A Simple and Novel Seed-Set Finding Approach for Iso-Surface Extraction

Chiang-Han Hung and Chuan-kai Yang

National Taiwan University of Science and Technology

Abstract

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 to be the whole dataset, iso-contouring does its search only on a relatively small subset of the original data-set. 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 the run time, iso-contouring algorithm starts from the intersected cells in the seed-set, and gradually propagates to form the whole iso-surface. As smaller seed-sets offer less cell searching time, most existing iso-contouring algorithms concentrates on how to identify an optimal seed-set. In this paper, we propose a new and linear-time approach for seed-set construction. This presented algorithm could reduce the size of the generated seed-sets by up to one or two orders of magnitude, compared with other previously proposed fast (linear time) algorithms.

1. Introduction

Volume rendering has been an important research topic in recent years due to its wide applications in various areas, including medical diagnosis, numerical simulations, production of education or entertainment, and so on. While there are numerous techniques of volume rendering, iso-surface extraction is one of the most popular approaches. In general, Iso-surface extraction consists of two phases, cell searching and triangle generation. As the procedure of the second phase is nearly fixed, most of the current research concentrates on reducing the time spent during 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 at least one cell from the seed-set. Assuming continuous variation over the scalar fields defined on the cells, iso-contouring algorithm propagates from the intersected cells in the seed-set to form the desired iso-surface. 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 significantly reduces the traffic sent to the graphics card during the rendering stage, thus speeding up the performance. Second, the cell search domain is often dra-

matically decreased, and therefore the search time for finding the cells on the iso-surface is also minimized. Furthermore, many other techniques that help 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 seed-set size should be relatively smaller so that we could quickly locate where to start the iso-surface propagation. This defines the very goal during the preprocessing time of an iso-contouring algorithm. Inspired by the *min-max span space* proposed by [LSJ96], 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 with high efficiency. In addition, our proposed method is independent of other optimization techniques, such as the ideas proposed by Bajaj et al. [BPS96, BpS97b, BpS97a], in the sense that it could be applied together with other approaches. For example, it can be shown that when combined with the existing volume thinning approach, the size of the resulting seed-set from the original volume thinning approach could be further reduced to be 140 times smaller than the one using volume thinning approach alone.

The rest of the paper is organized as follows. Section 2 reviews the related work on iso-surface extraction, while sec-

tion 3 details the technique of volume thinning, which serves as a comparison and test-bed for our proposed approaches, and the concept of min-max span space, whose representation of a dataset inspires our new idea of 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

Volume rendering techniques can be classified into two big categories: direct volume rendering, such as raycasting [Lev88, Lev90], and indirect rendering, such as iso-surface extraction. Lorenson's marching cubes [LC87] pioneered the research on iso-surface extraction. Unlike direct rendering to generate images directly from the datasets, indirect rendering outputs polygonal meshes which are sent to the graphics engines for traditional rendering. In general, given an iso-value, the way iso-extraction proceeds is to first identify the cells intersected with the desired iso-surface, then extract the desired iso-surface cell by cell through outputting the approximated polygons. Therefore, there are two phases during the iso-surface extraction process, namely the cell searching and polygon (triangle) generation. While the second phase is quite standard now, except for the ambiguity problem, the first phase still leaves room for improvement.

There are essentially three schools of thoughts trying to reduce the cell searching time. The first type is called the space-based approach. Wilhelms et al. [WG92] proposed an octree decomposition method of this type for regular volume datasets. Each node in the octree records the *min* and *max* within it. A given iso-value is checked against the coarsest level in the octree and recursively sent to finer levels if necessary. However the real efficiency, usually defined by how much percentage of cells get touched , is very data-dependent and this method cannot be readily applied to irregular grids. The second type is called the range-based approach. Gallagher [Gal91] proposed a method of this type, which divided the range of data values into sub-ranges, cal led *buckets*. For each cell, we identify its starting bucket as well as its *span*, or the number of buckets its range intersects with. Cells with the same span are grouped together while within each group sub-groups are formed according to their starting cell. Span numbers greater than a threshold can be grouped together to save storage. Given a query, all the span groups are traversed and depends on the span that group represents, a number of more buckets will be traversed accordingly. Another very different method of this type, proposed by Livnat et al. [LSJ96], represents each cell by a point in a 2D plane with it y-coordinate to be the max value of a cell and the x-coordinate to be the min value of a cell. Each result to an iso-value query corresponds to a square

