ABSTRACT

A method is proposed which supports the extraction of isosurfaces from irregular volume data, represented by tetrahedral decompositions, in optimal time. The method is based on a data structure called interval tree, which encodes a set of intervals on the real line, and supports efficient retrieval of all intervals containing a given value. The implementation of the method is simple. Tests have shown that its practical performance reflects the theoretical optimality.


1 INTRODUCTION

Isosurface extraction and visualization represents one of the most effective and powerful techniques for the investigation of volumetric scalar datasets. In the same way as contour lines promote the understanding of a digital terrain model, isosurfaces improve the comprehension of the phenomenon, process or structure that the scalar field represents.

The problem of the extraction of isosurfaces from a volumetric dataset can be formally stated as follows. A scalar volume dataset is a pair \((V, W)\), where \(V = \{v_i \in \mathbb{R}^3, i = 1, \ldots, n\}\) is a finite set of points in a domain \(\Omega \subset \mathbb{R}^3\); \(W = \{w_i \in \mathbb{R}, i = 1, \ldots, n\}\) is the corresponding set of scalar values, obtained by measuring a tri-variate scalar field \(f(x, y, z)\) in the sampling points in \(V\), i.e. \(w_i = f(v_i)\).

A digital interpolating model for volume datasets \((V, W)\) is a pair \((\Sigma, \Phi)\), where \(\Sigma\) is a subdivision of \(\Omega\) into cells \(\sigma_1, \ldots, \sigma_m\), and \(\Phi\) is a corresponding family of real tri-variate functions \(\phi_j : \sigma_j \rightarrow \mathbb{R}, j = 1, \ldots, m\), which interpolate the values of \(W\) at all points of \(V\). If functions of \(\Phi\) are coincident at the common boundary of adjacent cells of \(\Sigma\), then a continuous function \(\phi\) is defined piecewise by \(\Phi\) on \(\Sigma\):

\[
\phi(p) = \phi_j(p) \text{ if } p \in \sigma_j, \forall j = 1, \ldots, m
\]

Function \(\phi\) is in practice an estimate of the (otherwise unknown) measured function \(f\) over \(\Omega\).

Given \(q \in \mathbb{R}\), the set \(S(q) = \{p \in \Omega \mid \phi(p) = q\}\) is called an isosurface of \(\phi\) at value \(q\). If \(\phi\) is continuous, and \(q\) is not an extremal value of \(\phi\), then \(S(q)\) is a 2-manifold embedded in \(\mathbb{R}^3\), possibly with a one-dimensional boundary contained in the boundary of \(\Omega\). \(S(q)\) is defined piecewise on the cells of \(\Sigma\): each cell \(\sigma_j \in \Sigma\) such that \(\min_{\sigma_j} \phi_j \leq q \leq \max_{\sigma_j} \phi_j\) is called active at \(q\), and it contributes to \(S(q)\) for a patch corresponding to the locus of points

\[
S_j(q) = \{p \in \sigma_j \mid \phi_j(p) = q\}
\]

The isosurface extraction problem consists in finding all patches \(S_j(q)\) that correspond to active cells, given \((\Sigma, \Phi)\) and \(q\). Traditional methods for isosurface extraction - such as the Marching Cubes (MC) [9], and its straightforward generalization to tetrahedral models, namely the Marching Tetrahedra (MT) - analyze in turn every cell of \(\Sigma\), and for each active cell compute the corresponding isosurface patch. Hence, the computational complexity of such techniques is linear in the size of \(\Sigma\).

Some speedup techniques, including the one proposed here, are aimed at avoiding analyzing non-active cells, hence yielding an output-sensitive computational cost.

In the following sections, we first survey related works and then we describe our proposal for a time-optimal isosurface extraction technique based on the use of the interval tree data structure [3]. In spite of the generality of the proposed approach (the interval-based strategy makes the solution suitable for regular or curvilinear datasets as well), the focus of the paper is on irregular volume data. This is because in the case of hexahedral decompositions the memory overheads due to interval tree storing are excessive if compared with those of alternative data structures (e.g. octrees or pyramids). Finally, we conclude by giving details on the implementation of the algorithm, an analysis of its computational complexity, and some quantitative results.

