Parallel View-Dependent Isosurface Extraction Using Multi-Pass Occlusion Culling

Jinzhu Gao and Han-Wei Shen
Department of Computer and Information Science
The Ohio State University
Columbus, Ohio 43210
E-mail: gao@cis.ohio-state.edu and hwshen@cis.ohio-state.edu

Abstract

This paper presents a parallel algorithm that can effectively extracts only the visible portion of isosurfaces. The main focus of our research is to devise a load-balanced and output-sensitive algorithm, that is, each processor will generate approximately the same amount of triangles, and cells that do not contain the visible isosurface will not be visited. A novel multi-pass algorithm is proposed in the paper to achieve these goals. In the algorithm, we first use an octree data structure to rapidly skip the empty cells. An image space visibility culling technique is then used to identify the visible isosurface cells in a progressive manner. To distribute the workload, we use a binary image space partitioning method to ensure that each processor will generate approximately the same amount of triangles. Isosurface extraction and visibility update are performed in parallel to reduce the total computation time. In addition to reducing the size of output geometry and accelerating the process of isosurface extraction, the multi-pass nature of our algorithm can also be used to perform time-critical computation.

CR Categories: I.3.1 [Computer Graphics]: Hardware Architecture—Parallel processing; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—visible line/surface algorithms

Keywords: view-dependent, occlusion culling, isosurface extraction, multi-pass algorithm, hierarchical data structures

1 Introduction

Displaying isosurfaces is an effective technique for visualizing three-dimensional scalar fields as it can precisely reveal the geometric structure of the underlying field's value distribution. To extract isosurfaces, intensive research effort has been undertaken in the past decade, which resulted in many robust and efficient algorithms such as the well-known Marching Cubes algorithm [8], and various accelerating techniques [16, 15, 7, 14, 5, 1, 2] that can significantly reduce the complexity of isosurface cell search process. While effective, these isosurface extraction algorithms can generate a huge number of triangles for a large-scale dataset, which can easily overwhelm the underlying computing power due to the requirements in the speed of CPU, the performance of graphics hardware , and the capacity of main memory. To reduce the cost of surface generation and rendering, researchers have proposed various approaches such as surface decimation schemes [13], view-dependent methods [6, 11], and parallel rendering algorithms [10, 12]. In general, surface decimation is mostly performed as a post-processing step and thus is not suitable for interactive exploration of isosurfaces with different isovalues. For view-dependent methods, the challenge is to reduce the overhead for computing the visibility information. Utilizing massively parallel computers or PC clusters to extract isosurfaces can be very effective. The main challenge is to design an output-sensitive and load-balanced algorithm so that the available processors are fully utilized to perform only useful calculations.

In this paper, we present a parallel view-dependent algorithm for large-scale isosurface extraction. The main purpose of our research is to reduce the number of triangles extracted from an isosurface while still allowing the user to explore different isovalues interactively at run time. To achieve the goals, our algorithm performs visibility culling and isosurface extraction in a coherent manner so that only the visible portion of an isosurface will be generated. The crux of the problem is to balance the workload of surface generation, so that the number of visible triangles generated by each processor will be approximately the same. In addition, we intend to design an output-sensitive algorithm so that most of the cells that do not contain the visible portion of the isosurface will not be visited. The main challenge for us is to perform visibility determination and load balancing without actually generating the complete isosurface. To solve the problem, we propose a multi-pass algorithm that can find the visible cells progressively and extract isosurface from those cells in parallel. Our algorithm starts with a front-to-back octree traversal to skip empty cells rapidly and collect isosurface cells. A progressive image space visibility culling is then used to identify visible isosurface cells on the fly. To distribute the workload, we use a binary image space partitioning algorithm to distribute the visible cells to different processors. Isosurface extraction and visibility update are performed in parallel to speed up the overall computation. In addition to accelerating isosurface extraction and reducing the size of output geometry, the multi-pass nature of our algorithm allows the user to stop the processing at an early time, which provides the possibility of time-critical computation. In the following, we first briefly survey the related work in section 2. Then the details of the algorithm are given in section 3. The experimental results and analysis are presented in section 4. Finally, we conclude our paper by summarizing our contributions and discussing the future work in section 5.

2 Related Work

Displaying isosurfaces is an effective technique to analyze three-dimensional scalar fields generated from large-scale numerical simulations. Traditionally, the Marching Cubes algorithm [8] is used to perform such a task. Although the algorithm is effective, it requires a linear search for the isosurface cells, which can be inefficient for large-scale datasets. To speed up the search process, researchers have proposed various acceleration techniques. Examples include the use of octree data structures [16] for a rapid skip of empty cells. An advantage of using octrees is that the isosurface cells can be traversed in a front-to-back order, which is a desired feature for view-dependent algorithms. In addition to the octree algorithm, many value decomposition methods [15, 7, 14] have also been proposed. These algorithms can achieve near optimal speed for finding isosurface cells. The disadvantage of value-based methods is that they do not maintain the spatial relationships among the extracted isosurface cells. Algorithms using contour propagation [5, 1] can also be very effective. Their challenge is to identify the initial seed cell to begin the propagation. Although all of the above algorithms can speed up the search of isosurface cells, for a large dataset, extracting and rendering millions of triangles interactively still remains to be a major challenge.