region with its lower right corner touching the line defined by the equation of $x = y$. A *kd-tree* is used to build a hierarchy based on those points. At run time to answer a query the kd-tree is traversed to find the right and lower boundary and those points falling within this region are reported. Shen et al. [SHLJ96] later improved the searching time complexity further by using a uniform partition in the area of the 2D plane defined by $x < y$, the only area where all the cells of a data set can fall into. Yet another method, proposed by Cignoni et al. [CMM⁺97], demonstrated how to use the concept of *interval tree* to answer an iso-surface query, which is essentially a *stabbing query* in the field of *computational geometry* [Ede80], to achieve the optimal time complexity for searching. It built a hierarchical data structure so that an iso-surface query could be answered at run time by a logarithmic time complexity. The third type is called the surface-based approach, which at its preprocessing stage identifies a subset of the original dataset, called seed-set, and then at run time propagates to form the entire iso-surface from the intersected cells within the seed-set. The way of identifying a seed-set from the dataset distinguishes every method of this type. Itoh et al. proposed to build an a seed-set through an extrema graph, which is originally consisted of the local maximum and 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. However, as an iso-surface can be either closed or open, the above approach is suitable only the iso-surface is closed. In order to cope with the open iso-surface, boundary cells are sorted and included as well. [IK95]. The inclusion of boundary cells incurs great overhead. To address this issue, later they proposed a volume thinning approach to form a skeleton from the original dataset, and this skeleton serves as a seed-set [IYK96], thus eliminating the need of sorted boundary lists. By observing some basic property of a seed-set, Bajaj et al. begins with the whole dataset as a seed-set and gradually reduce the redundant cells with a sweeping paradigm [BPS96]. 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, the 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 [KOB⁺97] by constructing a *contour tree* which contains the local maximal, local minimal and saddle points, and for the first time, can be proved to generate a seed-set that is at most twice the size of the optimal seed-set size. However, the required running time is $O(NlogN)$. Our approach, on the contrary, may not be able to produce the seed-set as small as the contour tree approach does, but it is a *linear time* algorithm, and extremely easy to implement. Among the described related work, we will further detail the min-max span space and vol-

ume thinning approach in later sections as they serve as the foundations of our new approach.
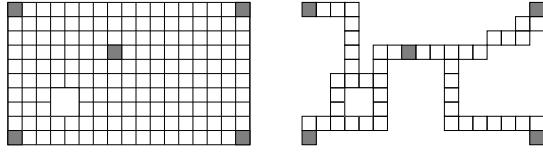
Figure 1: *A 2D volume thinning process. On the left: the original dataset, where the extremum points are marked in dark gray. On the right: The resulting skeleton after the thinning process.*

3. Background

3.1. Volume Thinning

Itoh et al.'s volume thinning approach [IYK96] 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 minimum, in other words, a maximum or minimum compared to all its neighbors. Starting from the whole dataset, with those extremum points marked as non-eliminable, cells that will not affect local connectivity are removed. Here a cell denotes a cube which has eight scalar values defined on its eight grid points separately. The entire process proceeds as if the whole volume gets thinner and thinner, and at the end a skeleton is formed. See Figure 1(modified from [IYK96]) for a demonstration of a 2D thinning process.

