1996

Fast Isocontouring for Improved Interactivity

Chandrajit L. Bajaj

Valerio Pascucci

Daniel R. Schikore

Report Number:
96-024

FAST ISOCONTOURING FOR IMPROVED INTERACTIVITY

Chandrajit L. Bajaj
Valerio Pascucci
Daniel R. Schikore

Purdue University
Department of Computer Sciences
West Lafayette, IN 47907

CSD TR-96-024
May 1996

Fast Isocontouring For Improved Interactivity*

Chandrajit L. Bajaj† Valerio Pascucci† Daniel R. Schikore†

Shastra Lab & Center for Image Analysis and Data Visualization
Department of Computer Sciences
Purdue University

ABSTRACT

We present an isocontouring algorithm which is near-optimal for real-time interaction and modification of isovalues in large datasets. A preprocessing step selects a subset S of the cells which are considered as seed cells. Given a particular isovalue, all cells in S which intersect the given isocontour are extracted using a high-performance range search. Each connected component is swept out using a fast isocontour propagation algorithm. The computational complexity for the repeated action of seed point selection and isocontour propagation is $O(\log n' + k)$, where n' is the size of S and k is the size of the output. In the worst case, $n' = O(n)$, where n is the number of cells, while in practical cases, n' is smaller than n by one to two orders of magnitude. The general case of seed set construction for a convex complex of cells is described, in addition to a specialized algorithm suitable for meshes of regular topology, including rectilinear and curvilinear meshes.

Keywords: Visualization, Scalar Data, Isocontouring, Range Query

1 INTRODUCTION

A wide range of techniques have been developed for the visualization of scalar fields defined by a function $\mathcal{F}(\mathrm{x})$ over a given domain D. One of the most common and useful approaches is to compute and display isocontours $C = \{\mathrm{x} | \mathcal{F}(\mathrm{x}) = w\}$. It is estimated that in a 3D domain D, the average number of cells intersected by an isocontour will be $O(n^{2/3})$ [5], where n is the number of cells, which can be generalized to $O(n^{(d-1)/d})$ for a d-dimensional domain. Hence algorithms which perform an exhaustive covering of cells are found to be inefficient, spending a large portion of time traversing cells which do not contribute to the contour.

This fact has a great impact on the amount of interaction which is possible between the user and the visualization. Interactive manipulation and control of visualization parameters allow the user to more quickly locate a region of interest and in general provide the user with a better understanding of the scalar field as a whole from display of contours, which inherently represent only a subset of the entire field.

We present an automated isocontour extraction algorithm of near-optimal complexity for the case of multiple isovalue queries. A one-pass preprocessing step through the volume data selects a subset S of the volume cells which are maintained as candidate seed cells. The general case for a cell complex of arbitrary topology is described, as well as a simplification for structured data. For any isovalue, it is guaranteed that each connected component of the isocontour will intersect at least one cell in S. A subsequent preprocessing step generates a search structure for the cells in S, permitting $O(\log n' + k)$ search for all cells in S which contain a given isovalue, where n' is the size of the S and k is the size of the output. Cells which intersect the given isovalue are used as start cells for an isocontour propagation algorithm, visiting only the cells which are intersected by the isocontour, resulting in an overall complexity of $O(\log n' + k)$. We present results and statistics for volume data from several domains.

2 RELATED WORK

Extraction of isocontours from scalar data has received a great deal of attention in recent years. Among the contributions to the field are methods for classifying and computing intersections within a single cell [7, 11, 12, 17]. Here we are concerned primarily with the search for intersected cells,

[1] See also
`http://www.cs.purdue.edu/research/shastra`

[2] Department of Computer Sciences, Purdue University, West Lafayette, IN 47907-1398 USA. Email: {bajaj,pascucci,drs}@cs.purdue.edu

while the choice of triangulation technique can be chosen based on the data characteristics and topological needs of the application.

The majority of the techniques for accelerating the extraction of isocontours do so by limiting the number of cells that are visited, thereby reducing the overhead associated with the inevitable search for cells which are intersected by the isocontour.

Wilhelms and Van Gelder[18] use an efficient partial octree partitioning of a structured mesh with hierarchical $[min, max]$ data in order to quickly locate cells which are intersected by the isocontour while skipping large regions of space with no contribution to the contour. Such a geometric decomposition works well for smooth data with high spatial coherence but suffers when applied to noisy data.

Recent techniques have concentrated on processing of the *value space* of the cells rather than the *geometric space*. Giles and Haimes [3] describe a method which forms two sorted lists of cells, one by minimum value and the other by maximum value. The maximum cell range, Δw, is computed, which allows the limitation of the search to cells with minimum value in the range $[w - \Delta w, w]$ for a given isovalue w. Cells in this range which do not cross the threshold are removed by inspection. For small changes in w, an incremental approach of adding new cells and removing cells outside the given w gives improved performance. Evident in this approach is the fact that a single cell with a large Δw drastically reduces the effectiveness of the technique when specifying a random isovalue.

Gallagher describes a technique called *span filtering* [2], in which the entire range space is divided into a fixed number of *buckets*. Cells are grouped into buckets based on the minimum value taken on by the function over the cell. Within each bucket, cells are classified into several lists based on the number of buckets which are *spanned* by the range of the cell. For an individual isovalue, cells which fall into a given bucket only need be examined if their span extends to the bucket containing the isovalue.

Itoh and Koyamada compute a graph of the extrema values in the scalar field [5]. Every connected component of an isocontour is guaranteed to intersect at least one arc in the graph. Isocontours are generated by propagating contours from a seed point detected along these arcs. Noisy data with many extrema will reduce the performance of such a strategy.

Shen and Johnson describe a *Sweeping Simplices* algorithm which maintains two lists of cells, one sorted by minimum cell value, the other by the maximum cell value [14]. For a given isovalue, a binary search in the minimum value list determines all cells with minimum value below the isovalue. Pointers from the minimum value list to the maximum value list are followed to set a corresponding bit for each candidate cell. At the same time, the candidate cell with the largest maximum value which is less than the isovalue is determined. As a result, all marked (candidate) cells to the right of this cell in the maximum list must intersect the contour, as they have minimum value below the isovalue and maximum value above the isovalue. Optimizations are performed when the isovalue is changed by a small delta.