To reduce the number of triangles generated in an isosurface, Livnat and Hansen [6] proposed a view-dependent isosurface extraction algorithm that applies the occlusion culling technique proposed by Greene *et al.*[4] to isosurface extraction. By culling away the occluded isosurface patches, the algorithm improves the performance in both extraction and rendering time. The challenge of the algorithm is that calculating the visibility information incurs extra overhead, which can be a potential bottleneck of the process.

Another method to visualize the visible portion of an isosurface is to use the interactive ray tracing algorithm [11] proposed by Parker *et al.*. The algorithm computes the intersection point of a ray and the isosurface directly by solving a cubic equation without extracting triangles explicitly. In general, this algorithm is more suitable for parallel machines with a large number of processors.

Various parallel polygon rendering algorithms [10, 12] have been proposed to speed up the rendering of large scale polygonal datasets. To utilize the utility of parallel rendering, Gao, Shen and Garcia [3] previously proposed a parallel visible isosurface extraction algorithm in conjunction with a sort-first parallel polygon rendering method to improve the interactivity. However, the load balance was not satisfactory as the algorithm distributes the entire isosurface cells instead of the visible isosurface cells to the processors, which can result in load imbalance because the numbers of triangles generated by the processors can be very different.

3 Parallel View-Dependent Isosurface Extraction

In this section, we present an algorithm that can effectively cull away the invisible part of the isosurface, distribute the workload evenly to multi-processors and extract the visible isosurface in parallel. The primary goal of our algorithm is to balance the workload of isosurface generation and visibility determination, and to ensure that each processor will generate approximately the same amount of visible triangles. To achieve this goal, we design an algorithm that consists of four key components. First, an octree data structure similar to the one proposed in [16] is used to identify the cells that contain the isosurface. In our algorithm, a cell is a leaf node of the octree, which is a $m \times n \times l$ sub-volume. The values of m, n and l are determined at run time. We use the term "active cells" to refer to the cells that contain the isosurface, and "empty cells" to represent the non-active cells. With this octree data structure, active cells can be efficiently identified by comparing the min/max values of the octree nodes with the isovalue. The second component of our algorithm is a multi-pass occlusion culling scheme used to identify the visible cells, where a "visible cell" is an active cell that contains the visible portion of the isosurface. Our occlusion culling scheme is a multi-pass process as it is impossible to identify all the visible cells in one pass without actually extracting the isosurfaces. This is because we cannot decide whether an active cell is visible or not until we generate the portion of the isosurface in front of it. The third component of our algorithm is an image space partitioning scheme which distributes the visible cells among the processors by partitioning the image space based on the cells' projection areas. The last component of our algorithm is the visible isosurface extraction and visibility update process performed by multiple processors in parallel.

In the following, we first provide an overview of our algorithm by describing the interplay of these four components, and then describe the details for each of the components.

3.1 Algorithm Overview

Our algorithm begins with traversing the octree data structure in a front-to-back order, and collects the active cells into an active cell list by comparing the isovalue with the min/max values stored at each octree node. For each active cell in the active cell list, we determine its visibility using a conservative visibility test. An active cell is conservatively determined to be visible if its projected bounding box is not completely covered by the projection area of previously generated isosurface or the projected bounding boxes of active cells in front of it. To ensure this visibility test can be done efficiently, the projected bounding box of a cell is the rectangular area defined by the cell's projected min/max x and y coordinates in the image space. It is a conservative test because the actual isosurface inside the visible cell can still be invisible. However, we will not mistakenly mark a visible cell to be invisible. For those cells whose projection areas are covered by the projected bounding boxes of the front active cells, we cannot conclude that it is not visible since the triangles from the front active cells have not been extracted yet. Active cells that are not determined to be visible in the current pass will be tested again later. After the visibility test, the host node partitions the image space based on the visible cells' projection areas, and evenly distributes the visible cells to the slave nodes. After receiving the visible cells, each slave node will extract the isosurface patches independently using the Marching Cubes method and update a local coverage mask to mark the occupied pixels on the screen. The local coverage masks from the slave nodes will be composited together, and the aggregated coverage mask is sent back to the host node. Having received the coverage mask obtained from the extracted visible isosurface, the host node performs the following two tasks. First, the active cell list is scanned again and those cells that have projected bounding boxes completely occluded by the already generated isosurface are guaranteed to be invisible and thus can be safely discarded. Second, the conservative visibility test just mentioned is performed again to the remaining active cells. This results in another batch of visible cells, which will be evenly distributed to the slave nodes for surface extraction. After this, the same aforementioned process is repeated and our algorithm iterates until no more visible cells are found. Figure 1 provides a flow chart of our algorithm.