3.2. Min-Max Span Space

Min-Max span space was first proposed by Livnat et al. [LSJ96] to solve the cell searching problem in iso-surface extraction. 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 this point corresponds to the minimal scalar value defined on this 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 occur within the region where the half plane corresponding to the equation of $x \leq y$. Moreover, using this representation, given an iso-value c, the cells intersecting with the desired iso-surface are those within the 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, and what our approaches are. We have implemented two variants, and each of them will be detailed in the subsections.
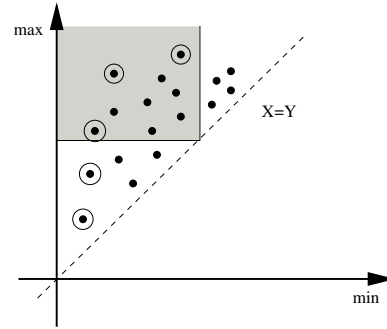
Figure 2: *A min-max span space of a dataset, where each cell is represented as a black dot. The grey area corresponds to an iso-value query, and the cells falling into this area intersect with the desired iso-surface.Those dots who have no upper-left neighbors are marked with circles.*

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 data set. 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 min-max span space domain. 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, given a dot distributed on the upper-left half plane in 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 those 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. Assume S represents the *union* of the set of all the points (cells) which does not have any other points on their upper and left-hand side, then this S must intersect with every iso-surface. Because 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 cell a 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 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 in 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*.

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 without hindering the capability to intersect with all the iso-surfaces that the original cells could intersect with. Second, such cells does not form a complete seed-set. Considering 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 $(30, 70)$. 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 as the seed-set, then for the iso-value query whose iso-values 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 such values. 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.

4.2. Variant 1
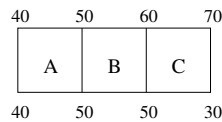
Figure 3: *A 2D counterexample. The numbers are the corresponding scalar values defined on the grid points.*

To take connectivity into account, we start with treating the whole dataset as the seed-set, and then gradually remove 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 its six face-connected neighbors to see if it has a *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, this means that neighbor has a range that includes the cell's range. 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 bookkeeping 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 vice versa from cell B's point of view, as shown in Figure 4. One simple and less precise approach to deal with this exception is to first assign an unique ID to each cell, then

when it comes to cell removal, only the cell with a bigger ID value is removed. However, this simple approach may produce a less optimal result, as shown in Figure 4, 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 4, 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 such a 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 the seed-set. Recall in the iso-surface propagation process, the intersected cells found in the seed-set are used to propagate to locate all the intersected cells with the desired iso-surface. Usually this propagation is performed through face connectivity, however, as an iso-surface could touch a vertex or pass through an edge, we could modify the surface propagation process accordingly. This modification also affects the seed-set construction as the definition of a *neighbor* of a cell gets changed. By taking the new definition into account, our algorithm requires little modification while most of it remains unchanged. Note that there is one more mod-
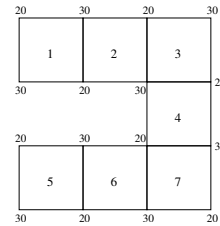
Figure 4: *A 2D example where several cells with exactly the same range are connected together.*

ification to be done during the surface propagation process. Refer to Figure 5 for a 2D illustration. In this Figure, the curves represent the iso-surfaces corresponding to iso-value equal to 25. According to normal surface traversal, such as the one in Bajaj et al. [BPS96], each cell only checks to see if any of its faces (edges or vertices) intersects with the iso-value, surface propagation will proceed along 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 (edge or vertex) neighbors of the current cell intersects with the given iso-value, surface propagation should proceed along that neighbor. There is still another optimization that we could perform and sometimes it can significantly reduce the skeleton size produced by the original thinning algorithm (as will be demonstrated in the performance evaluation section). The trick is when the eight scalar values of a non-isolated cell are all equal to one constant value, this cell can be removed, because this value must also
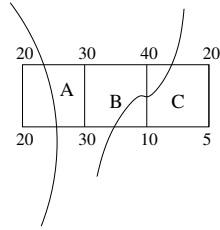
Figure 5: *A 2D example of iso-surface(s). Here the given iso-value is* 25.

