A New Seed-Set Finding Approach for Iso-Surface Extraction

CHUAN-KAI YANG AND CHIANG-HAN HUNG
Department of Information Management
National Taiwan University of Science and Technology
Taipei, 104 Taiwan

Iso-surface extraction is one of the most important approaches for volume rendering, and iso-contouring is one of the most effective methods for iso-surface extraction. Unlike most other methods having their search domain be the whole dataset, iso-contouring does its search only on a relatively small subset of the original dataset. This subset, called a seed-set, has the property that every iso-surface must intersect with it, and it could be built at the preprocessing time. When an iso-value is given at run time, an iso-contouring algorithm starts from the intersected cells in the seed-set, and gradually propagates to form the desired iso-surface(s). As a smaller seed-set could offer less cell searching time, most existing iso-contouring algorithms concentrate on how to identify a minimal seed-set. In this paper, we propose a new and efficient approach for the construction of a relatively small seed-set. This presented algorithm could reduce the size of a generated seed-set by up to one or two orders of magnitude, compared with other previously proposed linear-time algorithms.

Keywords: Iso-Surface Extraction, Iso-Contouring, Seed-Set, Min-Max Span Space, Volume Thinning

1 INTRODUCTION

Volume rendering has been a very important research field in recent years due to its wide applications in various areas, including medical diagnosis, numerical simulations, education and entertainment, etc. While there are numerous techniques of volume rendering, iso-surface extraction is one of the most popular approaches. In general, an iso-surface extraction process consists of two phases, *cell searching* and *triangle generation*. As the procedure of the second phase is nearly fixed, most of the current research therefore concentrates on reducing the time spent in the first phase, and among such, *iso-contouring* is one of the most effective methods. The idea of iso-contouring is to first identify a subset called *seed-set*, which has the property that every iso-surface must intersect with it. By assuming a continuous variation over the scalar fields defined on the cells of a dataset, an iso-contouring algorithm propagates from the intersected cells in the seed-set to form the desired iso-surface(s). Compared with other iso-surface

extraction methods, the benefit of iso-contouring is two-fold. First, the generated iso-surface could be readily converted into triangle strips, which could significantly reduce the traffic sent to the graphics card during the rendering stage, thus speeding up the rendering performance. Second, the size of the cell search domain is often dramatically decreased, and therefore the search time for finding the cells intersecting with the iso-surface is also minimized. Furthermore, many other techniques that help to reduce the cell search time could also be applied, such as *interval trees*, leading to even better overall performance. Compared with the original dataset, the derived seed-set size is often relatively smaller so that we could quickly locate where to start the iso-surface propagation. This defines the very goal of the preprocessing for an iso-contouring algorithm. Inspired by the *min-max span space representation* proposed by [15], we observed that a seed-set of a dataset could be constructed in a brand-new way, which will be explained in the ensuing sections. This observation leads to a fairly simple implementation but with high efficiency. In addition, our proposed method is independent of other optimization techniques, such as the one proposed by Bajaj et al. [2, 3, 4]; in other words, our method could be applied together with their approaches. For example, it can be shown that when combined with the existing *volume thinning* approach, the size of the resulting seed-set could be further reduced to be about 140 times smaller than the one by using the volume thinning approach alone.

The rest of the paper is organized as follows. Section 2 reviews the related work on iso-surface extraction, while section 3 details the technique of volume thinning, which serves as a comparison and test-bed for our proposed approach, and the concept of min-max span space, whose representation of a dataset inspires our new idea for seed-set construction. Section 4 presents our new algorithms, which could be viewed as new ways for minimizing a seed-set. Section 5 demonstrates the efficiency of our method when compared and/or combined with the volume thinning approach. Section 6 concludes the paper and envisions the potential future directions.

2 RELATED WORK

Many techniques for volume rendering have been developed, and these techniques can be classified into two big categories: *direct volume rendering*, such as raycasting [13,

14], and *indirect volume rendering*, such as iso-surface extraction. Lorenson's *marching cubes* method [12] pioneered the research on iso-surface extraction. There are two phases during the iso-surface extraction process, namely the *cell searching* and *triangle generation*. While the procedure for second phase is quite standard now, except for the *ambiguity* problem, the first phase still leaves room for further improvement.

There are essentially three schools of thoughts on reducing the cell searching time. The first type is to employ a *space-based* approach, such as the method proposed by Wilhelms et al. [18] to make use of octrees. The second type is to apply a *range-based* approach, such as the ones proposed by Gallagher [8], and Livnat et al. [15]. Shen et al. [16] later further improved the searching time complexity by using a uniform partition in the area of the 2D plane defined by $x \leq y$, the only area where all the cells of a dataset can fall into. This is also called the *min-max span space representation*, which will be detailed later. Yet another method, proposed by Cignoni et al. [6], demonstrated how to use the concept of *interval trees* to answer an iso-surface query, which is essentially a *stabbing query* first mentioned in the field of *computational geometry* [7], to achieve the optimal time complexity for searching. An *interval tree* is a hierarchical data structure built at the preprocessing time, so that an iso-surface query could be answered at run time with a *logarithmic time complexity*. The third type is to utilize a *surface-based* approach, which at its preprocessing stage identifies a subset of the original dataset, called a *seed-set*, and then at run time propagates to form the entire iso-surface(s) from the intersected cells within the seed-set. The way of identifying a seed-set from the dataset distinguishes the methods of this type. Itoh et al. [9] proposed to build a seed-set through an extremum *graph*, which is originally consisted of the *local maximum* and *local minimum* points. These extremum points are connected to form a graph so that at run time, as each iso-surface must intersect with such a graph, the intersected cells can be located and propagated to form the entire iso-surface(s). However, an iso-surface can be either closed or open, while the above approach is only suitable when the target iso-surface is closed. In order to cope with open iso-surfaces, boundary cells are sorted and included as well, but the inclusion of boundary cells may incur great overheads. To address this issue, they later proposed a *volume thinning* approach to form a *skeleton* from the original dataset, and this skeleton alone could serve as a seed-set [10], thus eliminating the need of sorted boundary lists. By observing some basic property of a seed-set, Bajaj et al. proposed an