Our algorithm requires multiple passes before the entire visible portion of the isosurface can be extracted. We use this multi-pass algorithm to identify visible cells and extract isosurfaces because it is impossible for the algorithm to find all the visible cells in one pass without generating the entire isosurface. With the proposed multi-pass algorithm, we are able to gather the visibility information progressively, use the information to identify the visible cells,

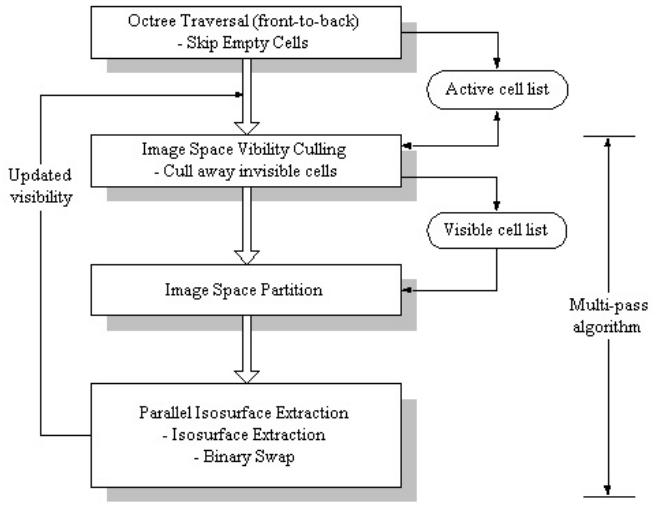

Figure 1: The algorithm data flow.

and evenly distribute the workload of isosurface extraction. Having the isosurface extraction performed in parallel not only speeds up the triangulation process, but also distributes the work of updating the visibility information among the multi-processors so that the computation overhead can be minimized. As an added benefit, our multi-pass algorithm can also allow the user to terminate the generation of the visible isosurface during or at the end of a desired pass to perform time-critical computation.

In the following sections, we describe these components of our multi-pass parallel algorithm in detail. We first introduce the technique we use to find the visible cells and remove invisible cells. Then we describe the method to partition the workload evenly for the processors. Finally we discuss the parallel isosurface extraction and visibility update process performed by each processor.

3.2 Progressive Image Space Visibility Culling

In each pass of the algorithm, the host node identifies the visible cells and removes the invisible cells. This is achieved by maintaining at the host node a two-dimensional global coverage mask that has the same resolution as the final image. Each entry in the coverage mask represents a pixel in the screen and can have three different values: 0, 1 and 2. An entry with a value 0 indicates that the pixel is uncovered. An entry with a value 1 indicates that the corresponding pixel is covered by a previously extracted isosurface patch. An entry with a value 2 indicates that the corresponding pixel is covered by the projected bounding box of an active cell in the current pass. At the beginning of the first pass, every entry of the coverage mask is set to 0. To decide the visibility of an active cell, we first project the cell onto the image plane and compute the projected rectangular bounding box based on the min/max x and y coordinates. Within the area of the bounding box, we check the values of the entries in the coverage mask. There can be three cases:

- At least one coverage mask entry has a value 0. We report the active cell is a visible cell. This cell is removed from the active list, and moved to a visible cell list. In the mean time, we fill all the entries of the coverage mask within the cell's projected bounding box with a value 2 except those entries with a value 1.

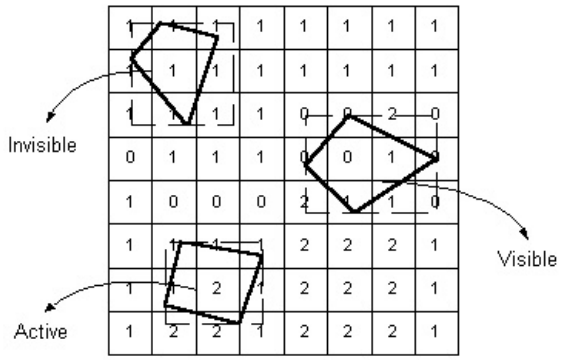

Figure 2: Examples of the classification of active cells based on the projected bounding boxes and the global coverage mask.

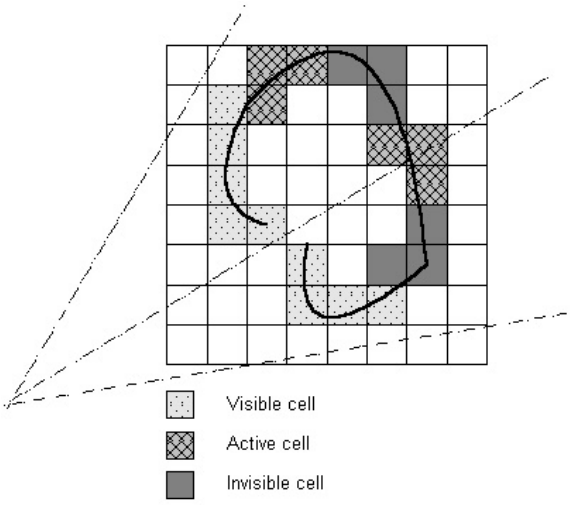

Figure 3: A 2D example of the classification of active cells after the first pass.