appear on one of its neighbors. At run time, when this particular iso-value is requested, one of its neighbors should be included, either because that neighbor belongs to the seed-set, or is reached by the propagation from a 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 the constant values into a *macro cell*, which 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. [BPS96], where a cell could be eliminated if the union of ranges of its faces (edges or vertices) contains its range, then this cell could be eliminated. 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 contain 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 the check separately, the check for the union of neighbor's range could not be checked independently, i.e., we would need to worry the existence of a cell's neighbor in variant 2. Second, as shown in Bajaj et al [BPS96], the order in which the checks are performed could affect the resulting seedset size. Figure 6 demonstrates an example of the first issue, where the range union of cell *A* and cell *C* could be used to cover cell *D*, which thus could be removed. However, the range union of cell *B*, cell *C*, and cell *D* could also be used to cover cell *A*, which indicates that cell *A* is also removable. Nevertheless, it is clear that in this case only one of *A* and *D* could be removed due to this *cyclicity*. A simple but more conservative way to avoid cyclicity entirely is to ensure that whenever a cell is removed, it cannot be further used as a union member to remove other cells. However,

this hinders the possibility of removing a cell while there is no cyclicity presented among the relationships of neighbors. Because of this, we need to keep track of the relationship between a cell and its neighbors when it is removable. To deal with the second issue, and at the same time without resorting to a nonlinear time algorithm as in [KOB⁺97], we perform the union check according to the the number of neighboring cells which could be used to remove a given cell. This is simply due to the fact that a cell with a bigger such number may suggest a larger dependency on others, therefore its check is deferred for better reduction efficiency. Although it seems sorting is unavoidable during the process, in practice, the number of neighboring macro cells are often around a dozen or so, therefore *counting sort* could be used, thus making the total complexity still *linear* in this phase. For a given cell, denoted by *C*, the check for range union
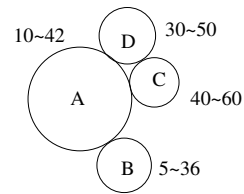
Figure 6: *An example showing the cyclicity of the relationships between a cell and its neighbors.*

of its neighbors is a recursive process. In the representation of min-max span space, all the neighbors of a given cell are represented by points in the four quadrants with cell *C* being the center. If there is a cell in the second quadrant, this means cell *C* has an upper-left neighbor, then it can be removed, as described in variant 1. Assume there exists no cells on the second quadrant, then the next thing is to find the union of neighbors, and because the cells in the fourth quadrant are all dominated by cell *C*, we only need to check the cells in the first and the third quadrants. We first find the leftmost cell in the first quadrant, denoted by *L*, and the topmost cell in the third quadrant, denoted by *T*, then perform the following. If cell *L*'s x-coordinate is larger than cell *C*'s y-coordinate, and since cell *L* is the one with the minimal x-coordinate, this means the there exists a gap between the maximum of the cell *C* and the minimum of cell *L*, therefore it is hopeless that the union of cell *C*'s neighbors could cover cell *C*. Similarly, if cell *T*'s y-coordinate is smaller than cell *C*'s x-coordinate, the check for union could terminate immediately. However, if cell *L*'s x-coordinate is less than cell *C*'s y-coordinate and cell *T*'s y-coordinate larger than cell *C*'s x-coordinate, then union of the range of cell *L* and *T* could *shrink* the original range to cover, i.e.,the range of cell *C*. The remaining *uncovered* range after the union could be viewed as a "dummy cell", and could be recursively checked by the union of those cells in the fourth quadrant. 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); moreover, the number of being a

neighbor of others for each cell is also fixed, thus the overall linear time complexity is guaranteed. As the 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 the run time, just as proposed by [BPS96].