Livnat, Shen, and Johnson describe a new approach which processes the cells into a 2D *min-max span space* [6]. Cells are preprocessed into a *Kd-tree* which allows $O(\sqrt{n} + k)$ query time to determine the cells which intersect the contour, where k is the size of the output. It is reported that in the average case, k is the dominant factor, providing optimal complexity. The same authors, with Hansen, have recently described an advancement which demonstrates improved empirical results by using an $L \times L$ lattice search decomposition in span space, in addition to allowing for parallel implementation on a distributed memory architecture [13].

A similar approach to ours has been developed independently by van Kreveld [16], in which seed sets are computed for the specialized case of a triangular mesh in two dimensions representing terrain for GIS applications. An *interval tree* is used to perform the search for intersected seed cells, resulting in worst-case complexity of $O(\log n + k)$. Our approach differs in the seed selection technique, which we generalize to a cell complex of arbitrary topology, in addition to developing a specialized simplification for regular grids.

In summary of the related work, isocontour acceleration techniques attack the problem of minimizing the search phase by forming a structure based either on the embedding space of the geometric mesh or the 2D span space of the scalar field. Characterizations of such approaches can be made based on how well they handle noisy as well as smooth functions, and whether the technique yields higher performance for consecutive isovalues which are close to one another.

Our approach is to initially extract a subset of cells S from the given volume such that for any given isovalue w, every connected component of the isocontour defined by w will intersect a cell in S. The set of cells S are preprocessed into a range search structure defined by the minimum and maximum value of each seed cell. From this structure, cells for a given isovalue can be extracted in $O(\log n' + k)$ time. From each selected cell, one or more connected components of the isocontour are extracted by propagation through cell adjacencies [4].

3 ALGORITHM OVERVIEW

The approach we take is based on the formalization and unification of three known techniques. The three leading ideas we are retaining are the following:

1. The extraction of an isocontour does not require searching all the cells of the mesh [5].

2. To improve the efficiency of the cell extraction, it is necessary to define a search structure over the set of cells [18].

3. The search space we need to work on is not the embedding space of the original mesh but the two dimensional span space [6].

Exploiting these three main ideas we get the following high level sketch of an isocontouring algorithm:

1. (Preprocessing A)
   Reduce the set of cells to a subset S that encompasses at least one cell per connected component of each isocontour.

2. (Preprocessing B)
   Construct an efficient search structure over the cells in the set S.

3. (Step 1)
   Given the scalar value w, perform a logarithmic search on the set S to find all the cells in S which intersect the isocontour of value w.

4. (Step 2)
   For each cell c found in step 1, trace the entire connected component of the isocontour intersected by c.

This approach allows us to obtain near-optimal worst case time complexity along with an even better performance in average non-perverse cases. If k is the size of the output and n is the size of the input mesh, the worst case complexity we get is $O(\log n + k)$. In practical cases we have observed a timing that grows linearly with the size of the output, implying an optimal average case complexity of $O(k)$.

The approach is applicable to any unstructured grid of cells on which a scalar field is defined. The scalar field itself is only assumed to be continuous. We only need a function R that, for each cell c in the mesh, returns the range $R(c)$ of all possible values assumed by the field on that cell. On the basis of this general framework, we then explore a simplified version of the method where a regular 2D or 3D grid is used as the mesh and the scalar field is approximated as a piecewise linear function interpolating the values sampled on the vertices of the mesh.

It is important to note how part A of the preprocessing is strictly connected with step 2 of the isocontouring process. In fact the cells that do not need to be stored in the set S are the cells that can be captured during step 2 performed from some cell in S. In the same way step B of the preprocessing is coupled with step 1 of the contouring algorithm, as in the former the search structure which is used in the latter is constructed. Details of the general algorithm for seed set construction will be described in the next section, followed by a simplified approach devised for regular grids.

3.1 Contour Propagation

Extraction of 2D surfaces from 3D data by *mesh propagation* is described by Speray and Kennon [15] for the case of arbitrary slices in unstructured meshes, while others have applied similar techniques to isocontour extraction [1, 4, 5].

The central idea is that, given an initial cell which contains the surface of interest, the remainder of the surface can be efficiently traced performing a breadth-first search in the graph of cell adjacencies, as illustrated for a 2D contour extraction in Figure 1. Use of a breadth-first search keeps the memory requirements minimal, as only the "advancing front" of the surface needs to be stored at any one time. One advantage to using a propagation approach over other techniques is that surfaces are easily transformed into a *triangle strip* representation for more efficient rendering [4]. Also of importance is the fact that shared vertices between cells are more efficiently located, as we are considering only a single closed contour at any given time. In our implementation, carefully hashed indexing of the advancing front allows us to efficiently eliminate recomputing intersections when the advancing front closes on itself, completing the extraction of a connected component. Similar to related caching techniques [1, 18], the cache is made efficient by discarding entries which are known to not be referenced again, based on the maximum number of cells which share a given edge.

3.1.1 Cell Connectivity

Given such a contour propagation scheme, we can abstract the concept to a cell connectivity graph defined in terms of the scalar field defined on the mesh. In this way we can easily determine a subset of the mesh cells from which all the possible isocontours of the scalar field can be computed using the given propagation scheme. Note that on the basis of the defined propagation scheme the connectivity graph is

