the surfaces (Artzy et al., 1981; Lorensen and Cline, 1987; Wyvill et al., 1986; Gordon and Udupa, 1989; Wilhelms and van Gelder, 1992; Shen and Johnson, 1995; Bajaj et al., 1996; Carneiro et al., 1996; Criscione et al., 1996; Livnat

☆ Preliminary results presented at the Vision, Modeling and Visualization Workshop, November 2004.

* Corresponding author. Tel.: +972 4 8240159; fax: +972 4 8249331.
E-mail addresses: itay.m.cohen@intel.com (I. Cohen), gordon@cs.haifa.ac.il (D. Gordon).

1361-8415/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.media.2008.10.005

et al., 1996; Shen et al., 1996; Kobbelt et al., 2001; Nielson, 2003; Zhang and, 2006). There are advantages and disadvantages of surface-based methods as compared to volume-based methods (see Bartz and Meissner, 1999). However, the amount of surface data is proportional to the surface of the objects while the volume data is proportional to its volume, so surface data usually takes up less space than volume data and it is more easily manipulated and displayed. Furthermore, surface data is more suitable for today's graphics engines, which are geared towards polygonal meshes. Another advantage of surface data is that it is amenable to modern surface-processing algorithms, such as those of (Touma and Gotsman, 1998, 2000).

The boundary-detection (BD) algorithm (Artzy et al., 1981) is probably the earliest topologically correct method of extracting the isosurface of an object. The voxels are considered as cubes with a uniform value inside. When an isovalue is determined, all voxels are categorized either as 0-voxels if their value is below the isovalue or as 1-voxels, otherwise. This approach is often referred to as the *digital model*. Starting from a user-specified seed boundary face, BD detects the (connected) outer surface of the object formed by all voxels which are edge-connected to the seed voxel, either directly, or by a path of connected voxels. BD provides the volume enclosed by the surface, but that volume may include internal cavities, which are not accounted for. Different objects, as well as cavities, require new seeds. BD is a graph traversal algorithm, performing a breadth-first search on the digraph defined by certain relationships between boundary faces. Gordon and Udupa (1989) showed that for objects defined by face-connected voxels, a modified BD achieved significantly faster results. See also (Herman and Webster, 1983) and (Kong and Udupa, 1992) for formal proofs of correctness of BD and the modified BD, respectively.

The marching cubes (MC) algorithm of Lorensen and Cline (1987) considers the voxel values as being assigned to grid points in 3D space, and assumes that at any other point (x,y,z), the value of $F(x,y,z)$ is a trilinear interpolation of the values at the eight grid points surrounding (x,y,z). Given a certain threshold (or isovalue), MC sweeps once through all the volumetric dataset and outputs all triangles defined by the isovalue. Thus, MC produces all the triangles forming the surfaces of all the objects, including the surfaces of any cavities. However, the original MC does not partition the volumetric data into connected components. Certain topological problems with MC were corrected in subsequent papers, such as (Shirley and Tuchman, 1990; Nielson and Hamann, 1991; Chernyaev, 1995; Carneiro et al., 1996; Nielson, 2003). Much research has also been done on speeding up MC through the employment of various data structures–see, for example, (Giles and Haimes, 1990; Wilhelms and van Gelder, 1992; Itoh and Koyamada, 1995; Shen and Johnson, 1995; Shen et al., 1996; Bajaj et al., 1996; Livnat et al., 1996; Cignoni et al., 1997; Sutton et al., 2000).

Wyvill et al. (1986) also assume the trilinear interpolation model, but they extract isosurfaces from seed voxels, so every connected surface requires a different seed. Separating the data into connected components can be done by volume-based methods (see Schiemann et al., 1996): starting from a user-specified seed voxel, all voxels that are connected to it are identified by some classical search method such as breadth-first or depth-first search. Such methods require a new seed for every object. Another problem is that the result is again volumetric, thus requiring large storage space. The display of such data either requires volume-based methods, or a surface extraction step to isolate the surface. Furthermore, internal cavities are not identified by this method.

Udupa and Ajjanagadde (1990), using the digital model, use a recursive algorithm to construct a "containment tree" in which each node corresponds to a surface and its children correspond to internal surfaces. Their method searches the interior of a surface for a boundary face, and then applies the modified BD to produce an internal surface. When an outer surface contains several internal surfaces (cavities), this method requires some method for marking each internal surface that has already been found. To the best of our knowledge, the "Weaving Wall" algorithm of Baker (1989) is the first approach that uses a single space sweep, as in MC, to separate the objects into connected components based on surface connectivity. The purpose of this was aimed at following changes in an animation sequence, but it was also used in (Baker, 1989) for tomographic surface reconstruction.

More recently, much work has been done on topology-based techniques. Bajaj et al. (1997) introduced the use of the *contour spectrum* and the *contour tree*, which captures the topological relationships between all possible isosurfaces. These ideas are based on the Reeb graph, introduced in (Reeb, 1946). The contour tree is a tree (in the graph–theoretic sense), drawn in 2D, whose vertices are either local extrema or saddle-points of the volume function, and the edges connect local extrema to their nearest saddle-point(s). The y-coordinate of every vertex is the function value at that vertex, and the x-coordinate is chosen so that the user can visualize the structure of the tree. Thus, the intersection of a horizontal line $y = y_0$ with the graph edges corresponds exactly to all the separate isosurfaces defined by the isovalue y_0.

The following is a very brief review of some of the work in this area. A detailed review is beyond the scope of this paper, and the reader is referred to the excellent reviews in the papers of Carr and Snoeyink (2003) and Weber et al. (2007). Pascucci and Cole-McLaughlin (2003) improve on previous work of Pascucci by computing the augmented contour tree in $O(n + t \log n)$ time, where n is the number of grid points and t is the size of the contour tree. Their computation also includes the topological type of each surface (its Betti number). Carr and Snoeyink (2003) replace the methodology of providing seeds for every possible surface with *path seeds*, which provide starting points for all surfaces. Their preprocessing time is $O(n \log n + t\alpha(t))$, where α is the inverse Ackermann function (which can be considered as a constant for all practical purposes). The path seeds are used for creating sets of isosurfaces at several different isovalues. Carr et al. (2004) introduce a method for simplifying complex contour trees. For recent results on time-varying Reeb graphs (see Edelsbrunner et al., 2008).

Several papers, such as (Takahashi et al., 2004; Takahashi et al., 2005; Takahashi et al., 2006; Weber et al., 2007) utilize topological information for volumetric rendering. This requires a preprocessing time of $O(n \log n)$. Takahashi et al. (2004) use skeletonization to obtain the contour tree and use it for the design of transfer functions. (In direct volume rendering techniques, transfer functions control visual attributes.) Takahashi et al. (2005) deal with *interval volumes* – these are volumes defined by two isovalues bounding the volumes. Takahashi et al. (2006) provide a means for obtaining nesting information between different volumetric objects. The latter two works reduce high-frequency noise and also provide nesting information. Weber et al. (2007) introduce unique transfer functions to different subvolumes and enable hardware-accelerated volume rendering.

In this paper, we present the "VS" system, which combines several methods for various volumetric problems. VS was developed independently of (Baker, 1989), and it provides the following features:

- VS uses a preprocessing step which requires only linear-time.
- Similarly to MC, VS initially performs just one space sweep through the volume data; this stage is called the *voxel-sweep*.
- As in the Weaving Wall algorithm, during the voxel-sweep, VS detects all surfaces of all the objects defined by the isovalue, and partitions the data into connected components based on surface connectivity.

- The voxel-sweep uses the union-find data structure (Aho et al., 1983) in order to efficiently maintain and merge the separate surface components.
- At the same time, the voxel-sweep also associates any internal cavities with their respective surrounding objects, to arbitrary levels of nesting.
- The runtime of the voxel-sweep is only about 2% more than that of the speeded-up MC.
- The topological data enables an accurate estimate of the volume and surface area of each separate component.
- VS enables noise elimination by simply ignoring components and cavities with a small volume.
- VS provides a technique called the "virtually-solid" display, which displays components and nested surfaces as if they were solid, even when cut by an arbitrary cut-plane. No new surface patches are created for this display.
- Interval volumes can be easily displayed by a simple mapping of the voxel values.
- The voxel-sweep can be easily extended to simultaneously handle several different isovalues.

The triangular mesh format of the surfaces of the components is useful both for modern display architecture and for state-of-the-art compression methods (Touma and Gotsman, 1998; Karni and Gotsman, 2000). The name "VS" is derived from "voxel-sweep" and "virtually-solid" display. A preliminary version of this work appeared in (Cohen and Gordon, 2004).

The rest of the paper is organized as follows. Section 2 explains the entire voxel-sweep stage. Sections 3 and 4 explain the volume estimation, noise reduction, and the "virtually-solid" display. Section 5 presents the results of several test cases, and Section 6 concludes with a discussion and potential extensions.