5. Performance Evaluation

We have implemented our system on a Pentium 4 2.8GHz machine with 1GByte memory, running the Windows 2000 Professional 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 comparison study. Table 2 juxtaposes the resulting seed-set sizes by different algorithms, where *Itoh*, *Bajaj* and *Kreveld* represent the results by the work in [IYK96] (volume thinning), [BPS96] (fast iso-contouring), and [KOB+97] (contour-tree), separately. Due to the limit of time, we did not get to re-implement Bajaj et al.'s method [BPS96] and Kreveld et al.'s method [KOB+97], so we extract the number from the paper and downloaded all the datasets available for our comparison. There are several relationships to observe from this table. For convenience and to reduce the table width, we use *W n* to denote the result of running variant *n* algorithm exclusively on the *whole* dataset, and similarly *S n* the result of running variant *n* algorithm exclusively on the resulting *skeleton* produced by the volume thinning approach. In addition, *W 1+2* and *S 1+2* denote the cases where we run both variant 1 and 2 algorithm on the whole dataset and on the resulting skeleton from the volume thinning approach. First, the numbers in *W 1* and *S 1* are all less than the numbers in *W 2* and *S 2*. This is simply because in the variant 2 algorithm we look for more chances to delete a cell, and the test in variant 1 algorithm is just a special case of that in variant 2 algorithm. Second, the numbers in *S 1* and *S 2* are smaller than those in *Itoh* as the former two are built on top of the latter one to seek further possibility of reduction. The most intriguing part is the comparison between *W 1+2* and *S 1+2*, where *W 1+2* wins almost in all the cases except for the *Hydrogenatom* dataset. And the same time, these two datasets are also where *W 1* lost most when compared to *Itoh*. After a detailed analysis we found because this dataset present thin layers of equal values which may be pricked into fragmented parts by the *W 1+2* approach, but can be handled more properly by the volume thinning approach.

For the comparison between *W 1+2* and *Bajaj*, the reason why *W 1+2* wins all the time is also self-evident, as explained previously that the tests performed in *Bajaj* can be deemed as a special case of what is performed in *W 1+2*. Overall, the *W 1+2* can reduce the dataset size by 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 *Bajaj*. The only available number from Kreveld et al. [KOB+97] is the *HIPIP* dataset, where although our algorithm produces a seedset about 60% larger than the one in [KOB+97], which involves higher time complexity, i.e., $O(NlogN)$, our linear time complexity still could find its use. Table 3 presents the timing results of using the *W 1+2* approach, so far the best implementation of ours. 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 PC, all the seed-sets could be generated with a reasonable speed. And most important of all, these constructed seed-sets could be stored or even used for building an *interval tree* to quickly answer repeatedly iso-value queries.

As mentioned previously, in 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 overhead 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. To demonstrate how the completeness of cyclicity check affects the resulting seedset size and running time, we intentionally compare three versions of this approach: one without performing the cyclicity check, one with limited cyclicity check, and the one with full cyclicity check. Here in the limited cyclicity check version we restrict the transitive dependency to be no more than 10 levels. As shown in this Table, even with limited cyclicity check, which thus ensures a linear running time, the produced seedset sizes and timing are still comparable to the ones with full cyclicity check. 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 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 find the union of all floating point scalar values, 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. To summarize, we have proposed an algorithm which makes the following contribution. First, it is extremely simple to implement, while at the same time performs much better than most existing algorithm. Second, it generalizes some existing scheme (such as the one by Bajaj et al. [BPS96]) while still preserving the linear time complexity for building a seed-set at the preprocessing time. Third, it can be easily combined with other approaches, such as volume thinning, to further reduce the size of a seed-set. Finally, it adds two minor optimizations by re-

Data set	Dimension	Range	# of Extremum Points	# of Skeleton Cells
MR Brain	$256 \times 256 \times 109$	$0 \sim 65535$	1489073	2160246
CT Head	$256 \times 256 \times 113$	$0 \sim 65535$	764819	1410784
CT Engine	$256 \times 256 \times 110$	$0 \sim 255$	220783	791428
SOD	$97 \times 97 \times 116$	$0 \sim 255$	2758	41902
HIPIP	$64 \times 64 \times 64$	$-0.55625 \sim 0.58136$	1465	10520
Hydrogenatom	$128 \times 128 \times 128$	$0 \sim 255$	44	1372
Aneurism	$128 \times 128 \times 128$	$0 \sim 255$	6034	33134
Bonsai	$128 \times 128 \times 128$	$0 \sim 255$	43479	155850
Skull	$128 \times 128 \times 128$	$0 \sim 255$	276530	495554
Foot	$128 \times 128 \times 128$	$0 \sim 255$	127554	234405