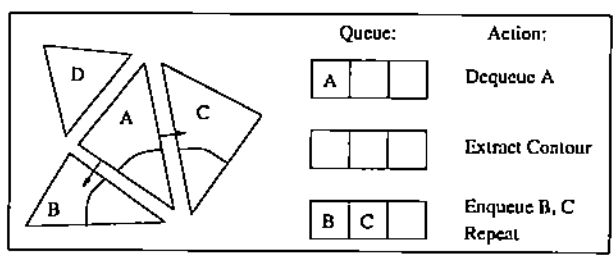


FIGURE 1: Illustration of contour propagation

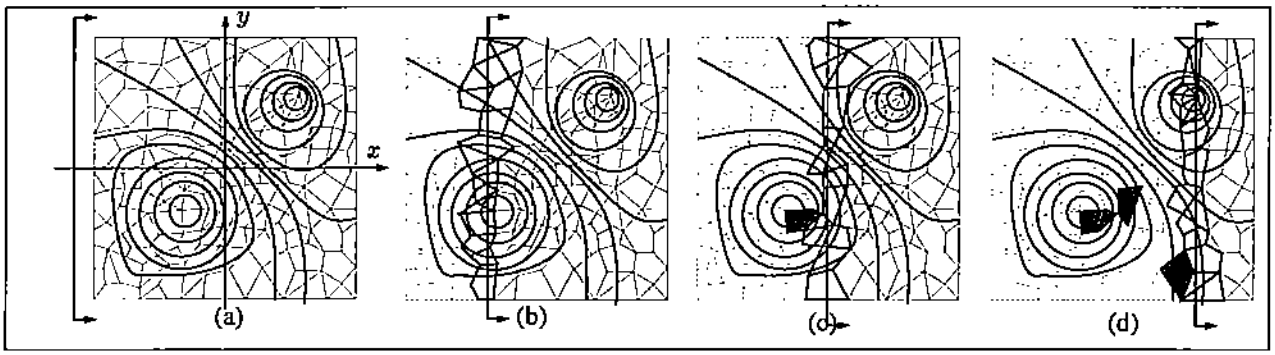


FIGURE 2: Sweep process on the mesh space along the x direction. As the line sweeps the seeds are collected (marked in the picture).

simply a labeled adjacency graph of the mesh cells. The use of a different propagation scheme implies the construction of a connectivity graph different from the adjacency graph. In general, to define the connectivity graph we assume:

1. The function $\mathcal{F}(x_1, \ldots, x_n)$ defining the scalar field of our n-dimensional mesh is continuous.
2. All the cells of the mesh are connected.
3. A function $R(c)$ is available that, for any given cell c of the mesh, returns the *range* of values assumed by $\mathcal{F}$ over the domain of c. Note that, since $\mathcal{F}$ is continuous, the range returned is always an interval $[min, max]$.
4. For each pair of connected (adjacent) cells (c_1, c_2), we define a *connecting interval*: $[min', max'] \subseteq R(c_1) \cap R(c_2)$ such that if the cell $c_1(c_2)$ is processed for a value $w \in [min', max']$, then the cell $c_2(c_1)$ will be also processed for the same value w. This is essentially the information we get from the contour propagation scheme.

Based on the above information, we construct a labeled graph G. For each cell c in the mesh, we have a node c in G that is labeled $R(c) = [min, max]$. For each pair of connected (adjacent) cells (c_1, c_2), there is an arc f in G connecting c_1 to c_2 labeled $R(f) = [min', max']$. We name the arc f because, with respect to the above propagation scheme, each arc of G represents the facet f along which the cells c_1 and c_2 are adjacent. In this case, the connecting interval of such an arc is the range $R(f)$ of the scalar field $\mathcal{F}$ on such facet f.

With reference to the graph G, we define connectivity relations between the nodes of the graph and hence between the corresponding cells of the underlying mesh. All cells which intersect the same connected component of a contour of isovalue w we call w-connected. Formally we have the following recursive definition:

Definition 1 *Consider a scalar value w and two nodes c_1, c_2 of G. c_1 and c_2 are said to be* w-connected *if one of the two following conditions holds:*

(a) c_1 and c_2 are connected by an arc f such that $w \in R(f)$.

(b) There exists a node c_3 that is w-connected to both c_1 and c_2.

We can extend the concept of w-connectivity between pairs of cells to the connectivity of a set of cells with respect to a range of values.

Definition 2 *Consider a subset S of the nodes of G and a node $c \in G$. The node c is* connected *to S if, for any $w \in R(c)$, there exists a node $c' \in S$ that is w-connected to c.*

3.1.2 Seed Sets

We now characterize some particular subsets, called *seed sets*, of the cells of a mesh in terms of the connectivity properties defined in the previous subsection. The seed sets are important because any isocontour of the entire original mesh can be traced by propagating from the cells of any seed set.

Definition 3 *A subset S of the nodes of G is a* seed set *of G if all the nodes of G are connected to S.*

If we wish to determine quickly all the cells of a mesh whose range contains a particular scalar value w, we can proceed as follows:

1. search for all the nodes $c \in S$ such that $w \in R(c)$;

2. starting from the nodes we have found and using the w-connectivity relation on the graph G (that is the contour propagation scheme), we find all the cells of the mesh whose range contains w.

To reduce the search time we need to reduce the cardinality of the seed set S as much as possible. Toward this end we will apply the following property:

Property 1 *If S is a seed set and $c \in S$ is a cell connected to $S - \{c\}$, then $S - \{c\}$ is a seed set.*

Proof: By hypothesis we have that c is connected to $S-\{c\}$. Also, from Definition 1(b), we have that any cell which is w-connected to c is also w-connected to some cell in $S-\{c\}$. Hence $S - \{c\}$ is a seed set. ◦

Property 1 provides us with a method to reduce the size of a seed set. If we wish to find a small seed set, we can start with the entire set of the cells – that is the largest seed set – and keep removing cells until we achieve a *minimal seed set*. Note that a minimal seed set is not the seed set with the minimum number of cells but a seed set from which we cannot remove any cell to obtain a new seed set.

The repeated application of Property 1 requires the knowledge at each step of the connectivity relations within the current seed set. Thus, we may start from the initial graph G. At each step, we remove the selected cell c along with all its incident arcs and add some new arcs between pairs of cells that were connected to c to take into account the connectivity relations induced by c on $G - \{c\}$. In particular, if two cells c_1 and c_2 are both connected to c with arcs f_1 and f_2, then the removal of c requires also the removal of f_1 and f_2 and potentially the insertion of a new arc f connecting c_1 to c_2. This new arc f needs to be inserted if $R(f_1) \cap R(f_2) \neq \emptyset$ (a case in which Definition 1(b) applies). If this condition is true, then the new arc is added with label $R_f = R(f_1) \cap R(f_2)$. If we proceed in this way, it becomes simple to determine if Property 1 can be applied. We can remove a cell c of the current seed set if:

$$\bigcup_{i=1}^{k} R(f_i) = R(c)$$

where $f_1, \ldots, f_k$ are all the arcs incident to the cell c in the reduced graph of the current seed set.

Given this general reduction scheme, we still have freedom to select the cells to be removed in any order. We can use a greedy approach, removing first the cells that we consider less likely to belong to a minimal seed set – for example the cells that have narrower range. In this way we can assume that the minimal seed set we achieve is not much larger than the seed set with the minimum number of cells. On the other hand, we can use this freedom to make the algorithm as simple as possible (a very important property in actual implementations).