algorithm [2], which initially treats the whole dataset as a seed-set then gradually reduces the redundant cells by a *sweeping* paradigm. Their algorithm first defines the range of a face (edge or vertex) connecting two cells to be the iso-value range within which if one cell intersects with the corresponding iso-surface, the other cell will also be enumerated through the same face (edge or vertex) during the surface propagation process. Then for a cell, one fundamental property is that, if the union of ranges of its faces (edges and vertices) contains its range, the cell can be removed. Kreveld et al. developed an approximation algorithm [11] by constructing a *contour tree* which contains the *local maximal*, *local minimal* and *saddle points*. And for the first time, it can be proved that their method can generate a seed-set that is at most twice the size of the optimal seed-set size. However, the required worst-case running time is $O(N^2)$ for 3D meshes. By making use of the idea of a contour tree, Bajaj et al. later proposed several methods that could trade the seed-set size for timing efficiency [3]. One of these methods is called *greedy climbing*, which tries to select the seed cell that allows us to "climb" or "descend" as much as possible in the contour tree. This algorithm, though faster than the original contour tree approach, still requires worst-case O(N log N) time complexity as it involves the operations of priority queues. To further speed up the seed-set construction process, another method, called *sweep filtering*, is also proposed. Without the use of priority queues, this approach proceeds by ensuring that the selected seeds always fall on the extrema of the contours in the current sweeping direction. Though a linear-time algorithm, the resulting seed-sets are often much larger, as will be shown in the performance results section. Our approach, on the other hand, may not be able to produce the seed-set as small as the one by the method of contour tree or greedy climbing, is a linear-time algorithm that could often generate seed-sets with a relatively small size, and at the same time being relatively easy to implement. Essentially it strikes a good balance between the timing constraint and the seed-set size.

Among the described related work, we will further detail the min-max span space and volume thinning approach in later sections as they serve as the foundations for our new approach.

3 BACKGROUNDS

3.1 Volume Thinning

Itoh et al.'s volume thinning approach [10] provides an efficient way to construct a seed-set. Its basic concept is to first identify the extremum points from a dataset. An extremum point is either a local maximum or local minimum, in other words, a maximum or minimum compared to all of its neighbors. Starting from the whole dataset, with those extremum points initially marked as non-removable, cells that will not affect local connectivity are removed. Here a cell denotes a cube which has eight scalar values defined on its eight corner points, respectively. The entire process proceeds as if the whole volume gets thinner and thinner, and eventually a skeleton is formed. See Figure 1 (modified from [10]) for a demonstration of a 2D thinning process.

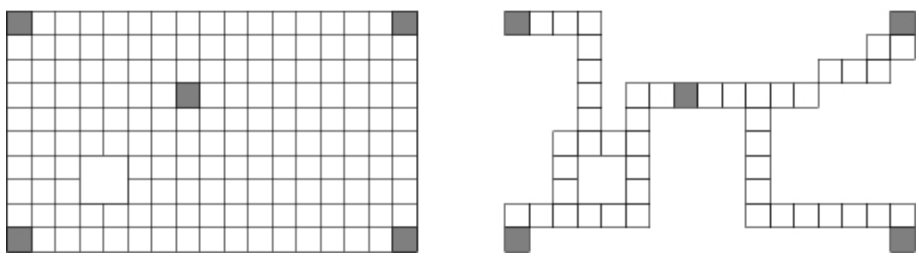

Figure 1. A 2D volume thinning process. On the left: the original dataset, where the extremum points are marked with dark gray. On the right: the resulting skeleton after a thinning process.

3.2 Min-Max Span Space

Livnat et al. proposed *Min-Max span space* to solve the cell searching problem in iso-surface extraction [15]. As our new approach is inspired by this representation of a dataset, it is necessary that we explain its basic concept before illustrating our new idea. A min-max span space, as shown in Figure 2, is a 2D representation of a volume dataset, where each cell (cube or tetrahedron) is denoted by a point. The x coordinate of a point represents the minimal scalar value defined on the corresponding cell, while y coordinate the maximal scalar value. As each cell's minimal scalar value cannot be greater than its maximal scalar value, all the points must fall within the half plane corresponding to the equation of $x \leq y$. Moreover, using this representation, and given an iso-value c, all the cells that intersect with the desired iso-surface must appear

within the half-open regions defined by $x \leq c$ and $y \geq c$, as shown by the light-grey region in Figure 2.

4 NEW APPROACH

In this section, we describe where our idea originates from, and what our approaches are. We have implemented two variants, and each of them will be detailed in the subsections.

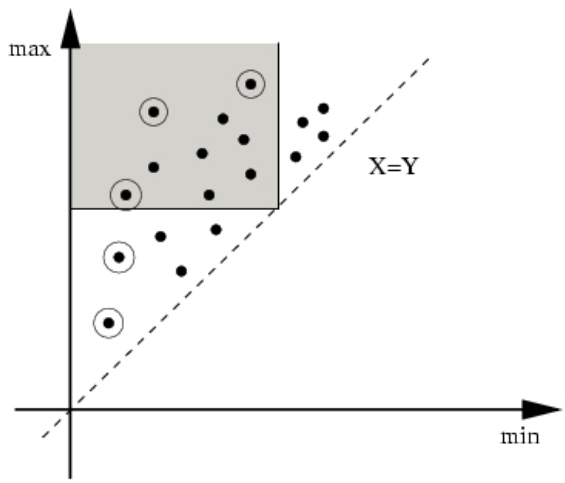

Figure 2. A min-max span space representation of a dataset, where each cell is represented as a black dot. The grey area corresponds to the region that could satisfy an iso-value query. All the cells falling into this area intersect with the desired iso-surface. All the dots who have no upper-left neighbors are marked with circles. These marked dots could be included in the final seed-set.

4.1 Upper-Left Envelope

Our new approach, though a surface-based algorithm, is in fact mainly inspired by the min-max span space representation of a dataset. Recall that a seed-set of a dataset should bear the property that every iso-surface intersects with it. As most surface-based approaches try to identify a seed-set from a dataset's original domain, what we are really curious about is how a seed-set behaves in the domain of the min-max span space. The first thing came to our mind is a line, which is parallel to the line of $x = y$. Apparently such a line satisfies the requirement: every iso-surface intersects with it. However, when

a dataset is represented by points distributed on the upper-left half plane of the first quadrant, it is not clear what line to choose and how a particular line corresponds to a seed-set. It did not take long before we realize that a good candidate for a seed-set does exist, and it is in fact in the forms of *envelope lines*. In terms of a min-max span space representation, we observe that all the cells, represented as points, which have no cells on their upper-left side, could be included in the seed-set. Figure 2 demonstrates such an observation. In this figure, those points marked in circles could be included in the seed-set. This claim can be proved by the following. Assuming *S* represents the set of all the points (cells) which do not have any other points on their upper-left side, then this *S* must intersect with every iso-surface. As long as an iso-surface passes through a dataset, it must intersect with this data by at least one cell, say cell *a*. If *a* cell belongs to *S*, then we are done; otherwise there must exist another cell, say cell *b*, which is on the upper and left-hand side of cell *a*. From Figure 2 we know that cell *b* must also intersect with this iso-surface. If cell *b* belongs to *S*, we are done; otherwise the procedure just described can be carried out recursively, and due to the fact that the cell number of a dataset is finite, we will eventually reach a cell which belongs to *S*, thus proving our claim. For convenience, we will call these cells in *S* to be on the *upper-left envelope*. This is in fact a special case of the so called *maxima finding* problem [1, 5], and it can be shown that an algorithm of worst-case time complexity of O(N log N) could be easily derived. However, we will not go into the details of this, because in the end we have found an even simpler and faster algorithm to fulfill our goal.