- All the coverage mask entries have values equal to 1. We can discard the cell from the active cell list as its bounding box is completely covered by previously extracted isosurface. We note that an entry can have a value 1 only after the first pass of our algorithm. That is, a portion of the visible isosurface has been extracted and the pixel is covered by an isosurface patch. This information is obtained from an aggregated coverage mask returned from the slave nodes.

- If all the coverage mask entries within the projected bounding box have values 1 or 2 and at least one entry has a value 2, this cell remains as an active cell and its visibility is undetermined at the current pass. We will check the cell again at a later pass.

Figure 2 shows these three cases classifying the active cells based on their projected bounding boxes. A 2D example is given in figure 3.

After the visibility test, the current batch of visible cells are passed to the image space partitioning routine to determine the workload distribution. In the mean time, we reset the coverage mask entries that have the values 2 to 0, as there is no need to keep the estimated visibility information from the cells' bounding boxes

anymore. Instead, the host node will wait until the visible isosurface patches to be extracted and then update the global coverage mask.

3.3 Image Space Partitioning

After a pass of the image space visibility culling is completed, we can begin to partition the collected visible cells to a number of subsets and assign each subset to a processor.

First we need to define the workload. Obviously the number of triangles that will be extracted from each cell is an optimal choice to measure the workload for each cell. However, it is impossible for us to know the number beforehand. To perform an estimation, in the process of identifying active cells, we traverse down an extra k steps in the octree from the $m \times n \times l$ cell level to count the number of smaller nodes in the cell that have min/max values intersecting the isovalue. The number of extra levels can be controlled based on the constraint in the computation time and the accuracy of the prediction. A larger k results in more accurate workload prediction but makes the process more costly.

Once we obtain an estimate of the workload for each visible cell, we can start to partition the cells into different subsets and assign each subset to a processor. Our partitioning scheme has two criteria. First, we intend to let each processor extract approximately the same number of triangles. Second, we intend to assign visible cells that are close to each other to the same processor. This way, updating the visibility information becomes more efficient. Furthermore, keeping adjacent visible cells in the same processor can benefit the down-stream parallel rendering algorithm if a sort-first, or a hybrid of sort-first and sort-last scheme [10, 12] is used.

We use a binary space partitioning scheme to determine the workload distribution. The process works as follows: Initially the entire screen is treated as a single tile and we divide the tile into two new tiles by splitting along its longest dimension. Suppose $xdim$ and $ydim$ are the dimensions of current tile in x and y directions respectively and $(xmid, ymid)$ is the coordinate of the center point of the projected bounding box for each cell. Assuming x is the current longest dimension, we divide the tile into two new tiles by partitioning the x dimension at where the total workload of the visible cells that have their $xmids$ on the left side is approximately the same as that on the right side. After this is done, we identify the next dimension to divide. This process is repeated until the number of tiles is equal to the number of processors. The result of this partitioning process is a binary tree where the leave nodes of the tree represent the final partition of the image space and define the workload assigned to each processor. That is, each processor will receive all the visible cells in one tile. As each tile contains a similar amount of workload, the workload for each processor will be similar. Figure 4 shows an example of binary image space partitioning when using two processors.

3.4 Parallel Isosurface Extraction

After we partition and distribute the visible cells, the processors will begin to extract the isosurfaces in parallel. The parallel isosurface extraction algorithm contains two steps: isosurface extraction and binary swap. The Marching Cubes method is used to extract the isosurface from the visible cells and a local hierarchical coverage mask proposed by Greene [4] is used to update the occlusion information at each processor. After the isosurface patches are extracted, the local coverage masks from the slave nodes will be composited together and sent back to the host node, which will use the updated visibility information in the next pass to determine another batch of visible cells. The process of compositing the coverage masks is done using a binary swap algorithm to reduce the communication overhead.

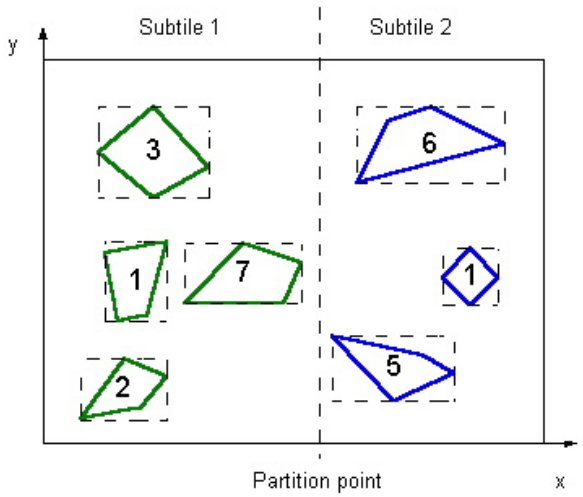

Figure 4: An example of binary image space partitioning when using two processors. The number inside each box represents the workload of the corresponding cell.