2. The voxel-sweep stage

2.1. Review of the marching cubes

Using the terminology of MC, we henceforth use the term "voxels" to refer to the grid points with which the discrete values of the data are associated. We assume that the dataset is completely surrounded by 0-valued voxels. MC proceeds by traversing the volume data slice-by-slice. In each slice, it traverses the rows in sequential order, and in each row, it traverses the voxels in sequential order. We refer to a row of voxels as a "voxel-row". There is no need to traverse the first voxel-slice because all its values are zero. In every consecutive slice, each new voxel-row (except the first) forms a sequence of cubes of voxels whose corners belong to the current voxel-row, the previous voxel-row, and two voxel-rows of the previous slice. In each cube of voxels, if any edge spans the isovalue, a surface vertex is determined on that edge by interpolation between the two voxel values of the edge. The isosurface is approximated by triangles whose vertices lie on the edges of the cube. Note that some of these vertices could have been formed while traversing the previous voxel-rows. Fig. 1 shows a cube of eight voxels, the three traversal directions, and a triangle formed by three interpolated values. For precise details of how the triangles are formed (see Lorensen and Cline, 1987).

Assuming that the isosurface is the boundary of a 3D object, we can determine for every surface triangle, the normal pointing *outwards* from the object according to values of the voxels spanning the triangle's vertices. In considering the eight voxels of a subcube, there are 256 possible combinations of voxels either being inside or outside the isosurface; this number can be reduced by using various symmetries. Note that there is a theoretical problem in MC when a voxel value is exactly equal to the threshold value, because the triangle degenerates to a point. This problem is easily resolved by changing the voxel value by some small $\epsilon \neq 0$.

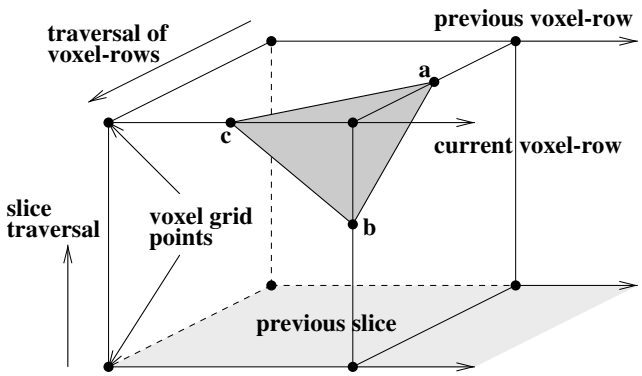

Fig. 1. A cube of voxels, showing the three traversal directions, and a triangle formed by three interpolated values.

The original MC algorithm had a topological flaw which caused occasional holes in the surface representation, and it did not resolve ambiguous cases. As noted earlier, these problems were corrected in subsequent papers, such as (Shirley and Tuchman, 1990; Nielson and Hamann, 1991; Chernyaev, 1995; Carneiro et al., 1996; Nielson, 2003), and the methods of (Chernyaev, 1995; Carneiro et al., 1996) were also tested in the voxel-sweep. The ambiguities of MC can also be resolved by the digital surface approach of (Lachaud and Montanvert, 2000). This method depends on the a priori decision of the user on whether 1-voxels which are edge-connected but not face-connected should be considered as belonging to the same object.

2.2. 2D analogy of the voxel-sweep

In order to explain the voxel-sweep, we first present a 2D analogy. Fig. 2 shows a grid of points at which the values of the function $F(x,y)$ are given. We assume that all the values at the extremal grid points are zero, and that the values inside define the three boundaries shown in the figure. As in MC, the vertices

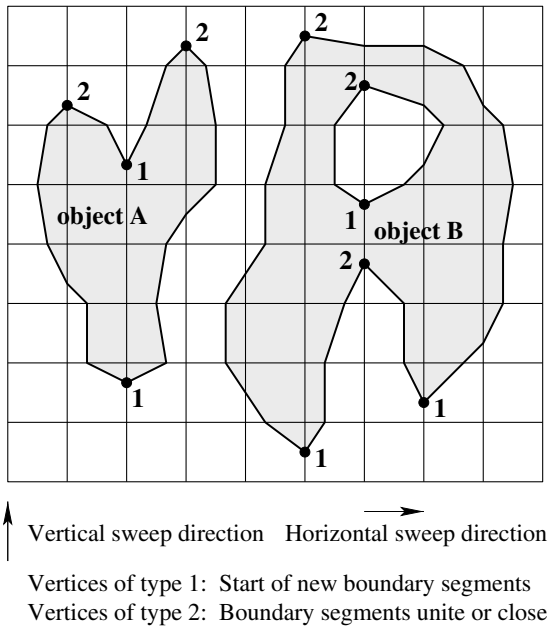

Fig. 2. 2D analogy of the voxel-sweep.

of the boundaries lie between two grid points whose values span the isovalue; the exact position of each boundary vertex is a linear interpolation of the values at the grid points.

We assume that the horizontal sweep direction goes upwards and at each horizontal front, the grid is swept from left to right. As the sweep proceeds upwards, new boundary segments are encountered. In Fig. 2, vertices marked with a "1" denote the start of new boundary segments, and those marked "2" indicate points at which boundary segments unite or close. Initially, three segments, starting from the three lower 1-vertices, are associated with different surfaces. The sweep then encounters a 2-vertex of object *B* which unites two separate segments; this is where the union-find data structure is used. Then, two more 1-vertices are encountered, one in each object. In object *B*, the new segment closes at a 2-vertex, forming an internal cavity. Further up in object *B*, the initial segment closes at the topmost 2-vertex. In object *a*, the new segment unites with the previous one (at the lower 2-vertex), so there is now only one segment, which closes at the higher 2-vertex.

2.3. The 3D voxel-sweep

The difference between MC and the voxel-sweep is that the latter uses a "union-find" (also called "merge-find") data structure (Aho et al., 1983) to keep the connected triangles together in one mesh, and a mechanism for the nesting information. The union-find data structure is applied in the following type of situation: We start out with n objects which are initially placed in n different sets. At all times, the sets are disjoint. After that, we process $O(n)$ operations of the following type:

- Find(a): returns the identity of the (unique) set to which a belongs.
- Union(A,B): unites the sets A and B. The union's identity will be either A or B.

It is well-known that with this data structure one can perform a sequence of $O(n)$ union and find operations on n objects in amortized time of $O(n\alpha(n))$, where $\alpha(n)$ is the inverse Ackermann function. This function grows so slowly that for all practical applications, $\alpha(n) \leqslant 4$; (see Aho et al., 1983, p. 189).

The union-find structure is implemented as a collection of rooted trees, one for each set. Each tree vertex contains the identity of one object; the tree's identity can be simply the identity of the object at its root. The root also contains the number of elements in the tree. Every other tree vertex has one link, pointing towards the root. The Find(a) operation has two stages: first, it traverses the links from the object a to the root and obtains the set's identity. Then, it traverses those same links again, and "collapses" the path by setting all the links to point directly at the root. This collapsing stage is the major factor contributing to the efficiency of the algorithm. The Union(A,B) operation takes $O(1)$ time. It compares the size of the two sets, sets the root of the smaller set to point to the root of the larger set, and updates the size of the unified set.

VS uses the union-find structure during the voxel-sweep stage as follows: Every boundary triangle is considered as an object and connected triangles form one set. Two triangles a, b are considered as *connected* if they either share a common edge, or there is a sequence of triangles $c_1, \ldots, c_k$ such that every triangle in the sequence $a = c_1, \ldots, c_k = b$ (except B) shares a common edge with the next.

When a new triangle a is encountered, it is either adjacent to a previous one, or not. If it is not adjacent to a previous one, then it is placed in a new set. If it is adjacent to a previous triangle b, we first find b's set by executing the operation B = Find(b), and then we set a's link to point directly at B. Occasionally, the new triangle a may be adjacent to more than one previous triangle, say $b_1, \ldots, b_k$, with $k = 2$ or $k = 3$. As before, a's set is determined as $B = \text{Find}(b_1)$. $b_2, \ldots b_k$ are now processed, and their sets are compared with B. If all their sets are equal to B, nothing further is done. However, if for some $2 \leqslant i \leqslant n$, we find that $C = \text{Find}(b_i) \neq B$, then we perform $B = \text{Union}(B, C)$. This operation unites two boundary segments which were considered as separate. If $i < k$, the process continues by comparing $\text{Find}(b_{i+1})$ with B.

Clearly, in testing for adjacent triangles, it is sufficient to consider only triangles that share an edge with the new triangle a, for the following topological reason: for any triangle b that shares only a single vertex with the new triangle, there is a sequence of triangles $b_1, \ldots b_k$ such that b is edge-adjacent to b_1, each b_i is edge-adjacent to b_{i+1}, and b_k is edge-adjacent to a. Therefore, a and b will, at some stage, belong to the same set. At the end of the sweep, all triangles connected to some initial triangle are in one unique set, and each set forms a unique connected component of the isosurfaces.