Although it seems that we have found a perfect seed-set this way, there are still two imperfections. First, as will be shown later, these cells are not optimal, i.e., it is still possible to further reduce the cell number of such a seed-set without hindering its capability to intersect with all the iso-surfaces. Second, such cells still do not form a complete seed-set. Let us consider a 2D counterexample given in Figure 3. In this figure, the corresponding intervals for cell *A*, *B* and *C* are (40, 50), (50, 60), and (30, 70), respectively. It is clear that only cell *C* is on the upper-left envelope while the other two are not. However, if we only retain cell *C* into the seed-set, then for the iso-value query whose iso-value falling in the interval of (40, 50), there is no way of propagating from cell *C* to cell *A*, as cell *B* does not intersect with any value of this range. On a deeper thought, what is really missing here is the consideration of connectivity. Put it more

concretely, cell *B* should be retained so that the iso-surface propagation can reach cell *A* through cell *B*.

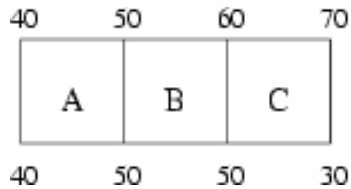

Figure 3. A 2D counterexample. The numbers are the corresponding scalar values defined on the grid points.

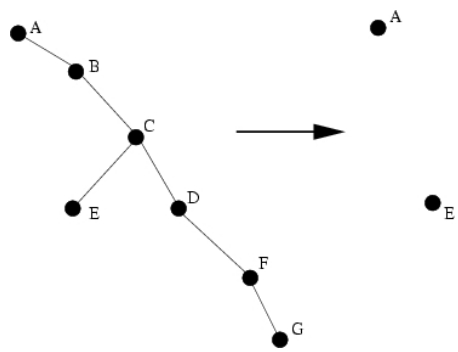

Figure 4. A cell reduction process. On the left are connected components of the original dataset, while on the right the reduced components of the dataset.

4.2 Variant 1

To take connectivity into account, in variant 1 we design our algorithm as the following. We start by conceptually constructing a graph in the min-max span space where two cells (or points in the min-max span space) are connected by an edge if they are connected through a face. From this graph we try to remove unnecessary cells, as shown in Figure 4. In this Figure, a given cell can be removed from the graph if it has an upper-left neighbor, because such a neighbor would have a smaller minimum and a larger maximum, therefore at run time, the cell can be *re-discovered* by the surface propagation process from this neighbor. Notice that once a cell is removed, the edges connected to it should be transitively adjusted, as shown in this Figure. However, from the implementation point of view, it is in fact *not necessary to construct the connected graph*,

instead, we could start with treating the whole dataset as a seed-set, and then gradually remove the unwanted cells one by one. Most importantly, *the algorithm requires only one pass of scan through all the cells then a seed-set can be constructed*. For each cell, we just need to check all of its six face-connected neighbors to see if it has an *upper-left* neighbor in the min-max span space representation, or equivalently, if it has a neighbor whose range contains this cell's range. If so, the cell can be removed from the seed-set; otherwise, it should be retained. The reason behind this is straightforward: if a cell has a face-connected neighbor which appears to its upper and left-hand side in the min-max span space's representation, it means that neighbor has a containing range of the current cell. This inclusiveness property guarantees that once that neighbor is preserved in the seed-set, the current cell could always be re-connected through the corresponding face. Notice the transitive property is implicitly preserved during this process, thus requiring no other bookkeepings or particular data structures. In other words, each cell could be checked individually without worrying its neighbors' existence. However, there is one exception. If cell *A* and cell *B* are adjacent with each other by a face, and if cell *A* and cell *B* have exactly the same range while all other neighbors of cell *A* and *B* do not have containing ranges, then our algorithm will remove cell *B* from cell *A*'s point of view, and remove cell *A* from cell *B*'s point of view, as shown in Figure 5. One simple and less precise approach to deal with this exception is to first assign a unique ID to each cell, and then when it comes to a cell's removal, only the cell with a larger ID value is removed. However, this simple approach may produce a less optimal result, as shown in Figure 5, where both cell 1 and cell 5 will be retained, while only one of them should be preserved. To correct this, first we leave all such cells intact, then on each such connected region, as shown in Figure 5, we apply the cell propagation process as if we are to find all the intersected cells with an iso-surface. During this process, we could identify the cell with the smallest ID, and thus only one such cell should be retained, while all others could be safely discarded.

There is one more optimization that we could perform to further reduce the size of a seed-set. Recall in the iso-surface propagation process, the intersected cells found in the seed-set are used to propagate and to locate all the intersected cells with the desired iso-surface(s). Usually this propagation is performed through face connectivity as shown in Figure 6(a). However, as an iso-surface could also pass through an edge, shown in Figure

6(b), or touch a vertex, shown in Figure 6(c), we could modify the surface propagation process accordingly. This modification also affects the seed-set construction as the definition of a *neighbor* for a cell gets changed. By taking the new definition into account, our algorithm requires little modification while most of it remains unchanged.

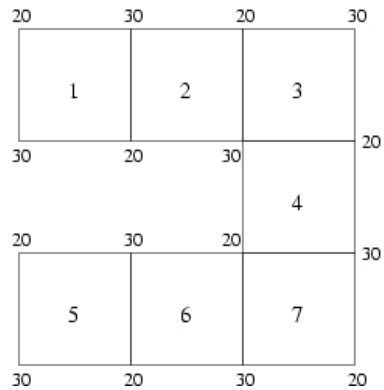

Figure 5. A 2D example where several cells with exactly the same range are connected together.