2 RELATED WORK

Given an isovalue \(q\), the aim of the speedup techniques is to avoid the analysis of non-active cells, with respect to \(q\), and to exploit coherence in the extraction phase.

Speedup techniques can be classified according to four main non-orthogonal criteria:

---

1 Via S. Maria, 46 - 56126 Pisa ITALY
Email: {cignoni, montani}@iei.pi.cnr.it
2 Via dei Marini, 6 (Torre di Francia) - 16149 Genova, ITALY
Email: puppo@ima.ge.cnr.it
3 Via S. Maria 36, 56126 Pisa, ITALY
Email: r.scopigno@cnuce.cnr.it


- **working domain**: this defines the classes of datasets which can be managed by each speedup technique; the approaches proposed often lack generality because they are designed for a single dataset class, e.g. regular datasets;

- **search modality**: active cells can be searched for by moving either in the geometric space (*geometric approach*) or, alternatively, in the interval space, defined as the set of the $\min - \max$ data value interval of each cell, searching for active intervals and then for the corresponding active cells (*interval approach*). The selection of the search modality is often subject to the geometric structure of the underlying dataset: the geometric approach is generally well suited for regular datasets, which permit the use of regular, hierarchical spatial decomposition structures, which are particularly suitable for the speedup of the traversal and classification phase; the interval approach is independent of the geometric structure of the dataset, though it generally implies higher costs in terms of memory requirements;

- **local coherence (or coherence between cells)**: given the isosurface patch extracted from a cell, information can be exploited from the local computations, both to detect the neighboring cells which will be intersected by the same isosurface, and to reduce redundancy in geometric computations (vertices are shared by adjacent cells). The process is iterative and it primes a sort of propagation over the cells. Most existing techniques do not exploit local coherence because: (a) managing a propagation process entails storing the state of the computation, and frequent access to a data structure which stores this status, with significative overheads; (b) if the isosurface bounds a surface with multiple components then the propagation does not warrant all the active cells being visited, hence implying that cells should be tested once in any case, with a corresponding reduction in efficiency;

- **global coherence (or coherence between isosurfaces)**: some techniques can exploit the coherence between different isosurfaces. Part of the information derived from the extraction of the isosurface at value $q$ (i.e. active intervals and/or cells) can be efficiently reused in the extraction of an isosurface at value $q'$, with $q'$ close to $q$. This feature is particularly important in applications requiring the simultaneous visualization of multiple isosurfaces or which change the isosurface $q$ continuously and smoothly.

Wilhelms and Van Gelder [15] propose the use of a branch-on-need octree, a hierarchical spatial decomposition data structure, to purge subvolumes while fitting isosurfaces. Each node of the tree contains the $\min - \max$ interval of the underlying sub-volume. The nodes at the lowest level represent groups of $2 \times 2 \times 2$ cells (this is not mandatory, but reduces the global size of the octree structure). The hierarchical nature of the octree enables the user to restrict the search to only the $2 \times 2 \times 2$ parts of the original dataset containing at least one active cell. The method adopts a geometric approach. The domain is limited to the case of regular datasets or to curvilinear datasets defined on a warped regular grid. It does not exploit global coherence. Local computations are saved and reused through a hash-based caching strategy. It has been demonstrated [8] that the worst case time complexity for the octree search phase is $O(k + \log(n/k))$, with $n$ the number of cells and $k$ the size of the output.

A similar geometric approach is used by Criscione et al. [2]. In real 3D datasets it is uncommon to have large uniform regions. This fact reduces the advantages in terms of required space of octree-based data structures. In [2] a simpler pyramid data structure is proposed. Each level of the pyramid is a 3D array of intervals. Each entry contains the $\min - \max$ data value interval of the (usually 8) corresponding entries of the lower level. Local coherence does not need to be taken into account because of the particular approximated fitting approach adopted. As in the previous method, the domain is limited to regular datasets. The empirical computational complexity is very close to that observed for the octree.