A simple and efficient strategy for computing a small seed set S is to apply a sweep in the grid space while maintaining only the part G' of the graph G relative to the cells of the grid intersecting the sweep hyperplane (note that the complete graph G does not need to be stored because it is equal to the adjacency graph of the grid cells). For a 2D unstructured grid, such as in Figure 2(a), the sweep hyperplane is a line parallel to the y direction moving along the x direction. Figure 2(b) shows the cells of the mesh that need to be represented in G' in thick outline. Those are the cells on which Property 1 is being tested. The connectivity relations among cells on the right of the sweep line (thin solid lines) do not need to be stored in G' because they are still like in G. The connectivity relations among the cells on the left of the sweep line (thin dotted lines) do not need to be stored in G' since such cells have already been discarded. Figure 2(c)-(d) shows how during the sweep process the cells that cannot be discarded are marked as seed cells.

In a regular grid the sweep process can be simply implemented as a traversal of the grid by rows using a regular marching scheme. In the following section we will examine such a case in which we take a simplified approach specialized for 2D and 3D grids of regular topology. The technique is applicable to both rectilinear and curvilinear grids as we depend only on the topological structure of the grid.

3.2 Generating Seed Sets

In theory, the use of an $O(\log n)$ search for seed points in a seed set does not require that we extract a subset of the cells: the complexity is no worse if we use the entire set. In practice, because of the overhead of storing the search structure for the entire set of cells, in addition to our ability to propagate an isocontour from one cell to the next efficiently, we are interested in constructing a small set of cells, with the only requirement that each connected component of an isocontour is represented in the selected set.

We introduce a simplification of the connectivity graph technique described in the previous section for determining a seed set S. The simplification does not require that we store the entire graph, but instead we maintain a subset of the information from the graph which can be locally propagated from cell to cell using simple rules when marching in a regular order with temporary storage complexity of $O(n^{(d-1)/d})$. We begin with all cells c in the set S. We associate with each seed cell a computed range $\mathrm{T}(c)$, which represents the range of values for which the given cell is a seed cell. Initially, we have $\mathrm{T}(c) = R(c)$, the entire range of the cell, hence S is a seed set. We present an incremental seed elimination technique to reduce the seed set S. The reduction and re-

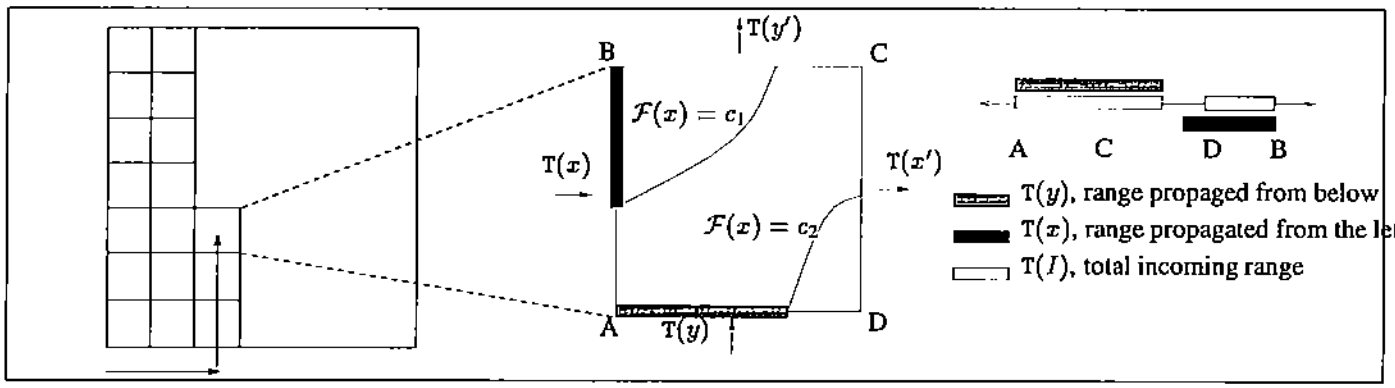


FIGURE 3: Illustration of range propagation for a single cell

moval of seed cells is based on propagation of *responsibility ranges* of isovalues. The information propagated from cell to cell in marching order is a range T for each dimension of the regular grid. An *incoming range* T represents the range of values w for which responsibility has been propagated to the current cell from the neighboring cells. The incoming range is always a subset of the range of the shared face in the direction of propagation. The complement of the incoming range in the direction which varies fastest consists of values w for which the current cell is w-connected to either (i) a processed cell which remains in the seed set or (ii) an unprocessed cell to which responsibility for the value w has been propagated. An *outgoing range* represents the responsibility range which is propagated from the current cell to a neighboring cell. Illustrated for the 2D case in Figure 3, the marching order is Y varying fastest, X varying slowest.

We describe the processing of a cell c at index (i, j) in a topologically regular grid of dimension (n_x, n_y). Boundary conditions are handled directly through the following notation, defined for simplicity:

1. $\mathrm{T}(f_u)$ represents the range of the incoming face in the U direction, where U is an arbitrary dimension.

2. $\mathrm{T}(u)$ represents the incoming range propagated in the U direction. In the case of the boundary condition $u = 0$, we take $\mathrm{T}(u) = \mathrm{T}(f_u)$.

3. $\overline{\mathrm{T}(u)}$ represents the complement of $\mathrm{T}(u)$ with respect to the range $\mathrm{T}(f_u)$ of the shared face, or $\mathrm{T}(f_u) - \mathrm{T}(u)$. Note that the propagated range $\mathrm{T}(u) \subset \mathrm{T}(f_u)$.

4. $\mathrm{T}(f_{u'})$ represents the range of the outgoing shared face in the U direction. In the boundary case when there is no adjacent cell in the outgoing U direction ($u = n_u - 2$), we assign $\mathrm{T}(f_{u'}) = \emptyset$, indicating that no propagation may occur in the given direction.

5. $\mathrm{T}(u')$ represents the range propagated from the current cell to the outgoing adjacent cell in the U direction.