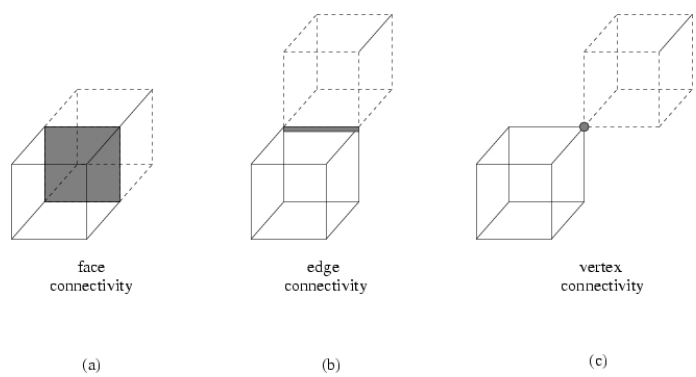

Figure 6. Different connectivity between cells.

Note that there is one more modification to be done during the surface propagation process so that our approach is feasible. Refer to Figure 7 for a 2D illustration. In this Figure, the curves represent the iso-surfaces corresponding to iso-value of 25. According to a normal surface traversal, such as the one used in Bajaj et al. [2], each cell only checks to see if any one of its faces intersects with the iso-value, surface propagation will then proceed along the direction of that face (edge or vertex) neighbor. Without modification, it is apparent that a surface propagation starting from cell *C* will not reach cell *A*, as the face between cell *A* and *B* does not intersect with the given iso-value. To

correct this, we just need to make little modification to the surface propagation process: *if the range of any face neighbors of the current cell intersects with the given iso-value, surface propagation should proceed along the direction of that neighbor*.

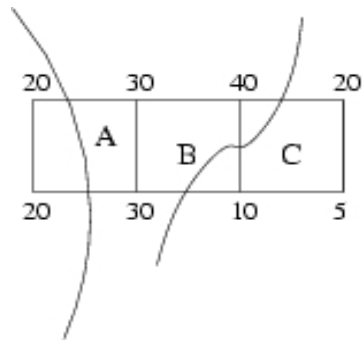

Figure 7. A 2D example of iso-surface(s). Here the given iso-value is 25.

There is still another optimization that we could perform and sometimes *it can significantly reduce the skeleton size produced by the original volume thinning algorithm*. The trick is that when the eight scalar values of a *non-isolated* cell are all equal to one constant value, this cell can be removed, as this constant value must also appear on one of its neighbors. At run time, when this particular iso-value is requested, one of its neighbors would be included, either because that neighbor belongs to the seed-set, or is reached by the propagation from the seed-set, and thus this cell will eventually be included as well. Therefore this cell does not need to be present in the seed-set. To be combined with the previous checking process, in our implementation, we further group all the adjacent cells with a constant value into a *macro cell*, which could bear an arbitrary shape. When a macro cell is determined to be removed, all the cells within a macro cell are removed simultaneously; otherwise, only one cell within the macro cell is retained, as it suffices to do so. Because of this *grouping*, from this point on, unless we mention explicitly, all the cells are implicitly macro cells.

4.3 Variant 2

Recall the method proposed by Bajaj et al. [2], where a cell could be eliminated if the union of ranges of its faces (edges or vertices) contains its range. By making use of this property, Bajaj et al. applied a sweeping algorithm to obtain an approximate seed-set.

Similar to the modification we just made during the surface propagation process, we could modify Bajaj et al.'s cell removal property to the following. If the union of ranges of a cell's neighbors contains its range, then this cell can be removed. There are two issues regarding the implementation of variant 2. First, unlike the case in variant 1 where each cell could perform its check individually, the check for the union of neighbor's ranges may not be performed independently, i.e., we would need to worry about the existence of a cell's neighbors in variant 2. Second, as shown in Bajaj et al. [2], the order in which the checks are performed could affect the resulting seed-set size.

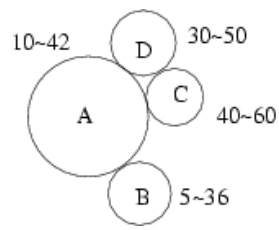

Figure 8. An example showing the cyclicity of the relationship between a cell and its neighbors.

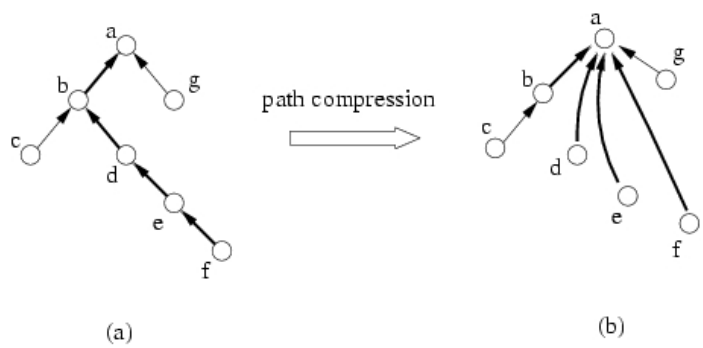

Figure 9. The path compression process, where (a) is the initial configuration and (b) is the configuration after performing the search on cell f.

To address the first issue, Figure 8 demonstrates an example, where the range union of cell *A* and cell *C* could be used to cover cell *D*, which therefore could be removed. However, the range union of cell *B*, cell *C*, and cell *D* could also be used to cover cell *A*, and this indicates that cell *A* is also removable. Nevertheless, it is clear that in this case only one of cell *A* and cell *D* could be removed due to this *cyclicity*.

For the convenience of discussion, if a cell, say cell *A*, whose range could be covered by the union of the ranges of two of its neighbors, say cell *B* and cell *C*, then we

call cell *B* and cell *C* the *parents* of cell *A*, while cell *A* a *child* of cell *B* and cell *C*. Notice that this relationship is *transitive*, that is, cell *B* and cell *C* may have their own parent(s) as well. For this, we call cell *D* an *ancestor* of cell A if there is a transitive parent-child relationships between them, while cell *A* a *descendant* of cell *D*. In particular, we call an ancestor cell which has no parents a *terminal ancestor*. To avoid cyclicity, a naive implementation would be to associate each child cell with a link list of all of its parents, therefore to check for the cyclicity of a cell, say cell *E*, we search recursively to see if cell *E* appears in any of the link lists of its parent(s) and ancestor(s). If so, cell *E* cannot be removed; otherwise cell *E* is marked as *removable* and all its neighboring cells that are used for this removal are put in cell *E*'s link list accordingly. Empirical results show that it is possible to have a long transitive relationship between two cells therefore the checking for cyclicity could be time-consuming.