A completely different geometric approach, based on the use of an *extrema graph*, is proposed by Itoh and Koyama [6]. The nodes of the graph are the cells of the volume which hold local extrema data values (local minima or maxima). Each arc of the graph supports a list of the cells connecting its two end nodes. Given an isovalue, an active arc is searched for in the extrema graph. The cells connected to this arc are then sequentially scanned until an active one is found and, finally, a propagation method is activated on this cell. Local coherence is therefore exploited by this propagation process. At the end of the process, boundary cells need to be tested further when the method scans other arcs on the graph. The propagation algorithm adopted, following Sperary and Kennon [13], uses a FIFO queue to store the identifiers of the adjacent active cells to be visited, and marks those cells to avoid enqueueing them again. The algorithm can be applied both to regular and irregular datasets. Global coherence is not taken into account. The worst case complexity is estimated [8] to be $O(n)$.

The above techniques make use of data structures which relate to the underlying geometric structure of the dataset, whereas the following methods are based on an *interval space* approach. The search is carried out on the interval set and, for each active interval, on the corresponding active cell. All such techniques apply both to regular and irregular datasets, but the waste of memory due to the loss of implicit spatial information makes them more suitable for irregular datasets.

Giles and Haimes [5] report an approach in which two sorted interval lists are constructed in a pre-processing phase by sorting the cells’ minimum and maximum values. The global maximum range of each cell is also computed. Given an isovalue, an *active list* containing all the active intervals is created by referring to the two sorted lists. If the specified isovalue is changed by less than the global maximum range with respect to the previous one, then the active list is augmented with the intervals lying between the two isovalue. Only one of the original lists is used in this process, depending on the sign of the isovalue’s change. The active list is then purged of all the cells that do not intersect the new isosurface. Only the cells corresponding to the intervals of the final active list are visited to extract the isosurface. If the isovalue change smoothly, the method exploits global coherence.

Gallagher [4] proposes a *span filter* algorithm. First, the range of the scalar values is subdivided into buckets. The number of buckets that a $\min - \max$ interval crosses is called the *span length* of the corresponding cell. The intervals are then distributed into different span lists according to their span lengths. Within each span list, the intervals are subdivided into buckets; the subdivision is based on the $\min$ value of the interval. Given an isovalue, the span lists are scanned, the active buckets retrieved, and the corresponding cells visited. Due to the lack of ordering between the intervals of each bucket, the technique cannot exploit global coherence.

Shen and Johnson [12] try to overcome the limitations of [4] and [5] by defining the *sweeping simplices* algorithm in which both sorted lists of the active list algorithm, and the spatial decomposition into buckets of the intervals of the span filter algorithm are present. Their algorithm is able to exploit global coherence for small changes in the selected isovalue.

Local coherence is not taken into account explicitly in all three of the above methods. The previous algorithms present a time complexity of $O(n)$ for the search phase.
Livnat, Shen, and Johnson [8] present a near optimal extraction algorithm by first introducing a new representation space for the \( \min - \max \) intervals, the \textit{span space} (see Figure 1) and then using a kd-tree to locate points (i.e. the \textit{active intervals}) in this space. Their algorithm exploits global coherence and it is estimated to have a \( O(\sqrt{n} + k) \) worst case complexity for the search phase.

The span space is very useful to geometrically understand the algorithms based on the interval approach and we will refer to this representation in the next sections.

It is difficult to get useful results from the computational analysis of the methods outlined above. Very often, such an analysis does not predict the behaviour of the algorithms in real applications. Moreover, implementations of the solutions reviewed are not available in the public domain, at least to our knowledge, thus making it difficult to compare these different solutions empirically.

We should also mention other authors, who have used the interval tree (i.e., the same data structure on which we base our work) to address problems related to isosurface extraction.

In [7], Laszlo considers the extraction of wireframes approximating isosurfaces from a volume grid whose cells are generic polyhedra. Given a threshold value, the wireframe is produced by connecting pairs of points obtained by intersecting grid edges with the corresponding isosurface (linear interpolation is used along each edge). The method requires a complicated definition and pre-computation of \textit{supporting pairs}, i.e., pairs of grid edges that can generate edges of the wireframe. The interval tree is used to speedup the search for edges and supporting pairs that contribute to producing the wireframe. The implementation reported shows a considerable overhead of the data structure.

In [14], van Kreveld deals with the extraction of isolines from triangulated terrain data. Each triangle is associated with an interval in an interval tree, and this structure is used to speed up the extraction of triangles that contribute to form isolines.