2.4. Detection and association of cavities

We wish to obtain the following information during the voxel-sweep stage: for every outer surface of an object, we want to have a list of the internal cavities, if any. Furthermore, for every internal cavity, we wish to associate it with any internal objects, and so on. The importance of finding internal cavities in an object is obvious. Internal objects (inside the cavities), if any, are also important for clinical reasons, e.g., consider kidney stones. Also, such internal objects may represent a portion of the outer object which appears disconnected due to sampling and/or discretization errors. VS works in principle for any level of nesting.

Our approach to this problem uses the well-known "parity principle". Assume that you have several objects whose interiors do not intersect, and every connected boundary surface, including cavities and internal objects, has a unique identity, which we assume to be an integer. Suppose you shoot a ray R from the outside in some direction to the end of the dataset, as shown in Fig. 3. Note that all the boundaries have the same isovalue, so as R runs through the dataset, its density values along the ray alternate between being below and above the isovalue. The initial value is below the isovalue because the dataset is surrounded by 0-valued voxels. The points at which R intersects a surface are exactly the triangle vertices lying on R. According to this, a surface B is internal to surface A if and only if R has passed through A an *odd* number of times before it meets B.

However, our problem is complicated by the fact that we do not know ahead of time if two surface segments will eventually unite (as in object A of Fig. 2), or if the one encountered later will eventually close on itself to form a cavity (as in object B of Fig. 2). Our solution to this is to consider rays passing through voxel-rows

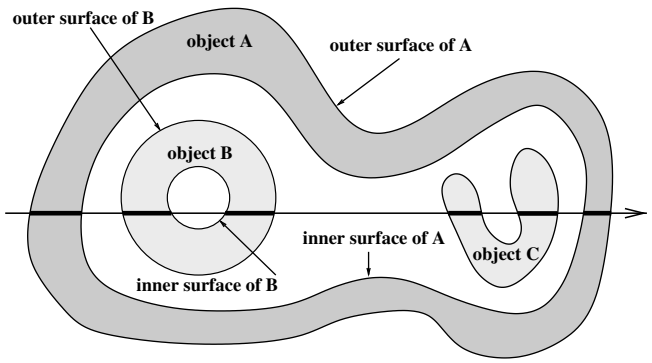

Fig. 3. Ray intersection with inner end outer surfaces.

and to keep track of *potential* pairs of different surfaces, say (A,B), such that one of the following will be true: either A and B will unite at some point and this information will be discarded, or A and B will remain different until the end of the sweep, which means that B is internal to A. Note that although the surface connectivity information is derived from an entire cube of voxels, there is only one new voxel-row in our method of sweeping through the dataset. All the required data from the previous three voxel-rows has already been gathered. This approach is implemented as follows.

During the voxel-sweep, we maintain two dynamic data structures, P and S. P consists of pairs of surfaces (A,B) such that B is a potential candidate for being a interior surface of A. If, during the sweep, A and B unite, then (A,B) is removed from P, so at the end, we will be left only with the relevant pairs of surfaces. P is initialized to empty at the start of the voxel-sweep. Also, P does not hold any redundant transitive information, i.e., if $(A,B),(B,C)\in P$, then $(A,C)\notin P$. S is a stack of surfaces, all of them distinct while on S. The surfaces on S are potential candidates for having an internal surface, and they are characterized by the fact that R has passed through each of them an *odd* number of times; only the last occurrence is kept on S. S is initialized to empty at the start of each row sweep, and at the end, it will be empty because R will pass through each surface an even number of times. Formally, we handle P and S as follows:

When R meets a surface A, perform the following operations:

1. Let $T=\mathrm{Top}(S)$.
2. If $A=T$, pop S.
3. If $A\neq T$, push A onto S and add the pair (T,A) to P.
4. If A unites two existing surfaces, then one of them must be A, and denote the other by B. We now perform the Union operation, and assume w.l.o.g. that $A=\mathrm{Union}(A,B)$. We now perform the following:
   - If $(A,B)\in P$, then:
     - Remove (A,B) from P.
     - If for some C, $(B,C)\in P$, then we also remove it from P (because the united surface appeared an even number of times before C).
   - If (B,A) is in P, we do likewise.
   - A or B or both may appear on S, and each can appear only once. Therefore, we proceed as follows: if only A is on S, we do nothing. If only B is on S, change it to A. If both appear on S, it means that the united surface appears twice, so A and B are removed.

There remains the question of how P and S should be organized in order to minimize the computational overhead. The simplest method of implementing P is to use a 2D boolean array such that position $(A,B)=1$ if $(A,B)\in P$, and 0 otherwise. Denote by P_A the Ath row of the array. When two surfaces A and B unite, then assuming the result is A, we perform the following:

1. We unite their sets of internal surfaces by setting $P_A=P_A\vee P_B$, where $\vee$ denotes the componentwise *or* operation.
2. We set all entries of P_B to 0.
3. We add B to a list of "free" surface identities, to be reused when a new surface is found. New integers are assigned to new surfaces only when this list is empty.

The above operations are in addition to those described previously, and they are specific to this particular data structure for P. Note that the required array size is unknown in advance, so it is determined by some user estimate. If at any time the array becomes too small because of new surfaces, we increase its size by reallocation. It is well-known (and easy to prove) that if the array size is increased by some constant factor every time reallocation is performed, then the time to handle a total of n surfaces is linear in n. The reuse of surface identities minimizes the number of reallocations.

Consider the above operations and how they affect the stack S. S should be organized so that arbitrary elements on it can be removed or modified. This is done by implementing S as a pair of data structures – a linked list for the actual stack, and a pointer array indexed by set indices. Given a set identity A (which is an integer), we use the array to access A's position on the list and remove or modify A as needed. The array can be reallocated similarly to P if necessary.

2.5. Speedup of the voxel-sweep

MC has a time complexity of $O(n)$ where n is the number of voxels, since it traverses all the voxels. This approach is not helpful for real-time applications and an acceleration technique must be applied. Lorensen and Cline (1987) used local coherence in order to save time on geometric computations. This was done by reusing calculations when moving from voxel to voxel, row to row and slice to slice. However, the complexity of MC still remains $O(n)$.

The main class of MC acceleration methods is known as "search structure techniques", which use special data structures in order to skip irrelevant voxels. Of course, setting up the data structures takes some preprocessing time, but since the structures are independent of any particular value of the isosurface, it can speed up the surface detection of any subsequently chosen isovalue. These methods can be divided into three main approaches: space-based, range-based and surface-based.

In space-based techniques, a spatial data structure is introduced to speed us the search for isosurface vertices. Among the structures used are octrees (Wilhelms and van Gelder, 1992), pyramid structures (Criscione et al., 1996), and 3D k-d-trees (Bentley, 1975). In range-based techniques, each cube is identified with the interval it spans in the range of the scalar field, and the range space is searched for intervals containing the isosurface. Various implementations of this approach can be found in (Gallagher, 1991; Shen and Johnson, 1995; Shen et al., 1996; Livnat et al., 1996; Cignoni et al., 1997). Among the data structures used in these works are range trees, k-d-trees and interval trees. In surface-based techniques one first needs to find an "active" cube (or "seed") of the isosurface for each connected component, and then traverse the whole isosurface moving through face/cell adjacencies. The previously-mentioned works (Artzy et al., 1981; Gordon and Udupa, 1989) are of this type, and so is (Speray and Kennon, 1990).

In VS, the acceleration techniques need to support both the connected component feature and the association of internal surfaces with their surrounding surfaces. The first one presents no problem for the various acceleration techniques, but the second feature requires an ordered access to the cubes of each row of cubes, as explained in Section 2.4. This is achieved by using hierarchy of balanced binary trees, which we call the MinMax Tree, organized as follows:

- Every voxel cube is associated with a pair of voxel values, called the *minmax range*, consisting of the minimum and maximum voxel values of the cube. Clearly, for any given isovalue v, if it lies outside the minmax range of a cube, then the cube can be skipped.
- Every row of cubes is associated with a binary tree, called a "row tree", built as follows: every tree node corresponds to a subrow of consecutive cubes, and it also has a minmax range spanning (exactly) the minmax ranges of its associated cubes. Thus, if a given isovalue is outside the tree node's minmax range, we

can skip the entire subrow of cubes. The tree's root corresponds to the entire row of cubes, and its left and right children correspond to the first half and the second half of the cube-row, respectively. This continues recursively down to the leaves, which correspond to pairs of adjacent cubes (assuming that the number of cubes in a row is a power of 2).

- Similarly, for every slice of cubes, we construct a binary tree, called a "slice tree", corresponding to the minmax values of the slice's rows. The root corresponds to the entire slice, and its minmax range is derived from the minmax ranges of all the slice's rows. The root's children correspond the the first and second halves of the slice, and so on.
- Finally, there is one tree (the MinMax tree) corresponding to the entire dataset. Its root will holds the minmax range of all the voxels, and its two children correspond to the first and second halves of the slices, and so on.