3.4.1 Isosurface Extraction

The slave nodes begin to extract the isosurfaces from the given visible cells as soon as they receive the workload assignments. As we do not distribute the octree to each slave node, Marching Cubes algorithm is used to loop through the voxels within the visible cells and extract triangles. Each time a new triangle is extracted, it is treated as an occluder and the occlusion information needs to be updated. This is done by updating the local hierarchical coverage mask as proposed by Greene [4] within each slave node.

In our implementation, the top level of the hierarchical coverage mask covers the entire image plane, which is divided into 8×8 tiles. One bit is used to represent a tile. The bit has a value 1 if the corresponding tile is completely covered; otherwise it is set to 0. For each tile, we recursively divide it into another level of 8×8 subtiles, and create a new $8 \times 8 = 64$ bit coverage mask for each subtile. The bottom level of the coverage mask is one level above the pixel level, and each 64-bit data in the array of this level represents 64 pixels covered by the sub-tile. When a triangle is extracted, we use a fast table lookup method proposed by Greene [4] to mark the corresponding bits in the hierarchical coverage mask.

When a slave node completes the extraction of its portion of visible isosurface and updates its local hierarchical coverage mask, it needs to send the bottom level of the hierarchical coverage mask back to the host node so that the host node can use the information to select the next batch of visible cells. This is done through a binary swap algorithm which composites the local coverage mask from the multiprocessors.

3.4.2 Binary Swap

We have one global coverage mask at the host node and each slave node has its own local coverage mask. At the end of each pass of visible isosurface generation, the local coverage masks are combined and sent back to the host node. A pixel in the global coverage mask is covered if it is covered in any of the local coverage masks. To minimize the communication overhead caused by sending the local coverage masks, we use the binary swap algorithm to perform the compositing and gathering.

The basic idea of binary swap is similar to that used in image

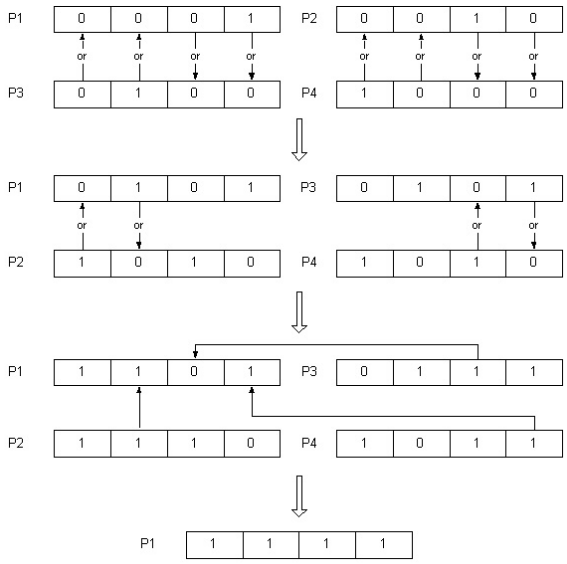

Figure 5: An example of binary swap using 4 processors. Each processor has a 4-bit coverage mask. P1 is the host node.

composition [9]. Each processor has a partner and they will exchange half of its array with each other. Each processor will conduct a bit-wise OR operation on the array entries received from its partner with its own corresponding array entries. The case involving 4 processors is shown in Figure 5. During the whole process, each processor is equally busy. Suppose there are p processors and the array size is n. Each processor is then responsible to update the n/p coverage mask entries. Finally, each processor sends back correct results of n/p array entries to the host node. The host node can now use this information to find visible cells and remove invisible cells for the next pass.

4 Results

Our algorithm is tested on an SGI Origin 2000 configured with 40 250MHZ IP27 processors and 16384 Mbytes of main memory. Each processor has 32 Kbytes data cache, 32 Kbytes instruction cache and 4 Mbytes secondary unified cache. Our parallel multi-pass isosurface extraction algorithm is applied to visualize a 256*256*145 UNC brain dataset and a 512*512*617 dataset for the leg section of the Visible Woman CT data.

Our algorithm can significantly reduce the number of triangles extracted by culling away the invisible portion of the isosurface. Table 1 compares the number of extracted triangles of visible isosurface with that of the whole isosurface while using two test datasets in two different view directions. From the table, it can be seen that large portions of the isosurfaces were culled away. Figure 6 shows an image of visible portion of the isosurface for UNC brain dataset. Because the total number of triangles being produced is significantly reduced, we are able to save both extraction and rendering time without affecting the correctness of the final image.