Table 1: *Characteristics of input data sets used in this performance study.*

Data set	Itoh	W 1	W 2	W 1+2	S 1	S 2	S 1+2	Bajaj	Kreveld
MR Brain	2160246	254713	183072	32170	425159	307039	210531	639891	N/A
CT Head	1410784	257633	125701	10032	334120	281006	198274	423366	N/A
CT Engine	791428	311839	88434	31874	284245	220231	163469	180048	N/A
SOD	41902	52438	7395	5833	17807	11294	9734	13004	N/A
HIPIP	10520	35852	3642	869	5671	3652	2237	4616	529
Hydrogenatom	1732	16079	3103	843	763	400	357	N/A	N/A
Aneurism	33134	1693	1693	1557	5973	4061	3019	N/A	N/A
Bonsai	155850	17389	15433	6952	41039	29942	21732	N/A	N/A
Skull	495554	70456	39418	19346	120530	93304	59646	N/A	N/A
Foot	234405	23322	21974	10581	50267	36708	23848	N/A	N/A

Table 2: *Comparison of the number of seed cells between Itoh et al.'s volume thinning approach and all variants of our approach running on the skeleton produced by Itoh et al.'s approach, under the connectivity of face, edge and vertex.*

moving the cells of constant values and equal-valued cells, which sometimes may help reduce the seed-set significantly.

6. Conclusion

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 by 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, in terms of face, edge and vertex connectivity, it is in fact simpler than the case of regular volume datasets. Second, just like the work done by Kreveld et al. [KOB+97], we will work on deriving a linear-time approximation al-

Data set	No Cyclicity Check		Limited Cyclicity Check		Cyclicity Check	
	Seed Set Size	Time(sec)	Seed Set Size	Time(sec)	Seed Set Size	Time(sec)
MR Brain	145140	10.4	55704	12.5	32170	19.6
CT Head	89716	9.3	35431	15.7	10032	21.2
CT Engine	87429	13.4	45433	42.5	31874	57.2
SOD	6843	5.7	6217	9.3	5833	11.5
HIPIP	2873	3.7	1239	12.5	869	20.9
Hydrogenatom	2461	8.9	1078	89.1	843	175.3
Aneurism	1693	2.1	1608	3.9	1557	4.3
Bonsai	10094	3.6	7536	4.6	6952	5.9
Skull	36592	7.5	24353	10.2	19346	14.5
Foot	14523	3.3	11371	4.8	10581	5.9

Table 3: *Seedset size and running time of the W 1+2 algorithm on all datasets.*

gorithm which could find a seed-set with provably small size.

References

[BPS96]   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.   1, 2, 4, 5, 6

[BpS97a]   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.   1

[BpS97b]   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.   1

[CMM$^+$97]   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.   2

[Ede80]   H. Edelsbrunner. Dynamic Data Structures for Orthogonal Intersection Queries. Technical Report Report F59, Inst. Informationsverarb.,Tech. University Graz, 1980.   2

[Gal91]   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.   2

[IK95]   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.   2

[IYK96]   T. Itoh, Y. Yamaguchi, and K. Koyamada. Volume Thinning for Automatic Isosurface Propagation. In *IEEE Visualization '96*, pages 303–310, October 1996.   2, 3, 6

[KOB$^+$97]   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.   2, 5, 6, 7

[LC87]   W. E. Lorensen and H. E. Cline. Marching Cube: A High Resolution 3D Surface Construction Algorithm. *Computer Graphics*, 21(4):163–169, July 1987.   2

[Lev88]   M. Levoy. Display of Surfaces from Volume Data. *IEEE Computer Graphics and Applications*, 8(3):29–37, March 1988.   2

[Lev90]   M. Levoy. Efficient Ray Tracing of Volume Data. *ACM Transactions on Graphics*, 9(3):245–261, 1990.   2

[LSJ96]   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.   1, 2, 3

[SHLJ96]   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.   2

[WG92]   J. Wilhelms and A. V. Gelder. Octrees for Faster Isosurface Generation. *ACM Transactions on Graphics*, 11(3):201–227, July 1992.   2