Let $N = n^3$ be the total number of cubes, and assume that $n = 2^k$. The construction of the MinMax tree can be done in $O(N)$ time, using $O(N)$ space, and without even using pointers, as follows. Consider a single row of cubes, denoted as cube$[0], \ldots,$ cube$[n-1]$. The tree structure uses an array node$[1], \ldots,$ node$[n-1]$, so that the root is node[1], and the children of node[j] are node[$2j$] and node[$2j+1$] (this is similar to the well-known heap data structure). We assume that each node[j] can hold two voxel values; these will be the minimum and maximum of all the cube values spanned by node[j]. In order to simplify our notation, we denote cube[j] by node[$n+j$]. Let minmax(x,y) denote a function which returns the ordered pair $(\min(x,y), \max(x,y))$. Fig. 4 shows how the row tree is set up; the slice trees and the final MinMax tree are set up similarly.

Given an isovalue v and a row of cubes, we need to determine the sequence of cube indices $0 \leqslant j_1 < \cdots < j_\ell < n$ of cubes such that

```
for i = 1 to k
    for j = n/2^i to n/2^(i-1) − 1
        node[j] = minmax(node[2j], node[2j + 1])
    endfor
endfor
```

Fig. 4. Setting up a row tree over one row of cubes.

```
Input: An isovalue v and a row-tree node[·].
Output: A sorted list of indices of cubes
        intersected by the isosurfaces.
S is a stack of integers, initially empty.
Push 1 on S
while S ≠ ∅
    j = pop(S)
    if (min(node[j]) ≤ v ≤ max(node[j]))
        if (j < n)
            push 2j+1 on S
            push 2j on S
        else add j − n to list of cube indices
        endif
    endif
endwhile
```

Fig. 5. Pseudo-code for getting the list of row cubes intersected by all isosurfaces defined by a given isovalue.

v lies in their minmax range. Essentially, this is done by traversing the row tree node[·] in inorder, but with the added efficiency that if v is not in the minmax range of a node, then the entire subtree rooted at that node is pruned. Fig. 5 shows the pseudo-code for this search. It is clear that the time to find these ℓ intersections is $O(\ell \log n)$. The higher-level trees are handled similarly.

Let s, r, c denote, respectively, the number of voxel-slices, the total number of voxel-rows and the total number of cubes intersected by all the isosurfaces defined by a given isovalue. Clearly, the time to determine the S slices is $O(s \log n)$, the time to determine the R rows is $O(r \log n)$, and the time to determine the c cubes is $O(c \log n)$. However, we have $s \leqslant r \leqslant c$, so the total time is $O(c \log n) = O(c \log N)$.

3. Quantization and noise elimination

Given a triangular mesh representing the surface of a 3D object, we can estimate its surface area by summing the areas of all the triangles. Note that this will be an underestimate, because only the vertices lie on the true surface; most of the triangles are interior to the surface.

We can evaluate the volume of the object by using Gauss' divergence theorem. Lancaster et al. (1992) also use the divergence theorem, but our method differs in some details. Gauss' theorem states if we are given a differentiable vector field $\vec{F}(x,y,z) = (f_1, f_2, f_3)$, where f_i are scalar functions of x, y, z, defined on a compact three-dimensional region R which is simply connected, then

$$\iiint_R \nabla \vec{F} dv = \int\int_S \vec{F} \cdot \vec{N} ds, \tag{1}$$

where $\nabla \vec{F} = \partial f_1/\partial x + \partial f_2/\partial y + \partial f_3/\partial z$ is the gradient of $\vec{F}$, S is the surface of R, $\vec{N}$ is the normal to S (directed outside the volume enclosed by S), and "$\cdot$" is the dot product of two vectors.

Consider now the special case where $\vec{F}$ is taken as $\vec{F}(x,y,z) = (x,y,z)$, so $\nabla \vec{F} = \partial x/\partial x + \partial y/\partial y + \partial z/\partial z = 3$. Substituting into Eq. (1), we get:

$$3 \iiint_R dv = \int\int_S (x,y,z) \cdot \vec{N} ds. \tag{2}$$

The left-hand-side of Eq. (2) is simply 3 times the volume of R, so the volume is

$$V = \frac{1}{3} \int\int_S (x,y,z) \cdot \vec{N} ds. \tag{3}$$

The surface integral of Eq. (3) can be approximated by a Riemann sum based on the surface data. Let n be the number of points which we want to use for sampling the surface. We need to determine a set of points on or near the surface $\vec{p}_i = (x_i, y_i, z_i), 1 \leqslant i \leqslant n$, and associate a normal vector $\vec{N}_i$ and a suitable surface area A_i with each point. The surface area is approximated by

$$V \approx \frac{1}{3} \sum_{i=1}^{n} \vec{p}_i \cdot \vec{N}_i A_i. \tag{4}$$

The approximation data can be chosen in two ways:

1. Consider all the triangles forming the surface, $\vec{p}_i$ are taken as their centroids, $\vec{N}_i$ are the triangle normals, and A_i are the triangle areas.
2. Each $\vec{p}_i$ is a triangle vertex, $\vec{N}_i$ is the average normal of all the triangles containing $\vec{p}_i$, and A_i is one third of the sum of the areas of the triangles containing $\vec{p}_i$.

Both methods will underestimate the true volume because the triangles are mostly interior to the surface; if R is convex, then

all the triangles are interior. This means that the triangle areas underestimate the surface area. However, in the first method, the position of $\vec{p}_i$ is also interior to the surface, while in the second method, $\vec{p}_i$ is on the isosurface. Thus, the second method is more accurate. Note that the average normals required by the second method can be used for Gouraud shading of the surface, so their calculation does not impose an additional computational burden.

Estimating the true volume of an object is now simple: We estimate the volume enclosed by the outer surface and subtract the volume of internal cavities, if any. If there are any objects interior to the cavities, then their consideration depends on the application; if it is judged that their separation from the surrounding object is due to sampling errors, then their volume(s) can be added.

Udupa (1981) obtains an estimate of the volume in the digital model during the boundary-detection stage by choosing some axis, e.g., the x-axis, and using a counter. The counter adds the x-coordinates of boundary faces facing the positive x-direction and subtracts the x-coordinates of faces facing the opposite direction. The result is the total number of cubes in the interior of the boundary. If the surface is an interior one, then one gets a negative result. Hence, the counting method can be applied to any set of surfaces to obtain their net volume, but the determination of interior cavities and their association with the exterior surface must still be done manually. In one slice of voxels, this estimate is equivalent to estimating an integral by the rectangle rule – a method in which the error is $O(h)$, where h is the size of a voxel side. Nyström et al. (2002) also base their method on the digital model, but they use triangles whose vertices lie at the midpoints (instead of at the correct interpolation point). Triangle-based methods are equivalent to estimating an integral by the trapezoid rule, in which the error is $O(h^2)$, so they are more accurate and they improve a lot better than cube-based methods when h is decreased. The midpoint triangles enable the use of lookup tables for the volume calculation, which is very fast (though less accurate than with MC triangles).

Noise reduction is trivially achieved by eliminating all objects whose volume falls below a user-specified threshold. A less obvious observation about noise is that it can also appear as small cavities in a large object, which can be eliminated by the same method. This will have a twofold effect: one is that an object's volume estimation will be more accurate. A second result is that such noise will not be seen if a cut-plane passes through it and the user wishes to visualize it using display method described in the next section. Noise elimination should not be an automatic feature of the application, because in clinical practice, only the expert radiologist should decide what constitutes noise and what constitutes small but possibly significant objects or cavities.

4. The "virtually-solid display method

One of the disadvantages of surface extraction lies in the interactive display of the objects. As long as the user wishes to see only entire objects, this is not a problem. However, in many applications, it is desirable to have the ability to display cutaway views of the objects by arbitrary cut-planes. Such views enable the user to visualize the internal structure of the objects. The outer surfaces are not a problem, and neither are the inner surfaces (surrounding an internal hole); both of them have normals pointing away from the object. The problem is how to display the surface of intersection between an object and the cut-plane, because our data at this point consists only of surfaces. A poor solution is simply to display the back polygons in some manner, but such a display looks unnatural and could be confusing to a practitioner. The most natural solution is to create new surface patches corresponding to the surfaces of intersection. Correct shading of such a surface patch will enable the accurate visualization of such cutaway views. However, such a solution is computationally time-consuming, so it is undesirable for real-time interaction.

The VS system includes a display method that creates the required views without adding new surface patches, and the cutaway views appear natural. This gives the impression that the objects are actually solid, so we call this method the "virtually-solid" display. Tarini et al. (2006) employ such a technique for molecular visualization, but their method does not account for interior cavities and interior objects.

Assume that the "−" side of a given cut-plane is to be discarded and only the "+" side is to be displayed. For every component, we maintain the minimum and maximum x, y and z values of its vertices. This data can be efficiently gathered during the voxel-sweep: when two components unite, the minimum x of the union is is simply the minimum of the two minimal x's of the components, and similarly for y and z and for the maximum. This data gives us a bounding box for every component. The first step of the display method is to discard entire components whose bounding box lies entirely in the "−" side of the cut-plane. Furthermore, any component whose bounding box is entirely in the "+" side can be displayed without any additional testing.