Our algorithm can also achieve a good speedup while using multiple processors. For the UNC brain dataset and Visible Woman's leg dataset, the speedups that the algorithm achieved with different numbers of processors are shown in figure 7 and figure 8 respectively. In our experiments, we used $8 \times 8 \times 8$ cell for UNC brain dataset and $16 \times 16 \times 16$ cell for Visible Woman's leg dataset as the unit for occlusion culling and work distribution. we used smaller

Dataset	View	Visible Isosurface (in triangle number)	Whole Isosurface (in triangle number)
UNC brain	1st	188,371 (17.8%)	1,056,866
UNC brain	2nd	235,928 (22.3%)	1,056,866
Visible Woman	1st	1,175,429 (56.7%)	2,072,636
Visible woman	2nd	1,056,001 (50.9%)	2,072,636

Table 1: Comparisons between the number of triangles extracted for the visible isosurface and for the whole isosurface.

Figure 6: An example of view-dependent isosurface extraction using UNC brain dataset. The visible portion of the isosurface is rotated so that we can see the culled portion of isosurface.

cells to estimate the workload as described previously. Figure 7 shows the speedup for UNC brain dataset using different workload approximation and different number of processors. When we used $4 \times 4 \times 4$ cells to estimate the workload, the efficiency was 76% and 58% for 8 processors and 16 processors respectively. When we used voxels to estimate the workload, the efficiency became 88% and 69% for 8 processors and 16 processors respectively. Figure 8 shows the speedup for Visible Woman's leg dataset. When we used $8 \times 8 \times 8$ cells to estimate the workload, the efficiency was 77% and 62% for 8 processors and 16 processors respectively. When we used voxels to estimate the workload, the efficiency was 84% and 63% for 8 processors and 16 processors respectively. For both datasets, the actual time spent using different processors is shown in table 2 and 3. We can see that extended traversal levels in the octree can improve the algorithm's performance as we can more precisely estimate the workload associated with each cell. In these two tables, we also give the time spent using the Marching cubes algorithm and assume we can get the optimal speedup when using multiple processors without considering the difficulty in load balancing. We can see that our algorithm performs much better because of reduced triangle numbers and good load balance. The figure 9 shows the actually time spent in different parts of the algorithm extracting isosurface patches from UNC brain dataset using different number of processors. Isosurface extraction time shows the time spent in extracting isosurface as well as updating coverage mask. Visibility time includes the time spent in visibility determi-

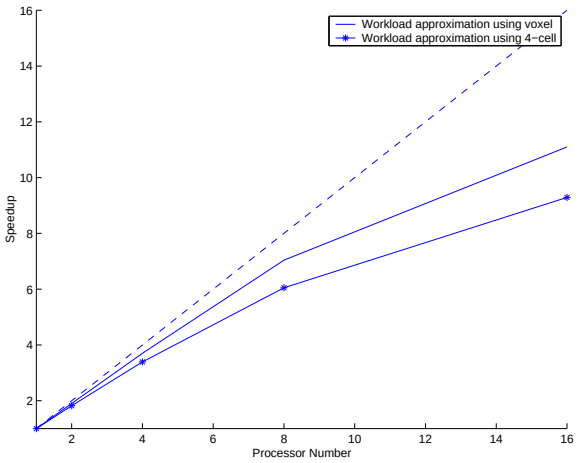

Figure 7: Speedup curves for multiple processors (UNC brain dataset). 4-cell represents a cell whose size is approximately $4 \times 4 \times 4$.

Processor Number	Brain 4-cell (seconds)	Brain voxel (seconds)	Brain Marching Cubes (seconds)
1	6.78	7.88	31.54
2	3.71	4.13	15.77
4	2.00	2.13	7.89
8	1.12	1.12	3.94
16	0.73	0.71	1.97

Table 2: Actual isosurface extraction time using different number of processors for UNC brain dataset.

nation at the beginning of every pass. Workload partition time is the time spent in partitioning the workload among all processors based on the projection area of each visible cell. Work assignment time shows the time spent in distributing the work to every processor. And mask compositing time is the time spent in compositing updated coverage masks gotten from all processors at the end of each pass. The figure shows that the isosurface extraction time is well scaled down when using more processors, and the overhead incurred by other computation is relatively very small.

To get good speedup, the overall workload should remain approximately constant when the number of processors in use becomes larger, and the workload should be evenly distributed to all the processors. Our algorithm can satisfy both criteria. First, our algorithm ensures that the granularity of image space partitioning does not affect the result of visible cell determination. This is achieved by having the host node find the visible cells based on the global coverage mask at the beginning of each pass, which makes the visible cell number independent of the processor number. In our experiments, using 25.5 as the isovalue, the numbers of visible triangles extracted from UNC brain dataset using 1, 2, 4, 8 and 16 processors are all $188,371$. And using 720.0 as the isovalue, the numbers of visible triangles extracted from Visible Woman's leg dataset using 1, 2, 4, 8 and 16 processors are all $1,175,429$. Second, by combining the multi-pass occlusion culling with the image space partitioning technique, our algorithm can ensures that the workload assigned to multiple processor are approximately even.