### 3 INTERVAL TREES FOR VOLUME DATA

The technique proposed in this paper is based on an interval approach and it can thus be used both for regular and irregular datasets. However, hereafter we will consider volume datasets made of points irregularly distributed in space, and linear tetrahedral models:

- \( \Sigma \) is a tetrahedralization of \( V \), i.e., a subdivision of \( \Omega \) into tetrahedra having \( V \) as the set of vertices;
- for each \( j = 1 \ldots m \), function \( f_j \) is a linear function interpolating \( f \) at the four vertices of \( \sigma_j \).

In this case, an isosurface of \( \phi \) is a triangulated surface: each active tetrahedron \( \sigma_j \) of \( \Sigma \) contributes for either one or two triangular patches to the isosurface. The patch(es) corresponding to an active tetrahedron \( \sigma_j \) is [are] found by computing the intersection points of the isosurface with the edges of \( \sigma_j \) through linear interpolation along such edges: such points are joined through edges to form a triangle [a pair of coplanar and adjacent triangles].

Our technique is based on the \textit{interval tree}; a data structure originally proposed by Edelsbrunner [3] (see also [11]) to give an efficient solution to the following query problem:

given a set \( \mathcal{I} = \{ I_1, \ldots, I_m \} \) of intervals of the form \( [a_i, b_i] \), with \( a_i \leq b_i \) on the real line, and a query value \( q \), report all intervals of \( \mathcal{I} \) that contain \( q \).

The problem can be effectively visualized (see Figure 1) using the span space introduced by Livnat, Shen, and Johnson [8]: each interval \( I_i = [a_i, b_i] \) is represented as a point in a 2D cartesian space using the extremes \( a_i, b_i \) as the x and y coordinates of the point. From a geometrical point of view, the problem of reporting all intervals \( I \) that contain the query value \( q \) reduces itself to collecting the points in the span space lying in the intersection of the two halfspaces \( \min \leq q \) and \( \max \geq q \).

A general definition and use of the interval tree are briefly described below. For each \( i = 1, \ldots, m \), let us consider the sorted sequence of values \( X = (x_1, \ldots, x_h) \) corresponding to distinct extremes of intervals (i.e., each extreme \( a_i, b_i \) is equal to some \( x_j \)). Thus, generally \( h \leq 2m \), though \( h \) could be considerably smaller than \( 2m \) if the intervals of \( \mathcal{I} \) can only have extremes at a predefined (and relatively small) set of values. The interval tree for \( \mathcal{I} \) consists of a balanced binary search tree \( \mathcal{T} \) whose nodes correspond to values of \( X \), plus a structure of lists of intervals appended to non-leaf nodes of \( \mathcal{T} \). The interval tree can be defined recursively as follows:

- \( \mathcal{I}_L = \{ I_i \in \mathcal{I} \mid b_i < \delta \} \);
- \( \mathcal{I}_r = \{ I_i \in \mathcal{I} \mid a_i > \delta \} \);
- \( \mathcal{I}_b = \{ I_i \in \mathcal{I} \mid a_i \leq \delta \leq b_i \} \).

The intervals of \( \mathcal{I}_b \) are arranged into two sorted lists \( \mathcal{AL} \) and \( \mathcal{DR} \) as follows:

- \( \mathcal{AL} \) contains all elements of \( \mathcal{I}_b \) sorted in ascending order according to their left extremes \( a_i \);
- \( \mathcal{DR} \) contains all elements of \( \mathcal{I}_b \) sorted in descending order according to their right extremes \( b_i \).

The left and the right subtrees are defined recursively by considering interval sets \( \mathcal{I}_L \) and \( \mathcal{I}_r \), and extreme sets \( \{ x_1, \ldots, x_{h-1} \} \) and \( \{ x_{h+1}, \ldots, x_h \} \), respectively.

An example of a simple interval tree, built out of few cells, is shown in Figure 2.
From a geometric point of view, the interval tree partitionates the span space as shown in Figure 3, given the interval points distribution presented in Figure 1. The numbers indicate the different levels of the tree, starting with the root at level 1.