We first compute the combined incoming range $\mathrm{T}(I)$, and complement range $\overline{\mathrm{T}(I)}$:

$$\mathrm{T}(I) = (\mathrm{T}(y) \cup \mathrm{T}(x)) - \overline{\mathrm{T}(y)} \tag{1}$$

$$\overline{\mathrm{T}(I)} = (\mathrm{T}(f_x) \cup \mathrm{T}(f_y)) - \mathrm{T}(I) \tag{2}$$

$\mathrm{T}(I)$ represents the subset of incoming isovalues which cell c must either account for in the seed set S or defer responsibility for by propagation through $\mathrm{T}(x')$ and $\mathrm{T}(y')$. The subtraction of $\overline{\mathrm{T}(y)}$ in Equation 1 above is justified based on the algorithm for range propagation presented below. For all $w \in \overline{\mathrm{T}(I)}$, there either exists a processed cell in S which is w-connected to c or the value w has already been further propagated, and hence $w \in \overline{\mathrm{T}(I)}$ need not be considered in processing c. This leads to the definition of $\mathrm{T}(R)$, representing the entire range of values which make up the responsibility range of cell c.

$$\mathrm{T}(R) = R(c) - \overline{\mathrm{T}(I)} \tag{3}$$

For $w \in \mathrm{T}(R)$, we must take care that c remains w-connected to S in order to maintain the property that S is a seed set. We also compute $\mathrm{T}(P)$, which represents the combined range of isovalues which may be further propagated through outgoing faces:

$$\mathrm{T}(P) = \mathrm{T}(f_{x'}) \cup \mathrm{T}(f_{y'}) \tag{4}$$

This leads to the following greedy algorithm for deferring seed cell selection through propagation of responsibility. Through the processing of a cell c, we maintain the invariant that S is a seed set.

```
if (T(R) ⊆ T(P)) then
    { Cell c can be safely removed from S }
    S = S − c
    { Propagate responsibility ranges }
    T(x') = T(f_x') ∩ T(R)
    T(y') = T(f_y') ∩ (T(R) − T(x'))
```

else
{ *Cell c must remain in the seed set* }
$\mathtt{T}(c) = \mathtt{T}(R)$
$\mathtt{T}(x') = \emptyset$
$\mathtt{T}(y') = \emptyset$
end

Proof: (S remains a seed set after processing of cell c)

Case 1 ($\mathtt{T}(R) \subseteq \mathtt{T}(P)$) - Recall that cell c is w-connected to a processed seed cell for $w \in \overline{\mathtt{T}(I)}$. Through propagated responsibility ranges, we have that c is w-connected to the remaining (unprocessed) seed set for $w \in \mathtt{T}(x') \cup \mathtt{T}(y') = [\mathtt{T}(f_{x'}) \cap \mathtt{T}(R)] \cup [\mathtt{T}(f_{y'}) \cap (\mathtt{T}(R) - \mathtt{T}(x'))] = (\mathtt{T}(f_{x'}) \cup \mathtt{T}(f_{y'})) \cap \mathtt{T}(R) = \mathtt{T}(P) \cap \mathtt{T}(R) = \mathtt{T}(R) = R(c) - \overline{\mathtt{T}(I)}$. Thus, c is connected to $S - \{c\}$, and by Property 1, $S - \{c\}$ is also a seed set, maintaining the invariant property.

Case 2 (Cell c remains in the seed set) - Cell c is trivially w-connected to S for $w \in \mathtt{T}(c) = \mathtt{T}(R) = R(c) - \overline{\mathtt{T}(I)}$. From the input conditions, we have that c is w-connected to a processed cell which remains in S for $w \in \overline{\mathtt{T}(I)}$. Thus, c is w-connected to S for $w \in R(c)$, maintaining the invariant property that S is a seed set.

◇

In the first case, the propagated range $\mathtt{T}(P)$ includes the responsibility range $\mathtt{T}(R)$ in its entirety, and cell c is removed from the seed set S. The responsibility range is propagated through the outgoing faces by the computation of $\mathtt{T}(x')$ and $\mathtt{T}(y')$. Note that the propagated ranges are disjoint and that the preference is to propagate the range in the X direction. It is this preference which allows us to remove $\overline{\mathtt{T}(y)}$ in equation (1). For all $w \in \overline{\mathtt{T}(y)}$, the associated w-connected component is either accounted for by a processed cell in the seed set S, or responsibility has been propagated to an unprocessed cell, hence w need not be considered for the current cell. The same cannot be said for $\overline{\mathtt{T}(x)}$, because the precedence of propagation indicates that responsibility for values $w \in \overline{\mathtt{T}(x)}$ may, through some path of responsibility propagation, ultimately be propagated through $\mathtt{T}(y)$. Consider the case of Figure 3, and suppose that the value A is a local minimum. Values $w \in \overline{\mathtt{T}(x)}$ overlap with the range $\mathtt{T}(y)$, providing incoming information which appears to conflict. In fact we cannot make use of the range $\mathtt{T}(u)$, where u is other than the direction which varies fastest in the marching order.

The second case above occurs when cell c cannot propagate the entire incoming range. Cell c remains in the set S, though $\mathtt{T}(c)$ is reduced to exclude the complement ranges which have been propagated elsewhere. In this case the empty set is propagated to outgoing edges, indicating that all values on shared faces are accounted for in the seed set S.

As described above, the *range propagation* method for selecting seed cells requires $O(n^{(d-1)/d})$ storage to maintain the propagated ranges for a sweeping line or plane in 2D or 3D. Note that our use of range subtraction may result in ranges with two disconnected components. In practice, disconnected ranges may either be maintained or closed by taking the smallest range which contains the entire disconnected range. Maintaining the disconnected range effectively requires that multiple seeds be processed into the search structure, increasing the number of seeds, while merging disconnected ranges simply means that two or more cells which are w-connected may be selected for inclusion in the seed set S. Of course, this greedy technique does not guarantee otherwise in the case that disconnected ranges are maintained. In our implementation, we maintain disconnected ranges through the seed cell selection, closing each range which is ultimately selected to remain in the seed set S. In practice the number of seed cells with disconnected ranges does not exceed 10% of the seed cells, and the number of seed cells does not exceed 10% of the data, as presented in the results in Section 4.

Results for a 2D regular mesh are illustrated in Figure 4. The relatively smooth function is sampled on a grid of size 64×64. Figure 4 (upper) shows the 206 seed cells chosen by the marching seed selection method. Figure 4 (lower) shows the decrease to 56 seed cells which is achieved using a more sophisticated method currently under development. Our preliminary results on small datasets have shown decreases in $|S|$ by factors ranging from 2.5 (noisy 2D MRI) to 20 (3D SOD data presented in the results section) over the marching seed selection method which is in current use, however the current implementation of the more sophisticated selection algorithm is too computationally expensive to be considered practical.