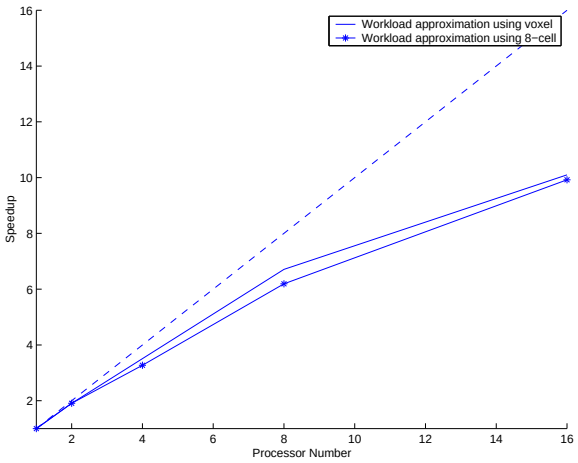

Figure 8: Speedup curves for multiple processors (Visible Woman's leg dataset). 8-cell represents a cell whose size is approximately $8 \times 8 \times 8$.

Processor Number	Woman 8-cell (seconds)	Woman voxel (seconds)	Woman Marching Cubes (seconds)
1	55.67	55.84	365.79
2	29.19	29.13	182.90
4	17.01	15.93	91.45
8	9.00	8.32	45.72
16	5.61	5.51	22.86

Table 3: Actual isosurface extraction time using different number of processors for Visible Woman's leg dataset

To measure the workload imbalance, we use the following formula:

$$Load_{Difference} = (100 \times \frac{load_{Max} - load_{Min}}{load_{Total}})\%$$

The overall load differences for both test datasets using different number of processors are shown in table 4. Table 5 and table 6 give examples of the load differences among 4 processors in each pass, as well as the total, for both test datasets. From the tables, it can be seen that the load differences are all very small, especially when compared to the total workload. Each pass still has a slight load imbalance among the processors because we distribute one group of visible cells to each processor without precisely knowing how many isosurface triangles we will extract from each cell. From the tables, we can also notice that in the last several passes, the load difference becomes larger. This is because the number of visible cells and thus the corresponding workload are reduced dramatically in the last several passes. When the number of visible cells becomes small, it is harder to exactly partition the cells so that every processor's workload is the same. Even though the slight load difference still exists, for a large-scale dataset that contains a large number of triangles for an isosurface, the workload difference is relatively small, which can be verified from our experimental results.

Besides reducing the total number of triangles for an isosurface and achieving good speedup, our algorithm can be used for time-critical computation due to its multi-pass nature. That is, we can stop the extraction at the end of an early pass and still receive a good approximation to the the final visualization result. From table 5 and

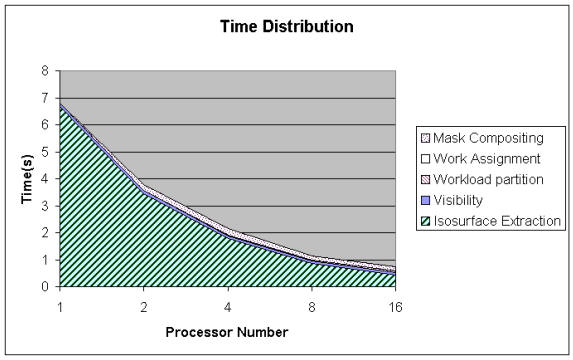

Figure 9: Time distribution for UNC brain datasets

Processor Number	UNC Brain Load Difference	Visible Woman Load Difference
2	2.48%	0.29%
4	2.58%	2.06%
8	1.33%	1.58%
16	1.27%	0.36%

Table 4: Load differences in extracting isosurfaces from two test datasets using different number of processors.

table 6, it can be seen that the number of visible cells extracted decreased progressively at the later passes of our algorithm, and a majority of the visible isosurface was extracted at the beginning. Figure 10 and figure 11 show the percentages of the final visible isosurfaces that were extracted after each pass for both datasets. The corresponding images generated at the end of several passes are shown in Color Plate 1 and Color Plate 2. It can be seen that we were able to achieve a good approximation of the final images even before the entire extraction process completed.

5 Conclusion and Future Work

The paper presents a new multi-pass algorithm for parallel view-dependent isosurface extraction. First, by traversing an octree in the front-to-back order, our algorithm can skip empty cells efficiently and collect active cells in the same order. Second, with the help of progressive image space visibility culling, visible cells can be successfully identified on the fly, and a large portion of isosurface can be culled away. This enables us to save both the triangulation as well as the rendering time. Third, using binary image space partitioning, the workload can be partitioned evenly among the processors. Furthermore, isosurface extraction is done in parallel by multiple processors and we use a binary swap algorithm to speed up the combination of local coverage masks. The multi-pass nature of our algorithm allows us to stop the processing at an early pass, which provides the possibility of time-critical computation. Finally, as each processor generates approximately the same number of triangles, our algorithm can be easily integrated with most of the parallel polygon rendering algorithms without the need of massive communication to redistribute the triangles.