The interval tree can be constructed in \( O(m \log m) \) time by a direct implementation of its recursive definition. The tree is balanced, hence it has at most a height of \( \lceil \log_2 m \rceil \) and \( h \) nodes: each node contains two pointers to interval lists. Each interval appears in two lists, hence giving a total of \( 2m \) list elements, each of which contains a pointer to its corresponding cell.

Given a query value \( q \), tree \( T \) is visited recursively starting at its root:

- if \( q < \delta \), then list \( \mathcal{AL} \) is scanned until an interval \( I_i \) is found such that \( a_i > q \); all scanned intervals are reported; the left subtree is visited recursively;
- if \( q > \delta \), then list \( \mathcal{DR} \) is scanned until an interval \( I_i \) is found such that \( b_i < q \); all scanned intervals are reported; the right subtree is visited recursively;
- if \( q = \delta \), then the whole list \( \mathcal{AC} \) is reported.

The geometric interpretation of the searching algorithm based on the span space representation is given in Figure 4. Each sector of the space (node of the tree) which contains the horizontal query line \( q \) (i.e. \( \delta \geq q \)) is visited top-down (scanning the \( \mathcal{AL} \) list) until the query line \( \max = q \) is reached; the sectors containing the vertical line \( q \) are visited left to right (scanning the \( \mathcal{DR} \) list) until the query line \( \min = q \) is reached.

Therefore, \( \lceil \log_2 h \rceil \) nodes of the tree are visited, and for each node only the intervals reported in output, plus one, are visited. Hence, if \( k \) is the output size, then the computational complexity of the search is \( O(k + \log h) \). Since \( \log h \) is the minimum number of bits needed to discriminate between two different values in \( X \), no query technique could have a computational complexity smaller than \( O(\log h) \), hence the computational complexity of querying with the interval tree is output-sensitive optimal. It is interesting to note that the time complexity is independent of the total number \( m \) of intervals, in fact it only depends on the output size, and on the number of distinct extremes.

Given a (continuous) digital interpolating model \((\Sigma, \Phi)\), we associate with each cell \( \sigma_j \in \Sigma \) an interval \( I_j \), whose extremes \( a_j \) and \( b_j \) are the minimum and maximum values of \( \phi \) inside \( \sigma_j \), respectively. If \((\Sigma, \Phi)\) is a linear tetrahedral model, this minimum and maximum correspond to the smallest and largest values of \( W \) taken at the vertices of \( \sigma_j \), respectively. Given a query value \( q \), isosurface \( S(q) \) intersects \( \sigma_j \) if and only if \( q \) is contained in \( I_j \). Therefore, we build an interval tree for the corresponding set of intervals \( I \), where each element corresponding to an interval \( I_j \) in an interval list contains a direct pointer to the corresponding cell \( \sigma_j \). Hence, a query to the interval tree returns in \( O(k + \log h) \) all \( k \) active cells.

### 4 IMPLEMENTATION

This section outlines the rationale behind our implementation of the interval tree data structure. The main issues in our implementation are the memory occupation and the simplifications due to the use of static data structures.

Moreover, in our investigation (isosurface fitting) the number \( m \) of intervals is usually larger than the number \( h \) of interval endpoints. Indeed, each cell corresponds exactly to one interval, whose extremes are the minimum and maximum values of the points of that cell. Therefore, the total number \( h \) of interval endpoints is certainly not greater than the number \( n \) of points; in most cases \( m > n \geq h \) holds.

We maintain the tree encoded as an implicitly stored binary tree. For each node \( r \) of the tree we store the discriminant value \( \delta_r \), and two pointers to the sorted lists \( \mathcal{AL} \) and \( \mathcal{DR} \). Because these lists are never updated, we can store all of them packed in a single array.
maintaining for each node of the tree the index of the first element of the list, and its length.

For each element of lists $\mathcal{A}_c$ and $\mathcal{D}_R$ we store the reference to the cell the list represents, and the value of the left [right] endpoint. If memory occupation is a relevant issue, we could improve the space efficiency of the structure by avoiding storing the endpoint values in the list (they can easily be recalculated by searching for the min [max] value in the referred cell).