3.3 Range Queries

In this brief section we analyze the problem of selecting all the cells of a given set S whose range contains an assigned value w. This problem is independent from the characteristics of the set S that can be the entire set of the mesh cells or any subset, e.g. a seed set. The important aspect to focus on is the selection criterion.

A cell c of S is selected iff $w \in \mathtt{R}(c)$.

To achieve a good search scheme, it is important to define what our search space is. As observed in [2], we do not need to search for the required cells in the embedding space of the cells since we select them only considering their range. In [6], the two dimensional span space is considered the search space, where each cell is represented by a point whose coordinates are the two extremes $[min, max]$ of the cell range. The search complexity achieved with a $Kd-tree$

FIGURE 4: Results of two seed selection techniques

in this way is $O(\sqrt{n}+k)$ where n is the number of cells in S and k is the number of cells reported (also in this case S encompasses all the cells in the mesh and is not reduced to a seed set). We want to go a step further and notice that the range of a cell is not simply defined by a pair of numbers (to be mapped to 2D points) but is actually an interval that can be mapped to the 1D line. That is, we assume as search space a set of 1D segments instead of a set of 2D points. In this way we can use well known search structures such as the *segment tree* (see e.g. [9] or [10]) or the *interval tree*, a specific case of a priority search tree [8]. Examples of each search tree are given in figure 5 for a small set of intervals and briefly discussed in the following paragraphs.

In a segment tree, the set of min and max values of the segments are simply sorted (along the 1D line), and a standard binary search tree is constructed over them. Additional information is then stored in each node of the tree. If a node a is the root of a subtree that spans values in the range $[min_a, max_a]$ and a cell c has a range that contains $[min_a, max_a]$, then the node a contains the label c. With such a structure, determining the cells which span a given value w is achieved through a binary search for the w in the tree. During traversal of the tree, all labels stored in visited nodes are collected. They are the labels of all cells whose range contains w. The time complexity achieved is $O(\log n' + k)$ in the case that all n' cells have distinct min and max values, while the storage complexity is $O(n' \log n')$. These are worst case bounds, and may improve in the special case that the number of distinct values is limited, as discussed in the results in Section 4.

In an interval tree, each node holds a *split* value s, and each interval is classified as *less than* ($max < s$), *greater than* ($min > s$), or *spanning* ($min < s < max$). Intervals which span s are stored in a node in the tree, while intervals which are entirely less than (greater than) s are recursed into the left (right) subtrees. Within each node, the intervals are sorted into two lists, the first sorted by increasing min value, and the second by decreasing max value. The storage complexity of the interval tree is $O(n)$ as each interval is stored two times. A query for an isovalue w consists of performing a search for w based on the split values. At each node, one of the two lists is traversed, depending on whether w is less than or greater than s. If $w < s$, the min-sorted list is searched to determine cells with $min < w$, and the traversal continues with the left subtree. If $w > s$, the max-sorted list is searched, and traversal continues with the right subtree. Intersected cells will always appear at the left of the lists due to their sorted order.

4 RESULTS

Results were computed on a Silicon Graphics Indigo^2 IMPACT with 128Mb memory and one 250Mhz R4400 processor. The search structure used in these examples was the segment tree.

Table 1 provides characteristics of our test data suite, as well as statistics for the preprocessing stage of the algorithm. Examination of the percentage of cells which remain in the seed set reveals that the set S is one to two orders of magnitude smaller than the entire set of cells for practical real data. This observation is very important because the number of seed cells n' represents the search overhead of $O(\log n')$ for the segment tree, indicating that in practical situations the dominant complexity will be $O(k)$, where k is the size of the output. We make special note of the number of distinct seed values (min or max), because the height of the segment tree is dependent on this number alone. For the case of scalar data which take on a limited number of values, such as 8-bit integer or quantized floating point values, the $O(\log n')$ is effectively made into a constant, resulting in an optimal time complexity of $O(k)$ as well a segment tree storage complexity of $O(n')$.

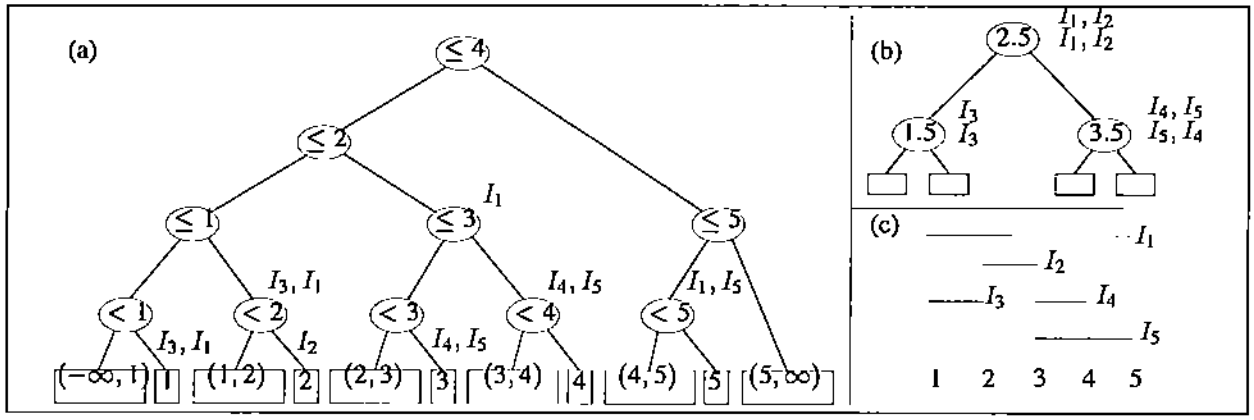


FIGURE 5: Segment tree (a) and Interval Tree (b) for the set of ranges (c)

Table 2 gives timing results on volume datasets of various sizes, with comparison to a brute force Marching Cubes approach. Times are reported in seconds and include computation of the isosurface and storage in an internal data structure. Images of each contour can be found in the color plates.