Of the remaining components, all triangles are tested against the cut-plane's equations. If all the vertices are on the "−" side, the triangle is discarded, and if they are all on the "+", it is displayed. If a triangle's vertices are on both sides of the cut-plane, then the triangle is divided into two parts, and only the part in the "+" side is displayed, as shown in Fig. 6. The displayed part of the triangle may be a smaller triangle as in the left part of Fig. 6, or it may be a quadrilateral. The quadrilateral may either be split into triangles as shown in the figure, or it can be displayed as it is. From a practical point-of-view, these operations are easily handled by OpenGL.

Consider now only the portions of the surfaces that lie in the "+" side of the cut-plane. As with any rendering, our method always displays the closest surface to the screen, but the cut-plane is excluded from this. All surface normals (of the displayed triangles) are either pointing towards the screen or away from the screen. Any portion of a surface (internal or external) whose normal points towards the screen is displayed in the usual manner.

Observe now that wherever the cut-plane intersects an object, and their intersection needs to be displayed, then the closest surface behind it necessarily has a *backward-pointing* normal. This is illustrated in Fig. 7: the surface patches (closest to the screen) with backward-pointing normals are enhanced. So, in order to render the cut surface correctly, all we need to do is to render the pixels corresponding to such surfaces as if they were displayed from the cut-plane. In Fig. 7, these portions of the screen are marked as "cut A" and "cut B". Such pixels are rendered with the same color as the rest of the object, but shaded according to some simple scheme suitable for a flat surface, such as constant (or flat) shading. Note that the rightmost "cut A" is due to two different back-facing surface patches of object A.

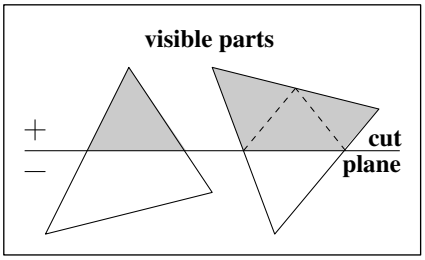

Fig. 6. A cut-plane intersecting surface triangles. The shaded portions are displayed.

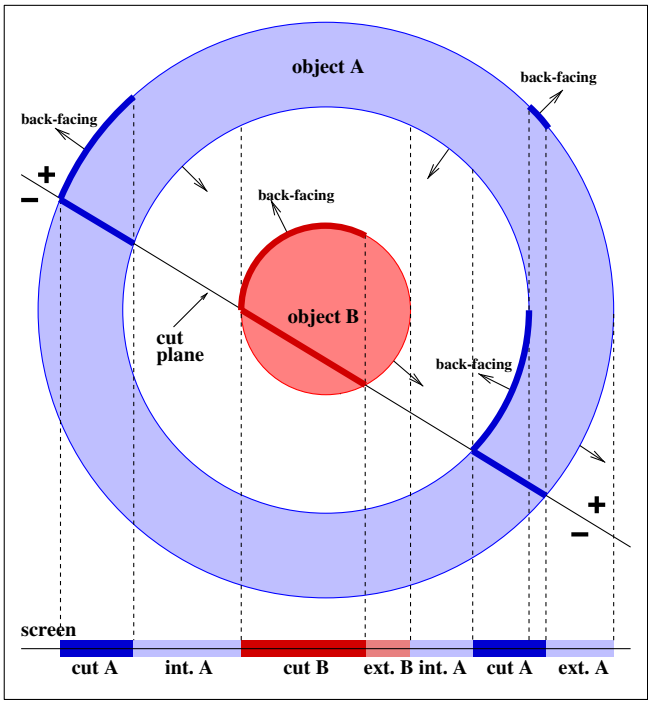

Fig. 7. The virtually-solid display: A cut-plane intersects objects *A* and *B*, and *B* is internal to *A*. The "−" side of the plane is discarded, and surface elements closest to the screen are displayed. No new surfaces are formed by the plane. All the forward-facing surfaces are displayed in the usual manner. If the normal of a surface is *back-facing* (see enhanced curved patches), then the corresponding pixel is displayed as if it was projected from the cut-plane.

One of the important topics in medical visualization is that of rendering speed. Real-time frame rates are the ideal goal for some applications, e.g., for visualizing a beating heart. Grevera et al. (2000), Grevera et al. (2005) report on extremely fast frame rates achieved with *shell rendering* and *T-shell rendering*, as compared to MC triangles rendered with OpenGL on a GPU (graphics processing unit). These methods are based on the digital model, but the *T-shell rendering* uses triangles with vertices at the midpoints of edges. Two technical points regarding rendering speeds should be borne in mind when using OpenGL on a GPU: the first one is the use display lists, as in (Grevera et al., 2000; Grevera et al., 2005). These take some initial time to set up, but rendering from different viewpoints is much faster. However, display lists are limited by the memory of the GPU. Another point is that triangle meshes can be rendered much faster by utilizing the *triangle strip* option of OpenGL, in which, after the initial triangle, just one vertex is needed for every additional triangle.

5. Results

The voxel-sweep and the marching cubes were implemented in C++ in the .NET environment. The program was run on an Intel Pentium 4 with 1 GB of memory and clock speed of 2.67 GHz, with an Nvidia G4-Ti4600 graphics card. The results were displayed with OpenGL, using the hardware *Z*-buffer, with Gouraud shading.

Fig. 8 shows the results of the VS algorithm on two objects: a CT scan of a human head (CThead), and a chain, created by three non-intersecting linked tori. The separate components are shown in different colors – we can see that in the skull, the lower jaw and the upper jaw are all separated. In the chain, each torus was identified as a separate component and assigned a different color. Other visual attributes, such as different levels of opacity, can also be assigned to different components

Fig. 9 describe four steps of VS on the "Tree" example, which consists of a hierarchical partitioning of a block into sub-blocks. Initially, all the separate components are shown in different colors since they are in different sets. As the sweep proceeds upwards, some sets combine and their colors also combine. Fig. 10 shows the "Boston teapot" during an intermediate step of the voxel-sweep.

Table 1 shows the run-times of MC and VS (in seconds) on the above datasets. Also shown in the table are some of the parameters of the datasets. It can be seen that both VS and the accelerated VS require, on average, only about 2% more time than MC and the accelerated MC, respectively.

The times in Table 1 do not include the preprocessing time for setting up the MinMax tree. These times are extremely fast since only arrays are involved and the tree is constructed in linear-time. All preprocessing times were less than one second. We also performed statistical experiments on random voxel arrays of sizes 256^3 and 512^3. Voxel values were taken as "short" integers and the testing program was compiled on the same machine under Linux, with the gcc compiler and the simplest "−O" optimization

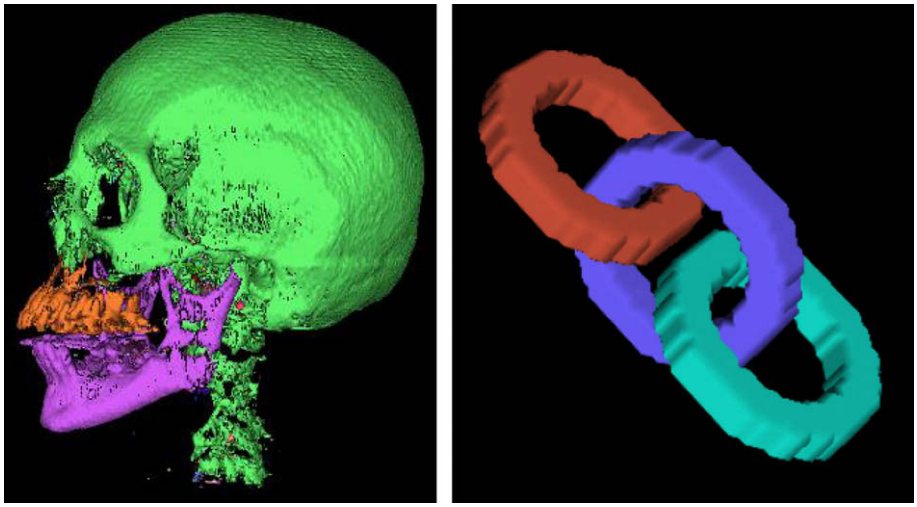

Fig. 8. CThead (left) and chain (right), showing connected components.