To improve on the checking efficiency, we borrow the idea of *path compression* from the operations of *disjoint sets* [17]. The basic concept can be best illustrated by the example shown in Figure 9, where we compare the configuration before and after performing the search on cell f. Note here each arrow points from a child to its parent(s), therefore in this Figure, cell *b* could be used to cover both cell *c* and cell *d*, while cell *d* alone could be used to cover cell *e*, as explained in the variant 1 algorithm. After path compression, essentially the search path from a descendant to its ancestor is dramatically shortened, thus speeding up the searching performance significantly.

Figure 10 demonstrates how we combine the variant 1 and variant 2 algorithms together. Figure 10(a) is the result after performing the variant 1 algorithm, where each region (marked with different colors) has only one terminal ancestor, denoted by two concentric circles. Figure 10(b) is the result after performing the path compression within each region, while Figure 10(c) shows the result after compressing the search path on the terminal ancestor cell in region *C*.

To deal with the second issue, i.e., the order of checks to be performed, and at the same time without resorting to a nonlinear-time algorithm as was done by Kreveld et al. in [11], we perform the union checks according to the number of neighboring cells used for removing a given cell. This is simply due to the fact that a cell with a larger such number may suggest a larger dependency on others, therefore its check is deferred for better reduction efficiency. Although it seems sorting is un-avoidable during the process,

in practice, the number of neighboring cells are often around a dozen or so, therefore *counting sort* could be used, thus making the total complexity still *linear* in this phase.

So far, the only thing left unexplained is, for a given cell, say cell *B*, how to find the least number of neighbors such that their range union could cover the range of cell *B*? For brevity, we will refer to this subset finding process as a finding *minimal cover* process, and it can be shown that a minimal cover could be found by a recursive process. Figure 11 demonstrates such a process, where in the representation of a min-max span space, all the neighbors of cell *B* are represented by points in the four quadrants with cell *B* being the center. If there is a cell in *B*'s second quadrant, it means that cell *B* has an upper-left neighbor, so cell *B* can be removed, and the corresponding process is already described in the variant 1 algorithm. Assuming there exist no cells in its second quadrant, and then the next step is to find the union of neighbors, and because the cells in the fourth quadrant are all dominated by cell *B*, we only need to check the cells in the first and the third quadrants. We first locate the *leftmost cell* in the first quadrant, say cell *A*, and the topmost cell in the third quadrant, say cell *C*. Note here according the definition of min-max span space, cell *A*'s x-coordinate and y-coordinate must be larger than those of cell *B*, therefore by finding cell *A* this way, we could maximize the chance of covering cell B, similar reason holds for finding cell *C*, as can be seen in Figure 11(a). If cell *C*'s y-coordinate is larger than cell *A*'s x-coordinate, the check for union would terminate immediately, as the union of cell *A* and *C* are sufficient to cover cell *B*. On the other hand, if cell *A*'s x-coordinate is larger than cell *C*'s y-coordinate, and since cell *A* is the one with the minimal x-coordinate, this means that there exists a gap between the maximum of the cell *C* and the minimum of cell *A*, therefore it is hopeless that the union of cell *B*'s neighbors could cover cell *B*. However, in the case where cell *A*'s x-coordinate is less than cell *B*'s y-coordinate and cell *C*'s y-coordinate larger than cell *B*'s x-coordinate, the union of the range of cell *A* and *C* could reduce the *uncovered* range of cell *B*, therefore the remaining uncovered range could be viewed a new *virtual cell* whose range could be recursively checked by the union of those cells in *B*'s fourth quadrant, such as cell *D* in Figure 11(b). Note that this recursive process for each cell could only take constant time as the number of neighbors for a cell is bounded by 26 (the effective number is even much smaller), equivalently, for each cell the number of being a neighbor of others is also fixed, therefore the overall linear-time complexity in this phase is guaranteed.

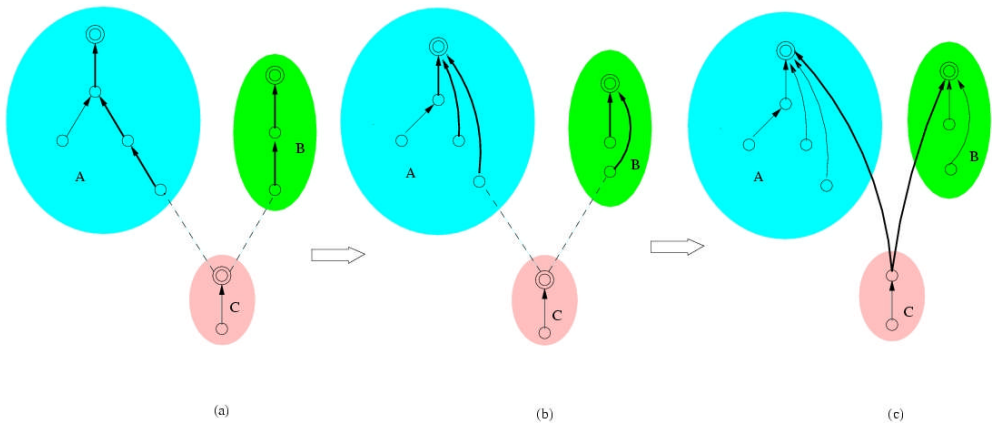

Figure 10. The path compression process in our algorithm.

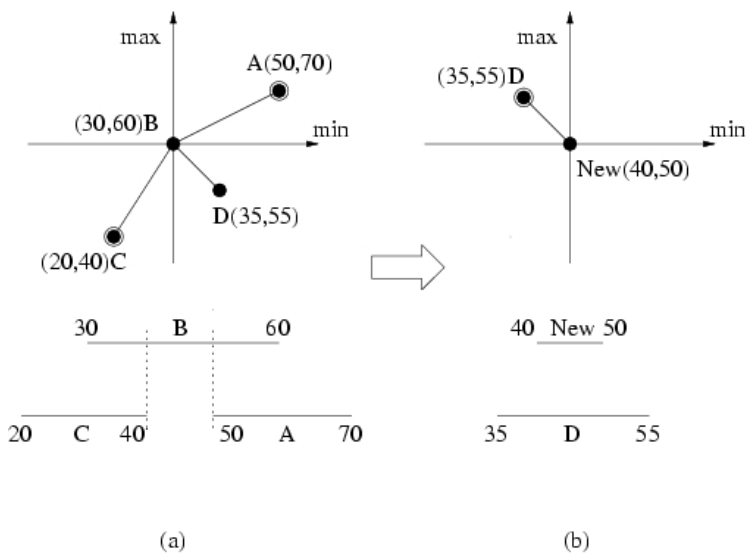

Figure 11. The procedure for finding a minimal cover.