Space Complexity
We assume that a memory word stores either a real value, or an integer, or a pointer. Given a scalar volume dataset $(V, W)$ composed of $n$ points, $m$ tetrahedral cells and $h < n$ distinct interval endpoints, we build an interval tree of $m$ intervals. For each interval, we store the reference to its cell and one interval endpoint in both the $\mathcal{A}_c$ and $\mathcal{D}_R$ lists, using $4m$ memory words. The tree is stored as a vector of $h$ elements, each containing the discriminant value, two pointers to the lists and the lists length, using at most $th$ memory words. Therefore, the total memory occupation is at most $th + 4m$ words, which is less than $4n + 4m$.

The space required is much higher than that needed to represent a regular dataset ($n$ words), and slightly lower to that needed to store an irregular one, adopting a tetrahedral representation ($4n + 4m$ words, to store the list of vertex coordinates, the field value and for each cell, the four vertex indexes). But comparing the interval tree memory overhead with the rough volume dataset size may be misleading. In the case of curvilinear or unstructured dataset, much more information is needed to perform efficiently the visualization (to render multiple semi-opaque isosurfaces or to adopt a projective DVR approach). An example is the need to store explicitly the adjacency information between tetrahedral cells and the plane equations of cell faces, then we need for each cell four references to adjacent cells, and four references to the plane equations (each equation is shared between two tetrahedra so we have $2m$ equations, each containing 4 values). The total memory requirement therefore rises to $4n + 4m + 4m + 2m = 4n + 20m$.

In most tetrahedral complexes there are approximately six tetrahedra for each point\(^1\), and therefore we can assume $m = 6n$. By exploiting this fact in the previous estimations, we find that the memory overhead due to the use of an interval tree on tetrahedral meshes is bounded from above by: $40n/124n \approx 32\%$.

5 EXPERIMENTAL RESULTS

We implemented the interval tree on top of our prototypal volume visualization system\(^2\), $\text{TAn}$ (Tetrahedra Analizer). $\text{TAn}$ supports multiresolution modeling and visualization of volume datasets by adopting an innovative representation based on tetrahedral decomposition\([1]\). No limitations are imposed on the class of data manageable: $\text{TAn}$ processes either regular, curvilinear or scattered data. Using tetrahedral decompositions is also an effective choice for the design of integrated visualization methods. In particular, $\text{TAn}$ supports isosurface fitting/visualization and an approximated hybrid rendering (isosurfaces and direct volume rendering, using the Projected Tetrahedra method).

We therefore used the interval tree data structure in $\text{TAn}$ to speed up isosurface fitting. The results are presented in Tables 1 and 2. Since we did not have true irregular datasets available, we tested the method on tetrahedral meshes obtained from either regular or curvilinear datasets by subdividing each hexahedron into tetrahedra.

Numerical results have been obtained on an SGI Indigo (100MHz R4000 cpu, 8K instruction and 8K data caches, 1MB secondary cache, 64MB RAM).

Table 1 reports data on the complexity of both the datasets used and the associated interval trees: the resolution of the datasets ($n$, the number of sites); the number $m$ of intervals, which is equal to the number of tetrahedral cells; the depth and the number $h$ of nodes of the interval tree; and the time (in seconds) required to build this data structure.

The space complexity can be simply calculated from these data using the space complexity defined in Section 4, $4h + 4m$ memory words. The space required to store the interval tree for the Blunt\(^7\) dataset is therefore 988K memory words, and Bucky requires 783K memory words. To give an idea, the respective sizes of a sophisticated tetrahedral representations (geometry, cell adjacency, face equations etc., as defined in the previous section) of the two datasets is $4.670K$ and $3.665K$ memory words respectively while the simplest representations (vertices geometry and field value plus vertex indexes for each cell) requires, respetively, $1.040K$ and $818K$ memory words.

Table 2 reports the efficiency of the interval tree in isosurface fitting. The first three columns of the table specify: the threshold selected (defined as a value in the domain of the scalar field $W$); the number of cells which are intersected by the isosurface (active cells); and the number of facets fitted. Column 4 - 6 show the times (in CPU seconds) required to traverse and classify the cells with or without the use of the interval tree.

\(^1\)Six tetrahedra are produced in average if a Delaunay approach is adopted to build the tessellation; a decomposition of a regular grids into 5 tetrahedra per hexahedral cell is possible as well.