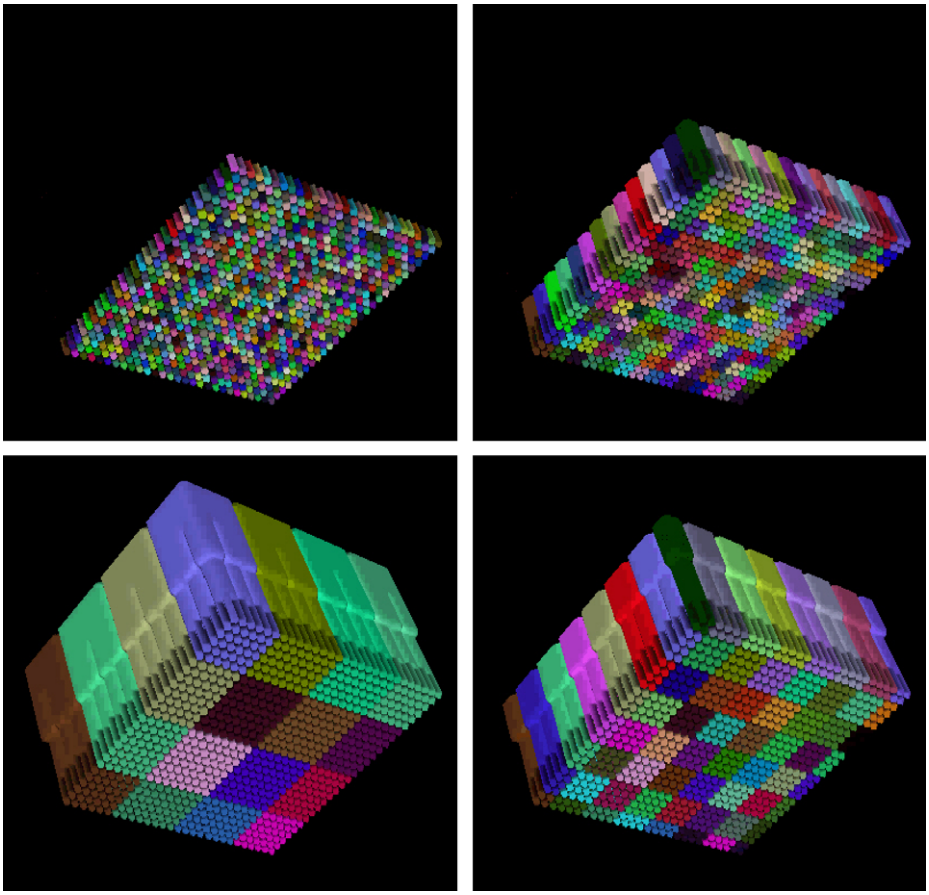

Fig. 9. Clockwise from top left: four steps of the tree example, showing union of components.

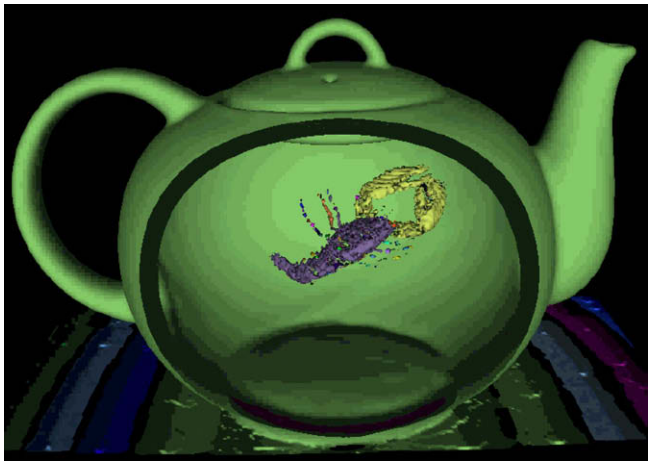

Fig. 10. Boston teapot with a lobster inside.

Table 1
Details of the four objects and the runtimes (s) of the marching cubes (MC), voxel-sweep (VS), and their accelerated versions (AMC and AVS).

Dataset	Dimensions	Triangles	MC	VS	AMC	AVS
CThead	$256 \times 256 \times 113$	464,414	4.49	4.58	2.9	2.94
Chain	$64 \times 64 \times 64$	10,560	0.109	0.11	0.06	0.062
Tree	$64 \times 64 \times 64$	151,824	0.836	0.88	0.8	0.81
Teapot	$178 \times 256 \times 256$	532,600	8.34	8.36	4.72	4.80

option. The average runtimes for the two array sizes were 0.237 and 1.892 s. These times conform to the theoretical linear running time of the preprocessing.

Fig. 11 shows the result of noise reduction of two different objects. Fig. 12 shows an image created by the virtually-solid display method. The left image shows the interior of a skull with part of it cut away by the cut-plane. The interior surface of the skull is shaded normally. The image on the right shows a sphere within a cavity in a surrounding sphere.

Fig. 13 shows a screen shot of the interactive user-interface built for the VS system. The image on the bottom right shows the image of a single slice, and the buttons below it move the image from one slice to another. The left half shows a 3D view of the object(s) after the voxel-sweep; the objects can be manipulated interactively

The graph on the right is a histogram of the voxel values, and above it are two sliders which determine the upper and lower bound of the region of interest. The sliders determine the positions of two markers on the histogram, so the user can choose these values according to certain particular features in the histogram, such as dips.

Note that while we discussed throughout the creation of an isosurface from a single isovalue, this can be extended trivially to account for two isovalues such that the region of interest lies between them. Assume all values are normalized to be between 0 and 1, and consider two values a, b such that $0 \leqslant a < b \leqslant 1$ and the user wishes to display all objects whose values lie between A and B. Denote $c = (a + b)/2$ and consider the mapping

$$f(x) = \begin{cases} x & \text{if } x < c \\ 2c - x & \text{otherwise} \end{cases}$$

Let $m = \min(0, 2c - 1)$. f maps the original range $[0, 1]$ to the interval $[m, c]$, so the region of interest is mapped to $[a, c]$. Hence, in the new interval $[m, c]$, we can simply use the single isovalue A. Two points should be noted: the first is that m can be negative, but we always have $m > -1$. Previously, we assumed that all voxel values at the boundary are zero, so if we want to display interval values, we should consider the exterior voxel values as -1. A second point is that the MinMax tree can be used for interval isosurfaces: instead of testing whether a single isovalue is within a given minmax range of a tree node, we should test whether the interval $[a, b]$ intersects the minmax range.

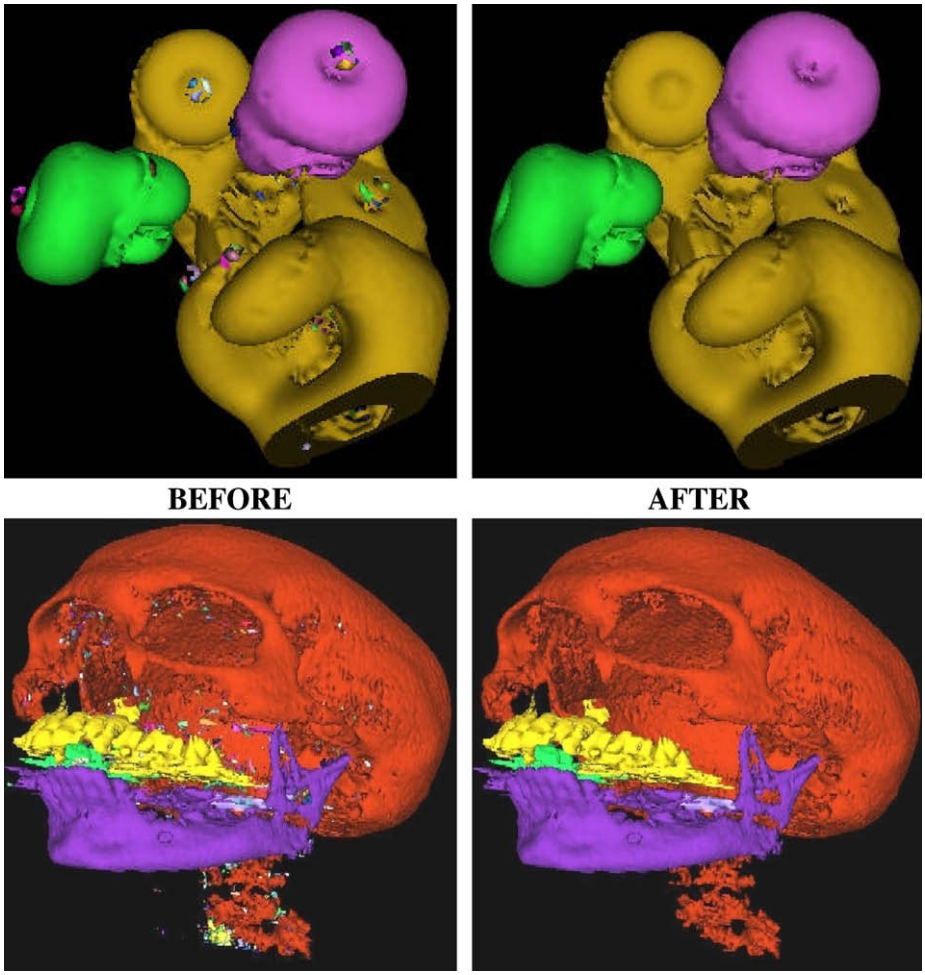

Fig. 11. Noise reduction by elimination of small components.