4.4 Combined with Volume Thinning

In addition to proposing a new approach, another goal of this paper is to combine this approach with existing approaches to seek for an even smaller seed-set. According to Itoh et al. [10], their volume thinning approach performs much better than their previous

extrema graph approach [9], due to the elimination of boundary cells. Therefore, we re-implement their volume thinning algorithm so that we could compare and combine their approach with ours.

Noticing that as volume thinning and our approach bear different perspectives towards seed-set reduction, we could combine these two approaches and see how the merged algorithm performs. To do that, we first generate a skeleton by using the volume thinning approach, then apply the procedure that we mentioned in the last sub-section, i.e., each cell in the skeleton checks to see if it could use the variant 1 or variant 2 algorithm to remove itself.

As a last remark, after the seed-set is constructed, we could build an *interval tree* from the cells in the seed-set to further speed up the seed cells searching at run time, just as proposed by [2]. Because a seed-set is usually relatively smaller compared with the original dataset, the cell searching for finding the intersected cells in the seed-set could become extremely fast. Note that although these intersected cells still need to be propagated to find all the cells intersected with a desired iso-surface, nevertheless, the asymptotic time complexity is only roughly proportional to the number of cells intersected with the iso-surface. That is, we only spent minimal effort on those non-intersected cells, and at the same time could enumerate cells in an order which facilitates the generation of triangle strips to enhance the rendering throughput.

5 PERFORMANCE RESULTS

We have implemented our system on a Pentium 4 2.8GHz machine with 1GByte memory, running the Windows XP operating system. We have collected and tested totally 10 volume datasets. Table 1 lists the characteristics of these 10 datasets. We also list the number of extremum points for reference. # of Skeleton Cells are the results by using Itoh et al.'s volume thinning algorithm, which we have re-implemented for the purpose of comparison. For convenience, we use *Whole_n* to denote the results of running the *variant n* algorithm exclusively on the whole dataset, and similarly *Skeleton_n* the results of running the *variant n* algorithm exclusively on the resulting skeletons produced by the volume thinning approach. In addition, *Whole_1+2* and *Skeleton_1+2* denote the cases where we run both the variant 1 and 2 algorithms on the

whole datasets and on the resulting skeletons with the volume thinning approach, respectively.

Table 1. Characteristics of input datasets used in this performance study.

Data set	Dimension	# of Extrem. Pts.	# of Skeleton Cells
MR Brain	256 × 256 × 109	1489073	2160246
CT Head	256 × 256 × 113	764819	1410784
CT Engine	256 × 256 × 110	220783	791428
SOD	97 × 97 × 116	2758	41902
HIPIP	64 × 64 × 64	1465	10520
Hydrogenatom	128 × 128 × 128	44	1372
Aneurism	128 × 128 × 128	6034	33134
Bonsai	128 ×128 × 128	43479	155850
Skull	128 ×128× 128	276530	495554
Foot	128 ×128 × 128	127554	234405

Table 2. Comparison of the number of seed cells between the original skeleton and the new skeleton after removing the constant cells.

Data set	# of Skeleton Cells	# of Skeleton Cells After Removing constant cells
MR Brain	2160246	2109589
CT Head	1410784	1410592
CT Engine	791428	776573
SOD	41902	27611
HIPIP	10520	10518
Hydrogenatom	1732	943
Aneurism	33134	12726
Bonsai	155850	115281
Skull	495554	491410
Foot	234405	200949

Table 2 shows the impact of removing those cells with constant values. As shown by the *Aneurism* dataset in this Table, the number of skeleton cells could become roughly three times smaller. This phenomenon is not completely occidental, as shown by Table 1, since there are only 256 possible values to be distributed to 127×127×127 cells, there may still exist some homogeneous regions which were not removed by the volume thinning process.

Table 3 demonstrates how connectivity affects the seed-set size, by using the variant 1 algorithm running on the *whole dataset*. In this paper, as each cell is a cube,

therefore it has 6 faces, 12 edges and 8 vertices. As shown by Figure 6, if we allow connectivity to be extended from face-connectivity, denoted by F, to face-connectivity plus edge-connectivity, denoted by $F + E$, or together with vertex-connectivity, denoted by $F + E + V$, for both the seed-set reduction and surface propagation, then the seed-set size could be reduced up to four times smaller, as demonstrated by the *Hydrogenatom* dataset in this table. As this table suggests, from this point on all the reported numbers are based on the $F + E + V$ connectivity. Note that for the numbers reported here, we use the *non-macro cell* version, and by comparing with Table 5, where macro cells are used, the further reduction of seed-set size is evident.

Table 3. Comparison of the number of seed cells using the variant 1 algorithm running on the whole dataset, under different connectivity implementations for all the datasets.

Data set	F	F+E	F+E+V
MR Brain	656908	335590	283335
CT Head	832534	428352	356679
CT Engine	838520	487104	419687
SOD	154338	80523	62619
HIPIP	92582	53486	41262
Hydrogenatom	112412	41122	27891
Aneurism	5411	3589	3241
Bonsai	63234	32626	27931
Skull	243459	119357	98049
Foot	73504	39055	32878

Table 4. The resulting number of seed cells using the variant 1 algorithm, under different traversal orders, for the dataset of Hydrogenatom.

Direction	XYZ	XZY	YXZ	YZX	ZXY	ZYX
Seed-set Size	843	239	612	587	282	648

As expected, with different traversal orders, the seed-set size may vary as well. Although we have mentioned in Section 4 that the variant 1 algorithm requires only one pass through the dataset, it does not always produce the same seed-set size when the traversal order is changed. Table 4 demonstrates that, for the dataset of Hydrogenatom, different traversal directions may cause a large variation on the resulting seed-set size. For example, the number corresponds to the direction of XYZ is about four times as big as the one corresponding to the direction of XZY. Note here the direction of XYZ means that, when we traverse the cells, we visit plane by plane of cells with their Z-coordinates

being in an ascending order, while within each plane we visit cells row by row with their Y-coordinates being in an ascending order. Other traversal directions can be derived similarly.