Evident in the data collected is the fact that the algorithm scales approximately linearly with respect to the number of triangles computed (or the number of cells intersected by the surface). In our implementation, performance ranges from 45K triangles/second to 97K triangles/sec, while the brute force approach has widely varying performance, from 2K triangles/sec to 40K triangles/sec. Figure 6 shows the actual performance in triangles/sec for multiple isovalue queries for the SOD dataset, demonstrating a performance which scales linearly with the number of triangles in the output. Using the same isovalues for the SOD data, Figure 7 compares our speedup (over traditional Marching Cubes) with the volume fraction, measured in triangles/cell. Evident from the plot is that our algorithm provides the greatest speedup when the surface of interest is small compared to the volume. This is consistent with the notion that for small numbers of triangles, the exhaustive search dominates the triangulation time.

5 CONCLUSIONS

We have presented a fast algorithm for computing isocontours from scalar volume data. Observed average complexity is $O(k)$, where k is the number of cells intersected by the contour. In the worst case ($\log n' > k$), the limiting factor becomes the search.

The importance of linearity with respect to the number of cells intersected by an isocontour cannot be overstated. With the ever-increasing size of volumetric data, contouring techniques which search the entire space grow with the size of the volume. Using the method we have presented, an increase in the size of the input results in an increase in computation on the order of the dimensionality of the contour. The result is that larger volumes which were prohibitive using less efficient algorithms are now accessible to the visualization user. For intermediate size volumes, the increased performance results in true interactive computation, allowing the visualization user to explore volumetric data, modifying isovalues and viewing the results in real-time, on desktop devices.

Acknowledgements

We thank Dr. Nelson Max for many constructive comments on early versions of this paper which have improved it greatly. We are also grateful for useful discussions with and suggestions from Dr. Marc van Kreveld, and the helpful comments of the anonymous reviewers.

The MRbrain, HIPIP and SOD datasets are from the Chapel Hill Volume Rendering Test Data suite. MRbrain was provided courtesy of Siemens Medical Systems, Inc., and edited by Dr. Julian Rosenman, North Carolina Memorial Hospital. HIPIP is courtesy Louis Noodleman and David Case, Scripp Clinic, La Jolla, CA. SOD is courtesy Duncan McRee, Scripps Clinic, La Jolla, CA. The EAS data is courtesy Bob Oglesby, Department of Earth and Atmospheric Sciences, Purdue University. The CT Engine data was obtained by anonymous ftp from Stanford University. The CT Cadaver is courtesy Elliot Fishman, M.D., and H.R. Hruban, M.D., Johns Hopkins Medical Institution.

This research was supported in part by NSF grants CCR 92-22467 and GER-9253915-02, AFOSR grant F49620-94-10080, ONR grant N00014-91-1-0370, and ARO grant DAAH04-95-1-0008.

References

[1] ARTZY, E., FRIEDER, G., AND HERMAN, G. T. The theory,

Data	Resolution	Seed cells	% of total	Distinct Seeds	Tree Size	Preprocessing Time (s)
MRbrain	256x256x109	539832	7.69%	2802	3085225	116.41
CT Engine	256x256x110	131765	1.86%	256	333372	92.95
CT Cadaver	256x256x300	652531	3.36%	256	1965679	271.73
CT Head	256x256x113	324449	4.46%	3115	1433219	109.63
HIPIP	64x64x64	2212	0.88%	1945	13403	3.24
SOD	97x97x116	9944	0.94%	153	39427	12.73
EAS	128x64x18	2454	3.84%	4302	19881	2.13

TABLE 1: Data and Preprocessing Characteristics

Data	Isovalue	# Tri	Marching Time (s)	Our Time (s)	Speedup	Marching Tri/sec	Our Tri/sec
MRbrain	500.5	973954	46.410	14.202	3.267	20985	68578
CT Engine	50.5	584916	43.169	6.278	6.876	13549	93169
same	200.5	142290	41.423	1.526	27.144	3435	93243
CT Cadaver	140.5	830016	121.080	12.504	9.683	6855	66380
CT Head	200.5	593456	43.443	6.672	6.511	13383	88947
same	-150.5	474378	44.342	4.867	9.110	10698	97468
HIPIP	0.1	1848	0.920	0.041	22.439	2008	45073
SOD	36.5	189590	5.225	2.310	2.261	36285	82073
EAS	-1.881	60622	1.483	0.830	1.787	40878	73039

TABLE 2: Surface Extraction Timing Statistics

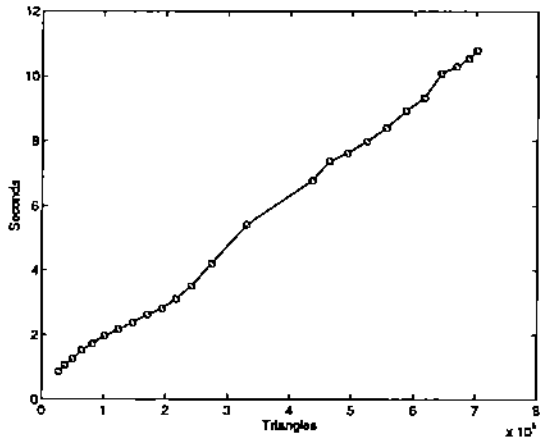


FIGURE 6: Timing for SOD data exhibiting linear performance

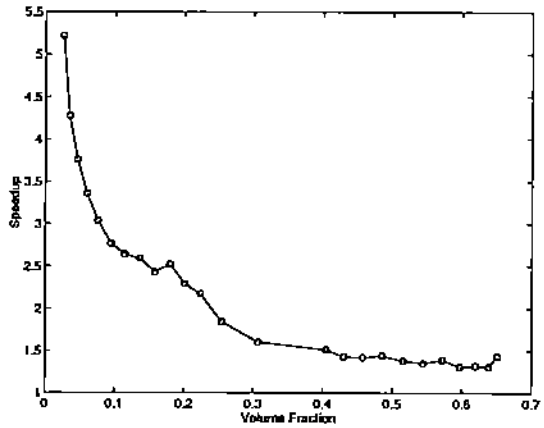


FIGURE 7: Speedup vs. Volume Fraction (triangles/cells)

design, implementation and evaluation of a three-dimensional surface detection algorithm. In *Computer Graphics (SIGGRAPH '80 Proceedings)* (1980), vol. 14, pp. 2–9.

[2] GALLAGHER, R. S. Span filtering: An efficient scheme for volume visualization of large finite element models. In *Visualization '91 Proceedings* (Oct. 1991), G. M. Nielson and L. Rosenblum, Eds., pp. 68–75.