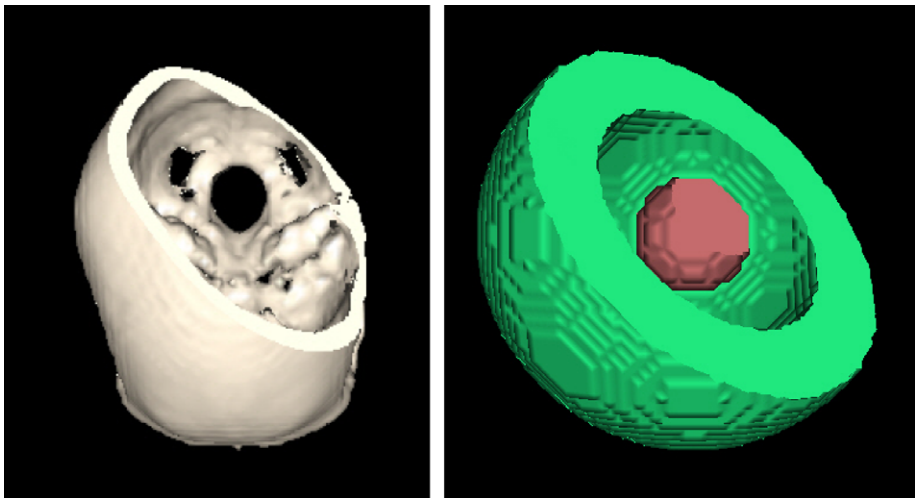

Fig. 12. Demonstration of the virtually-solid display method.

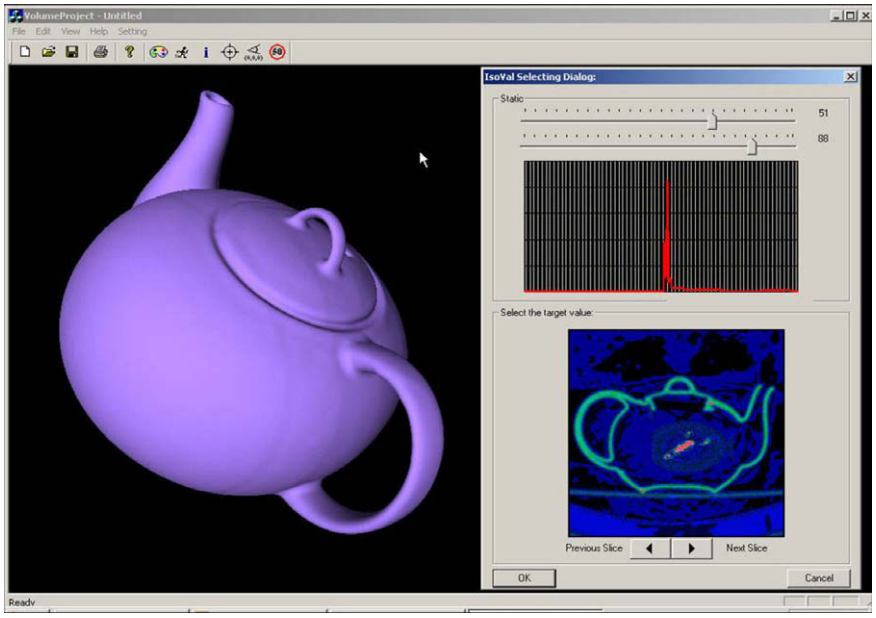

Fig. 13. The user interface of the VS system.

6. Conclusions

We have introduced the VS surface-based system, which enables several operations on voxel datasets. Using a very fast linear-time preprocessing step, it uses one sweep through the data to produce all isosurfaces determined by a given isovalue, separates them into connected components, and associates internal surfaces (cavities) and objects with their outer surface. The preprocessing can be used for any isovalue or interval of isovalues. The runtime is only about 2% more than the runtime of the speeded-up marching cubes algorithm. VS can be easily extended to handle several distinct isovalues by multithreading.

The association of an outer surface with internal cavities enables an accurate estimate of its enclosed volume by subtracting the volume of internal cavities from the volume enclosed by the outer surface. Noise can be eliminated by discarding objects and cavities with a small volume. A "virtually-solid" display method allows the user to introduce arbitrary cut-planes and to display nested surfaces and objects as if the objects were solid, without adding new surface patches to close up the intersection of the cut-plane and the objects.

Future research will concentrate on adding more features to the VS system. Of these, the most important is to allow a user to manipulate the isovalues within a small range and to obtain the results in real time, i.e., without running the voxel-sweep stage again. This concept is known as "exploring" the isosurfaces; (see Speray and Kennon, 1990; von Rymon-Lipinski et al., 2004; Zhang and, 2006). Improvements in rendering speeds are, as always, worthwhile future pursuits. Another topic for future research is the clinical evaluation of the VS system on many different medical datasets obtained with different modalities.

Acknowledgements

This research is based on the first author's MSc thesis in Computer Science, carried out under the second author's supervision at the University of Haifa. The research was supported by Grant No. 01-01-01509 from the Israel Ministry of Science and Technology. The CThead data was taken on a General Electric CT Scanner and provided courtesy of North Carolina Memorial Hospital. The Boston teapot data is due to Terarecon Inc., MERL, Brigham and Women's Hospital. The authors are indebted to the anonymous reviewers whose valuable comments led us to a much improved presentation of our results in the context of previous work.

References

Aho, A.V., Hopcroft, J.E., Ullman, J.D., 1983. Data Structures and Algorithms. Addison-Wesley, Reading, Mass.

Artzy, E., Frieder, G., Herman, G.T., 1981. The theory, design, implementation, and evaluation of a three-dimensional surface detection algorithm. Computer Graphics & Image Processing 15, 1–24.

Bajaj, C.L., Pascucci, V., Schikore, D., 1996. Fast isocontouring for improved visibility. In: VVS'96: Proceeding of the 1996 Symposium on Volume Visualization, San Francisco, October 1996.

Bajaj, C.L., Pascucci, V., Schikore, D.R., 1997. The contour spectrum. In: Proceeding of Visualization 1997. pp. 167–173

Baker, H.H., 1989. Building surfaces of evolution: the Weaving Wall. International Journal of Computer Vision 3, 51–71.

Bartz, D., Meissner, M., 1999. Voxels versus polygons: a comparative approach for volume graphics. In: Volume Graphics'99, International Workshop Processing. Swansea, U.K. 1999.

Bentley, J.L., 1975. Multidimensional binary search trees used for associative searching. Communications of the ACM 18 (9), 509–517 <http://doi.acm.org/10.1145/361002.361007>. ISSN: 0001-0782.

Cai, W., Sakas, G., 1998. Maximum intensity projection using splatting in sheared object space. Computer Graphics Forum 17 (3), 113–124.

Carneiro, B.P., Silva, C., Kaufman, A.E., 1996. Tetra-cubes: an algorithm to generate 3D isosurfaces based upon tetrahedra. Anais do IX SIBGRAPI. pp. 205–210

Carr, H., Snoeyink, J., 2003. Path seeds and flexible isosurfaces: using topology for exploratory visualization. In: Proceeding of Eurographics Visualization Symposium 2003. pp. 49–58.

Carr, H., Snoeyink, J., van de Panne, M., 2004. Simplifying flexible isosurfaces with local geometric measures. In: Proceeding of Visualization 2004. pp. 497–504.

Chernyaev, E., 1995. Marching cubes 33: Construction of topologically correct isosurfaces. Technical Report CERN CN 95-17, Institute for High Energy Physics, URL: <http://citeseer.ist.psu.edu/chernyaev95marching.html>.

Cignoni, P., Marino, P., Montani, C., Puppo, E., Scopigno, R., 1997. Speeding up isosurface extraction using interval trees. IEEE Transactions on Visualization & Computer Graphics 3 (2), 158–170.

Cohen, I., Gordon, D., 2004. The voxel-sweep: a boundary-based algorithm for object segmentation and connected-components detection. In: Girod, B., Magnor, M., Seidel, H.-P. (Eds.), Proceeding of 9th International Workshop on Vision, Modeling & Visualization (VMV 2004), Stanford, CA, USA, November 16–18. Akademische Verlagsgesellschaft Aka GmbH, Berlin. pp. 405–411 & 471.

Cohen-Or, D., Fleishman, S., 1995. An incremental alignment algorithm for parallel volume rendering. Computer Graphics Forum 14 (3), 123–133.

Cohen-Or, D., Sheffer, Z., 1994. Proximity clouds – an acceleration technique for 3D grid traversal. The Visual Computer 11 (1), 27–38.

Criscione, P., Montani, C., Scateni, R., Scopigno, R., 1996. DiscMC: an interactive system for fast fitting isosurfaces on volume data. In: Proceeding of Eurographics Workshop on Virtual Environments and Scientific Visualization'96. Springer-Verlag, London, UK, pp. 178–190. ISBN: 3-211-82886-9.

Edelsbrunner, H., Harer, J., Mascarenhas, A., Pascucci, V., Snoeyink, J., 2008. Time-varying Reeb graphs for continuous space-time data. Computational Geometry: Theory & Applications 41, 149–166.

Frieder, G., Gordon, D., Reynolds, R.A., 1985. Back-to-front display of voxel-based objects. IEEE Computer Graphics & Applications 5 (1), 52–60.