The reason for such a difference could be illustrated using Figure 12. In Figure 12(a), cell *A* could be eliminated by using either cell *B* or cell *D*. In Figure 12(b), when the traversal order is from top to bottom, we may choose cell *B*; on the other hand, cell *D* may be picked if we use the bottom-to-top order as in Figure 12(c). We have tested all of our datasets for multiple directions and so far only the Hydrogenatom dataset presents such a large variation in terms of the resulting seed-set size.

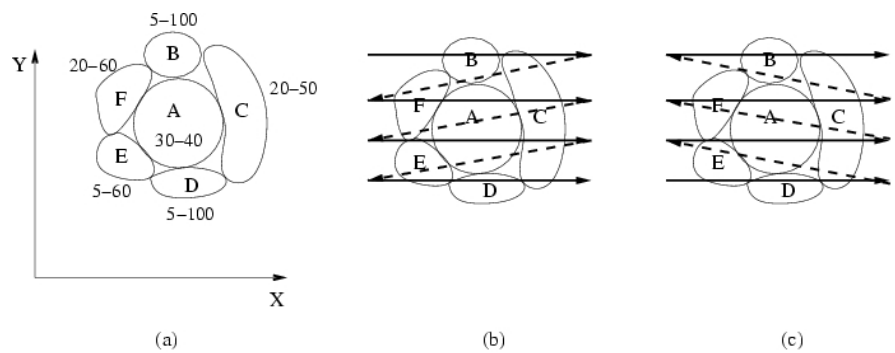

Figure 12. Applying different traversal orders when using the variant 1 algorithm.

Table 5 juxtaposes the resulting seed-set sizes by using different algorithms, where *Itoh*, *Bajaj1*, *Climb* and *Sweep* represent the results by the work in [10] (volume thinning), [2] (fast iso-contouring), *greedy climbing* and *sweep filtering* in [3] (different version of fast iso-contouring), respectively. Due to the limit of time, we did not get to re-implement Bajaj et al.'s method in [2, 3], so we extracted the numbers from their papers and downloaded the only dataset available, i.e., the *HIPIP* dataset, for comparison. There are several relationships to observe from this table. First, the numbers in *Whole_1* and *Skeleton_1* are all less than the numbers in *Whole_2* and *Skeleton_2*. This is simply because in the variant 2 algorithm we look for more chances to delete a cell, and the test in the variant 1 algorithm is just a special case of that in the variant 2 algorithm. Second, the numbers in *Skeleton_1* and *Skeleton_2* are smaller than those in *Itoh* as the former two are built on top of the latter to seek further possibility of reduction. The most

intriguing part is the comparison between *Whole_1+2* and *Skeleton_1+2*, where *Whole_1+2* wins in all the cases except for the *Hydrogenatom* dataset. And the same time, this dataset is also where *Whole_1* lost most when compared with *Itoh*. After a detailed analysis we found that because the *Hydrogenatom* dataset presents thin layers of equal values which may be pricked into fragmented parts by the *Whole_1+2* approach, therefore it can be handled more properly with the volume thinning approach.

Table 5. Comparison of the number of seed cells among Itoh et al.'s volume thinning approach, Bajaj et al.'s three versions of iso-contouring, and several variants of our approach running on the whole datasets or on the skeletons produced by Itoh et al.'s approach, under the connectivity of face, edge and vertex.

Data set	Itoh	W_1	W_2	W_1+2	S_1	S_2	S_1+2	Bajaj1	Climb	Sweep
MR Brain	2160246	254713	183072	32170	425159	307039	210531	639891	N/A	N/A
CT Head	1410784	257633	125701	10032	334120	281006	198274	423366	N/A	N/A
CT Engine	791428	311839	88434	31874	284245	220231	163469	180048	N/A	N/A
SOD	41902	52438	7395	5833	17807	11294	9734	13004	N/A	N/A
HIPIP	10520	35852	3642	869	5671	3652	2237	4616	529	6397
Hydrogenatom	1732	16079	3103	843	763	400	357	N/A	N/A	N/A
Aneurism	33134	1693	1693	1557	5973	4061	3019	N/A	N/A	N/A
Bonsai	155850	17389	15433	6952	41039	29942	21732	N/A	N/A	N/A
Skull	495554	70456	39418	19346	120530	93304	59646	N/A	N/A	N/A
Foot	234405	23322	21974	10581	50267	36708	23848	N/A	N/A	N/A

For the comparison between *Whole_1+2* and *Bajaj1*, the reason why *Whole_1+2* wins all the time is also self-evident, as explained previously that the tests performed in *Bajaj1* can be deemed as a special case of what is performed in *Whole_1+2*. Overall, *Whole_1+2* can reduce the dataset size to be up to 140 times smaller (as shown by the *CT Head* dataset) when compared with *Itoh*, and up to 40 times smaller (as shown by the *CT Head* dataset) when compared with *Bajaj1*. The only available numbers from Bajaj et al. in [3] are for the *HIPIP* dataset. Although their *climbing* algorithm could produce a seed-set about 60% smaller than ours, it requires a higher time complexity, i.e., $\Theta(N\log N)$; on the other hand, the much faster linear-time *sweeping* algorithm generates a seed-set much larger than ours. Overall speaking, our linear-time algorithm still looks very appealing. Note here for saving space, we use *W_n*, *S_n*, *W_1+2*, and *S_1+2* to denote *Whole_n*, *Skeleton_n*, *Whole_1+2*, and *Skeleton_1+2*, respectively, in Table 5.

As mentioned previously, in the variant 2 algorithm we need to perform the cyclicity check, where transitive dependency may be involved. Therefore to perform the full cyclicity check the incurred overheads may be indefinite, thus defeating the purpose

as a linear-time algorithm, although in practice the maximal depth required for cyclicity check is well within reach. Table 6 demonstrates the effectiveness of using *path compression* instead of the naive *link list* implementation. As can be seen from the table, the performance improvement can be up to twenty times, as shown by the Aneurism dataset.

Table 6. Timing comparison in the variant 2 algorithm by using different approaches for cyclicity check across all datasets. Numbers are in seconds.