\(^2\)The $\text{TAn}$ system is the result of a scientific cooperation between I.E.I., L.M.A. and CNUCE, all institutes of the C.N.R. It has recently been released in the public domain. The software, compiled for SGI workstations, and downloadable paper preprints are available on our World Wide Web site at URL: http://miles.cnuce.cnr.it/cg/homepage.html.
Finally, the times of the whole fitting process are presented in the last three columns. Fitting times include: classification; linear interpolation on the edges to compute vertices (no local coherence is taken into account in the current implementation, thus vertex interpolation is replicated on adjacent cells); indexing of vertices for the construction of the mesh representation (vertex and facet lists); normals computation. In particular, normal estimation on isosurface vertices is very expensive because we first compute the normals to the facets and then, for each vertex, compute the average of all of the incident facet normals. The fitting times can therefore be significantly reduced with a more accurate implementation, as briefly described in the next section.

The traversal only results show the efficiency improvements obtained by adopting the interval tree: traversal and classification times using the interval tree are from 1/10 to a minimum of 1/5 of the non optimized time. Figure 5 shows that times increase linearly with the size of the output (number of facets in the isosurface).

### 6 CONCLUSIONS

We have presented and tested a speedup method for isosurface extraction that is based on the interval tree. The method achieves optimal output-sensitive time complexity in extracting active cells, while it requires a moderate memory overhead on realistic data structures for irregular volume data (tetrahedral models).

The method considerably improves the performance of the traversal phase with respect to standard Marching Tetrahedron. However, the total speedup in the whole isosurface extraction can only be moderate, unless additional techniques are applied to overcome the bottleneck of fitting phases for each active cell. We are now working to a more accurate implementation, with the goal of speeding up vertex geometry and normal computation, as briefly described below.

- Fitting times can be considerably reduced by saving and reusing local computations for each face or edge. In this case, some tradeoff between memory requirements to store such values, and time requirements to retrieve them, must be considered. Indeed, if memory is reserved in the primary data storage for storing interpolated vertices for each tetrahedral cell then we obtain a redundant representation with a relevant overhead, though such information will only be used for a reduced set of cells (the active ones). Storing intersections for each edge avoids replication between adjacent cells, but involves a more complex data organization since it requires all edges to be represented explicitly. A possible compromise which requires an overhead proportional to the number of active cells, and an almost constant query time, is the hash–based technique proposed in [15].

- Fitting times may also be reduced by adopting an alternative approach for the estimation of normals, based on the direct computation of gradients on the tetrahedral mesh grid nodes and their interpolation on each cell edge.

Since any speedup technique requires some additional information to be stored, hence some memory overhead, any solution should always be selected according to the ratio between the size of the data and the size of the primary storage available on the target hardware. In this context, it may be worth developing flexible data structures, which allow a program to automatically load more or less information, and to select the right speedup techniques, depending on the specific dataset and platform it is running on.

Another approach may be to adopt an approximate fitting modality to simplify numerical computation in the fitting phase. One possibility is to adopt a generalization of the Discretized Marching Cube [10] to tetrahedral models: for each edge intersected by the isosurface, it is assumed that the intersection always occurs at its midpoint. The intersection point can be either pre-computed and stored (with a consequent overhead), or computed on-the-fly with only three sums and three divisions by two. Of course, this technique may affect the quality of the isosurface obtained.

Finally, the generality of the proposed approach should be noted: the interval-based strategy makes the solution suitable for regular or curvilinear datasets represented using hexahedral meshes as well. The data structures and the searching algorithm can be applied to hexahedral decompositions without any significant modifications.
but the memory overheads are in this case excessive. We are currently working on the reduction of the space complexity of this structure exploiting the regular structure of hexahedral dataset and comparing it with the results that can be obtained by adopting hierarchical regular data structures (octrees or pyramids).

ACKNOWLEDGEMENTS

This work was partially financed by the Progetto Coordinato “Modelli multirisoluzione per la visualizzazione di campi scalari multidimensionali” of the Italian National Research Council (CNR).

We wish to thank those who have made available on the public domain the datasets used in the evaluation of the proposed techniques. BluntFin, a 40×32×32 curvilinear dataset, is produced and distributed by NASA–Ames Research Center. Bucky, a 32×32×32 subsampling of a regular rectilinear dataset is courtesy of AVS International Center.

References