Gallagher, R.S., 1991. Span filtering: an optimization scheme for volume visualization of large finite element models. In: VIS'91: Proceeding of the 2nd Conference Visualization'91, Los Alamitos, CA, USA. IEEE Computer Society Press, pp. 68–75. ISBN: 0-8186-2245-8, PAPER.

Giles, M., Haimes, R., 1990. Advanced interactive visualization for CFD. Computing Systems in Education 1 (1), 51–62.

Gordon, D., Udupa, J.K., 1989. Fast surface tracking in three-dimensional binary images. Computer Vision, Graphics, & Image Processing 45, 196–214.

Grevera, G.J., Udupa, J.K., Odhner, D., 2000. An order of magnitude faster isosurface rendering in software on a PC than using dedicated, general purpose rendering hardware. IEEE Transactions on Visualization & Computer Graphics 6 (4), 335–345.

Grevera, G.J., Udupa, J.K., Odhner, D., 2005. T-shell rendering and manipulation. In: Galloway, R.L., Cleary, K.R., Jr. (Eds.), Proceeding of the SPIE Conference on Medical Imaging: Visualization, Image-Guided Procedures, and Display, vol. 5744. SPIE, Belingham, WA, USA, pp. 22–33.

Herman, G.T., Webster, D., 1983. A topological proof of a surface tracking algorithm. Computer Vision, Graphics, & Image Processing 23, 162–177.

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

Karni, Z., Gotsman, C., 2000. Spectral compression of 3D meshes. In : Proceeding of the ACM SIGGRAPH Conference. pp. 279–286.

Kobbelt, L.P., Botsch, M., Schwaneckle, U., Seidel, H.-P., 2001. Feature-sensitive surface extraction from volume data. In: Proceeding of the ACM SIGGRAPH Conference. pp. 57–66.

Kong, T.Y., Udupa, J.K., 1992. A justification of a fast surface tracking algorithm. CVGIP: Graphical Models & Image Processing 54 (2), 162–170.

Lachaud, J.-O., Montanvert, A., 2000. Continuous analogs of digital boundaries: a topological approach to iso-surfaces. Graphical Models 62, 129–164.

Lacroute, P., Levoy, M., 1994. Fast volume rendering using a shear-warp factorization of the viewing transformation. In: Proceeding of ACM SIGGRAPH Conference. pp. 451–458.

Lancaster, J.L., Eberly, D., Alyassin, A., Downs III, J.H., Fox, P.T., 1992. A geometric model for measurement of surface distance, surface area and volume from tomographic images. Medical Physics 19 (2), 419–431.

Levoy, M., 1988. Display of surfaces from volume data. IEEE Computer Graphics & Applications 8 (5), 29–37.

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

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

Lorensen, W.E., Cline, H.E., 1987. Marching cubes: a high resolution 3D surface construction algorithm. Computer Graphics 21 (4), 163–169. ACM SIGGRAPH Annual Conf. Proc.

Meagher, D.J., 1984. Interactive solids processing for medical analysis and planning. In: Proceeding of the National Computer Graphics Association. pp. 96–106.

Nielson, G.M., 2003. On Marching Cubes. IEEE Transactions on Visualization & Computer Graphics 9 (3), 283–297.

Nielson, G.M., Hamann, B., 1991. The asymptotic decider: resolving the ambiguity in marching cubes. In: Proceeding of the IEEE Visualization'91 Conference. pp. 83–91.

Nyström, I., Udupa, J.K., Grevera, G.J., Hirsch, B.E., 2002. Area of and volume enclosed by digital and triangulated surfaces. In: Mun, S.K. (Ed.), Proceeding of the SPIE Conference on Medical Imaging: Visualization, Image-Guided Procedures, and Display, vol. 4681. SPIE, Belingham, WA, USA, pp. 669–680.

Pascucci, V., Cole-McLaughlin, K., 2003. Efficient computation of the topology of level sets. Algorithmica 38 (2), 249–268.

Reeb, G., 1946. Sur les points singuliers d'une forme de pfaff complètement intégrable ou dune fonction numérique. Comptes Rendus de l'Acadèmie des Sciences de Paris 222, 847–849.

Reynolds, R.A., Gordon, D., Chen, L.-S., 1987. A dynamic screen technique for shaded graphics display of slice-represented objects. Computer Vision, Graphics, & Image Processing 38 (3), 275–298.

Schiemann, T., Nuthmann, J., Tiede, U., Höhne, K.H., 1996. Segmentation of the visible human for high quality volume based visualization. In: Höhne, K., Kikinis, R. (Eds.), Visualization in Biomedical Computing: 4th Intern'l Conf. Proc. VBC '96, Hamburg, Germany, September 22–25, Lecture Notes in Computer Science. Springer-Verlag, Berlin, pp. 13–22.

Shen, H.-W., Johnson, C.R., 1995. Sweeping simplices: a fast iso-surface extraction algorithm for unstructured grids. In: IEEE Visualization'95, Annual Conference. IEEE Computer Science Press, pp. 143–150.

Shen, H.-W., Hansen, C.D., Livnat, Y., Johnson, C.R., 1996. Isosurfacing in span space with utmost efficiency (ISSUE). In: Yagel, R., Nielson, G.M. (Eds.), IEEE Visualization'96, Annual Conference. IEEE Computer Science Press, pp. 287–294.

Shirley, P., Tuchman, A., 1990. A polygonal approximation to direct scalar volume rendering. In: VVS'90: Proceeding of the 1990 Workshop on Volume Visualization. ACM Press, New York, NY, USA, pp. 63–70. ISBN: 0-89791-417-1.

Sobierajski, L., Cohen, D., Kaufman, A., Yagel, R., Acker, D., 1993. A fast display method for volumetric data. The Visual Computer 10 (2), 116–124.

Speray, P., Kennon, S., 1990. Volume probes: interactive data exploration on arbitrary grids. In: VVS'90: Proceeding of the 1990 Workshop on Volume Visualization. ACM Press, New York, NY, USA, pp. 5–12 <http://doi.acm.org/10.1145/99307.99310>. ISBN: 0-89791-417-1.

Sutton, P., Hansen, C., Shen, H.-W., Schikore, D., 2000. A case study of isosurface extraction algorithm performance. In: VisSym'00: Joint Eurographics–IEEE TCVG Symposium on Visualization.

Takahashi, S., Takeshima, Y., Fujishiro, I., 2004. Topological volume skeletonization and its application to transfer function design. Graphical Models 66, 24–49.

Takahashi, S., Fujishiro, I., Takeshima, Y., 2005. Interval volume decomposer: a topological approach to volume traversal. In: Erbacher, R.F., Börner, K., Gröhn, M., Roberts, J.C., (Eds.), Visualization and Data Analysis 2005 (Proceeding of the SPIE). URL: <http://nis-lab.is.s.u-tokyo.ac.jp/shigeo/pdf/vda2005.pdf>.

Takahashi, S., Takeshima, Y., Fujishiro, I., Nielson, G.M., 2006. Emphasizing isosurface embeddings in direct volume rendering. In: Bonneau, G.-P., Ertl, T., Nielson, G.M. (Eds.), Scientific Visualization: The Visual Extraction of Knowledge from Data. Springer-Verlag, Berlin, pp. 185–206.

Tarini, M., Cignoni, P., Montani, C., 2006. Ambient occlusion and edge cueing to enhance real time molecular visualization. IEEE Transactions on Visualization & Computer Graphics 12 (5), 1237–1244.

Touma, C., Gotsman, C., 1998. Triangle mesh compression. In: Proceeding of the Graphics Interface Conference. 1998. pp. 26–34.

Udupa, J.K., 1981. Determination of 3D shape parameters from boundary information. Computer Graphics & Image Processing 17, 52–59.

Udupa, J.K., Ajjanagadde, V.G., 1990. Boundary and object labelling in three-dimensional images. Computer Vision, Graphics & Image Processing 51, 355–369.

von Rymon-Lipinski, B., Hanssen, N., Jansen, T., Ritter, L., Keeve, E., 2004. Hardware-accelerated point-based rendering and exploration of high-resolution isosurfaces. In: Proceeding of Visualization 2004. pp. 441–448.

Weber, G., Dillard, S., Carr, H., Pascucci, V., Hamann, B., 2007. Topology-controlled volume rendering. IEEE Transactions on Visualization & Computer Graphics 13 (2), 330–341. March/April 2007.

Wilhelms, J., van Gelder, A., 1992. Octrees for faster isosurface generation. ACM Transactions on Graphics 11 (3), 201–227.

Wyvill, G., McPheeters, C., Wyvill, B., 1986. Data structure for soft objects. Visual Computer 2, 227–234.

Ylä-Jääski, J., Klein, F., Kübler, O., 1991. Fast direct display of volume data for medical diagnosis. CVGIP: Graphical Models & Image Processing 53 (1), 7–18. January 1991.

Zhang, H., Kaufman, A., 2006. Interactive point-based isosurface exploration and high-quality rendering. IEEE Transactions on Visualization & Computer Graphics 12 (5), 1267–1274.