Data set	Linked Lists	Path Compression
MR Brain	120.691	40.516
CT Head	138.249	51.391
CT Engine	151.725	46.453
SOD	8.564	2.672
HIPIP	19.703	3.496
Hydrogenatom	18.282	12.843
Aneurism	4.327	0.192
Bonsai	5.419	1.178
Skull	12.743	3.610
Foot	4.571	0.435

Table 7 presents the timing breakdown of using the *Whole_1+2* approach, so far the best implementation of ours. Here *Init(ial) Proc(essing)* time includes program initializations, removing cells with constant values, etc., while *W_1*, *W_2* and *Total* represent the timing for performing the variant 1, variant 2 algorithms, and the total execution time, respectively. The main proportion of time spent on *W_2* is for checking the cyclicity, therefore is *data dependent*. For example, in the Hydrogenatom dataset, because the *ancestor/descendant* relationships are "deep" and common, most of the time is attributed to the cyclicity check. Note here we do not report the triangle interpolation time, as the focus of this paper is on the seed-set generation. We also do not include the skeleton generation time by the volume thinning approach as it is not our contribution. In fact, our code is still far from being optimized. Nevertheless, these results show that with a moderate class of PCs, all the seed-sets could be generated with a reasonable speed. And most importantly of all, these constructed seed-sets could be stored or even used for building an *interval tree* to quickly answer repeatedly iso-value queries.

To prove correctness, we have also verified the resulting seed-sets produced by all variants of our algorithms. We have devised a way to check if a seed-set is indeed a seed-set by testing all possible iso-values. For datasets with only integer scalar values, we just

need to test each integer within the scalar value range. For datasets with floating point scalar values, we first fnd the *union of all floating point scalar values defined on the grid points of the original data*, then exhaustively perform iso-value query with values coming only from the set of union. It can be shown that by testing such values, we could enumerate all possible cases of how all the cells of a dataset intersect with all possible iso-surfaces.

Table 7. Timing breakdown for the seed-set construction process across all datasets. Numbers are in seconds

Data set	Init. Proc.	W_1	W_2	Total
MR Brain	5.54	12.38	40.52	58.25
CT Head	5.83	17.91	51.39	75.13
CT Engine	7.34	11.08	46.45	64.88
SOD	1.02	1.88	2.67	5.56
HIPIP	3.52	2.72	3.50	9.73
Hydrogenatom	2.31	2.19	12.84	17.34
Aneurism	1.55	2.00	0.19	3.74
Bonsai	1.43	1.31	1.18	3.92
Skull	1.70	2.94	3.61	8.25
Foot	2.602	1.575	0.435	4.072

6 CONCLUSION AND FUTURE WORK

We have proposed and implemented a new approach to identify a seed-set from a volume dataset. This approach, though very simple, takes just linear time of preprocessing to construct a relatively small seed-set. Most importantly, due to its simplicity, it could also be combined with other seed-set finding approaches. In particular, our approach could be applied together with the volume thinning approach, to yield an even better result than applying volume thinning alone. Overall our algorithm could reduce the seed-set size to be up to 140 times smaller when compared with the original volume thinning approach. There are two directions that we plan to pursue in the future. First, we will generalize our work to handle tetrahedral volume datasets as well. Although tetrahedral volume datasets present more complex topology, it is in fact simpler than the case of regular volume datasets, in terms of face, edge and vertex connectivity. Second, just like the work done by Kreveld et al. [11], we will work on deriving a *linear-time* approximation algorithm that could find a seed-set for a volume

dataset with a *provably small size*.

REFERENCES

1. J. L. Bentley, K. L. Clarkson, and D. H. Levine, “Fast Linear Expected-Time Algorithms for Computing Maxima and Convex Hulls,” Algorithmica, 9:168–183, 1993.
2. C. L. Bajaj, V. Pascucci, and D. R. Schikore, “Fast Isocontouring for Improved Interactivity,” In Proceedings on 1996 Symposium on Volume Visualization, pages 39–46, October 1996.
3. C. L. Bajaj, V. pascucci, and D. R. Schikore, “Fast Isocontouring for Structured and Unstructured Meshes in Any Dimension,” In IEEE Visualization ’97 Late Breaking Hot Topics, 1997.
4. C. L. Bajaj, V. pascucci, and D. R. Schikore, “Seed Sets and Search Structures for Accelerated Isocontouring,” Technical Report 97-034, Department of Computer Science, Purdue University, 1997.
5. W. Chen, H. Hwang, and T. Tsai, “Efficient Maximal-Finding Algorithms for Random Planar Samples,” Discrete Mathematics and Theoretical Computer Science, 6:107–122, 2003.
6. P. Cignoni, P. Marino, C. Montani, E. Puppo, and R. Scopigno, “Speeding Up Isosurface Extraction Using Interval Trees,” IEEE Transactions on Visualization and Computer Graphics, 3(2):158–170, April 1997.
7. H. Edelsbrunner, “Dynamic Data Structures for Orthogonal Intersection Queries,” Technical Report Report F59, Inst. Informationsverarb.,Tech. University Graz, 1980.
8. R. S. Gallagher, “Span Filtering: An Optimization Scheme for Volume Visualization of Large Finite Element Models,” In IEEE Visualization’91, pages 68–75, October 1991.
9. T. Itoh and K. Koyamada, “Automatic Isosurface Propagation Using an Extrema Graph And Sorted Boundary Cell Lists,” IEEE Transactions on Visualization and Computer Graphics, 1(4):319–327, December 1995.
10. T. Itoh, Y. Yamaguchi, and K. Koyamada, “Volume Thinning for Automatic Isosurface Propagation,” In IEEE Visualization ’96, pages 303–310, October 1996.

11. M. J. Kreveld, R. Oostrum, C. J. Bajaj, V. Pascucci, and D. Schikore, "Contour Trees and Small Seed Sets for Isosurface Traversal," In Symposium on Computational Geometry, pages 212–220, 1997.
12. W. E. Lorensen and H. E. Cline, "Marching Cube: A High Resolution 3D Surface Construction Algorithm," Computer Graphics, 21(4):163–169, July 1987.
13. M. Levoy, "Display of Surfaces from Volume Data," IEEE Computer Graphics and Applications, 8(3):29–37, March 1988.
14. M. Levoy, "Efficient Ray Tracing of Volume Data," ACM Transactions on Graphics, 9(3):245–261, 1990.
15. Y. Livnat, H. Shen, and C. R. Johnson, "A Near Optimal Isosurface Extraction Algorithm Using the Span Space," IEEE Transactions on Visualization and Computer Graphics, 2(1):73–84, March 1996.
16. H. Shen, C. D. Hansen, Y. Livnat, and C. R. Johnson, "Isosurfacing in Span Space with Utmost Efficiency," In IEEE Visualization '96, pages 287–294, October 1996.
17. R. E. Tarjan, "Efficiency of a Good But not Linear Set Union Algorithm," Journal of the ACM, 22(2):215–225, 1975.
18. J. Wilhelms and A. V. Gelder, "Octrees for Faster Isosurface Generation," ACM Transactions on Graphics, 11(3):201–227, July 1992.