[3] GILES, M., AND HAINES, R. Advanced interactive visualization for CFD. *Computing Systems in Engineering 1*, 1 (1990), 51–62.

[4] HOWIE, C. T., AND BLAKE, E. H. The mesh propagation algorithm for isosurface construction. *Computer Graphics Forum 13*, 3 (1994), 65–74. Eurographics '94 Conference issue.

[5] ITOH, T., AND KOYAMADA, K. Automatic isosurface propagation using an extrema graph and sorted boundary cell lists. *IEEE Transactions of Visualization and Computer Graphics 1*, 4 (Dec. 1995), 319–327.

[6] LIVNAT, Y., SHEN, H. W., AND JOHNSON, C. R. A near optimal isosurface extraction algorithm for unstructured grids. *IEEE Transactions on Visualization and Computer Graphics 2*, 1 (1996), 73–84.

[7] LORENSEN, W. E., AND CLINE, H. E. Marching cubes: A high resolution 3D surface construction algorithm. In *Computer*

Graphics (SIGGRAPH '87 Proceedings) (July 1987), M. C. Stone, Ed., vol. 21, pp. 163–169.

[8] MCCREIGHT, E. M. Priority search trees. *SIAM J. Comput. 14* (1985), 257–276.

[9] MEHLHORN, K. *Data Structures and Algorithms 3: Multi-dimensional Searching and Computational Geometry*, vol. 3 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, Heidelberg, West Germany, 1984.

[10] MULMULEY, K. *Computational Geometry: An Introduction Through Randomized Algorithms*. Prentice Hall, Englewood Cliffs, NJ, 1994.

[11] NATARAJAN, B. K. On generating topologically consistent isosurfaces from uniform samples. *The Visual Computer 11*, 1 (1994), 52–62.

[12] NIELSON, G. M., AND HAMAAN, B. The asymptotic decider: Resolving the ambiguity of marching cubes. In *Visualization '91 Proceedings* (Oct. 1991), G. M. Nielson and L. Rosenblum, Eds., pp. 83–91.

[13] SHEN, H.-W., HANSEN, C. D., LIVNAT, Y., AND JOHNSON, C. R. Isosurfacing in span space with utmost efficiency. In *Visualization '96 Proceedings (to appear)* (Oct. 1996).

[14] SHEN, H.-W., AND JOHNSON, C. R. Sweeping simplices: A fast iso-surface extraction algorithm for unstructured grids. In *Visualization '95 Proceedings* (Oct. 1995), G. M. Nielson and D. Silver, Eds., pp. 143–150.

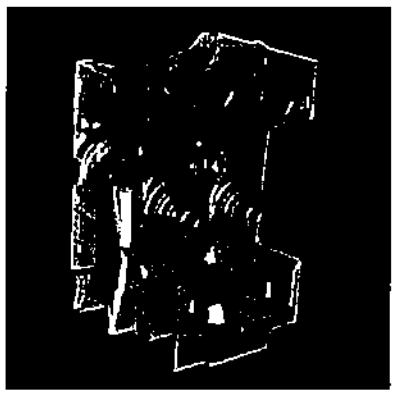
[15] SPERAY, D., AND KENNON, S. Volume probes: Interactive data exploration on arbitrary grids. In *Computer Graphics (San Diego Workshop on Volume Visualization)* (Nov. 1990), vol. 24, pp. 5–12.

[16] VAN KREVELD, M. Efficient methods for isoline extraction from a digital elevation model based on triangulated irregular networks. *To appear, International Journal of Geographical Information Systems* (1996). Also appeared as Technical Report UU-CS-1994-21, University of Utrecht, the Netherlands.

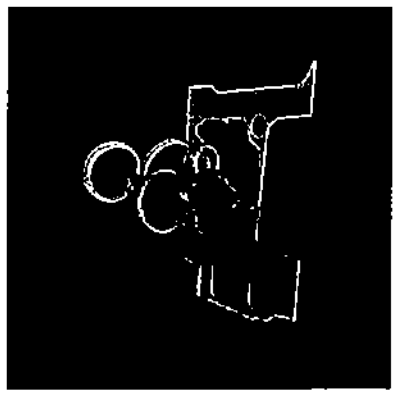
[17] WILHELMS, J., AND VAN GELDER, A. Topological considerations in isosurface generation: Extended abstract. *Computer Graphics (San Diego Workshop on Volume Visualization) 24*, 5 (Nov. 1990), 79–86.

[18] WILHELMS, J., AND VAN GELDER, A. Octrees for faster isosurface generation. *ACM Transactions on Graphics 11*, 3 (1992), 201–227.

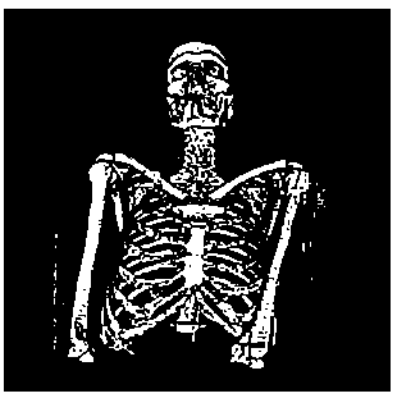
(a) MRbrain (isovalue 500.5)
973954 triangles



(b) CT Engine (isovalue 50.5)
584916 triangles



(c) CT Engine (isovalue 200.5)
142290 triangles

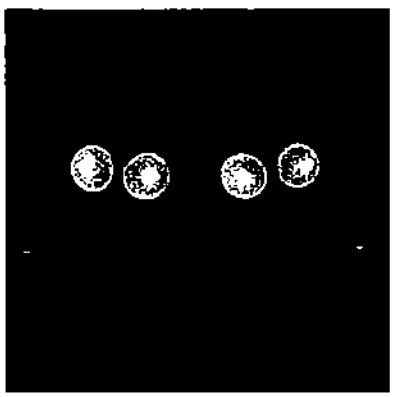


(d) CT Cadaver (isovalue 140.5)
830016 triangles

(e) CT Head (isovalue 200.5)
593456 triangles

(f) CT Head (isovalue -150.5)
474378 triangles

(h) SOD (isovalue 36.5)
189590 triangles



(i) HIPIP (isovalue 0.1)
1848 triangles

(j) EAS (isovalue -1.881)
60622 triangles