More work needs to be done in the future. Currently our algorithm assumes each processor has a copy of the data, which may limit the size of the datasets that we can process. We will design an efficient run-time data partitioning and distribution algorithm to reduce the local memory requirement. Another direction of our future work is to take into account the frame-to-frame coherence and

Pass	Cell Number	Total Workload (in triangle number)	Load Difference
1	791	49338	2.8%
2	615	73183	3.1%
3	284	37512	1.8%
4	134	15717	3.3%
5	80	7677	4.4%
6	32	3123	8.5%
7	7	596	19.1%
8	5	619	34.1%
ALL	1952	188371	2.58%

Table 5: Load differences in extracting isosurfaces from UNC brain dataset using 4 processors in each pass (8th pass uses only 2 processors and pass 9 to pass 12 use only 1 processor because of low workload).

Pass	Cell Number	Total Workload (in triangle number)	Load Difference
1	1163	386440	2.60%
2	884	436742	1.19%
3	342	156072	2.43%
4	187	83592	4.15%
5	129	56638	4.60%
6	75	29571	5.59%
7	43	14872	8.38%
8	21	3920	22.88%
9	6	3008	36.17%
10	5	2041	2.60%
ALL	2860	1175429	2.06%

Table 6: Load differences in extracting isosurfaces from Visible Woman's leg dataset using 4 processors in each pass (10th pass uses 2 processors and pass 11 to pass 13 use only one processor because of low workload).

minimize the amount of work required for visibility determination. Finally, the bottleneck of our algorithm is updating coverage mask in software. We are now searching for a new algorithm or hardware solution for fast visibility update and determination both for the host and the slave nodes.

Acknowledgements

This work was supported by The Ohio State University Research Foundation Seed Grant. Special thanks to Charles Hansen, Steven Parker and Chris Johnson at University of Utah for providing the test environment and useful information. The Visible Woman dataset is provided by the National Library of Medicine.

References

[1] Chandrajit L. Bajaj, Valerio Pascucci, and Daniel R. Schikore. Fast isocontouring for improved interactivity. In *Proceedings of Symposium on Volume Visualization '96*, pages 39–46. IEEE Computer Society Press, Los Alamitos, CA, 1996.

[2] Paolo Cignoni, Paola Marino, Claudio Montani, Enrico Puppo, and Roberto Scopigno. Speeding up isosurface extraction using interval trees. *IEEE Transactions on Visualization and Computer Graphics*, 3(2):158–170, 1997.

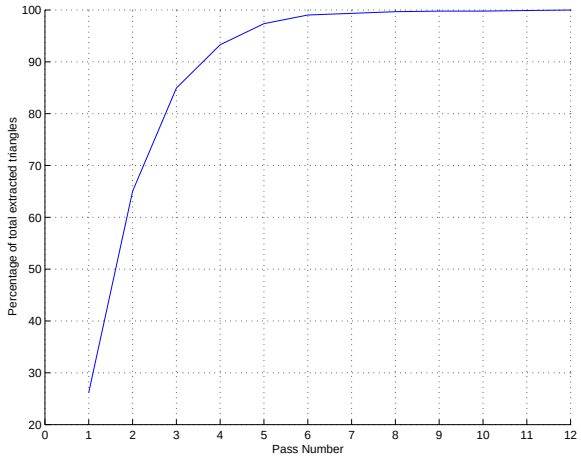

Figure 10: Percentage of extracted triangles after each pass (UNC brain dataset).

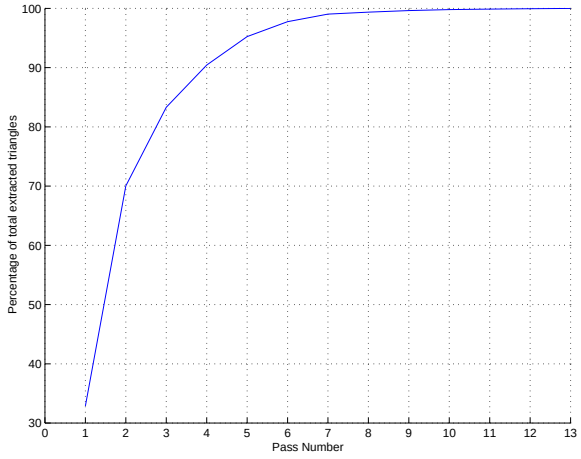

Figure 11: Percentage of extracted triangles after each pass (Visible Woman's leg dataset).

[3] Jinzhu Gao, Han-Wei Shen, and Antonio Garcia. Parallel view dependent isosurface extraction for large scale data visualization. In *Proceedings of the Tenth SIAM Conference on Parallel Processing for Scientific Computing 2001*. SIAM Activity Group on Supercomputing, 2001.

[4] Ned Greene. Hierarchical polygon tiling with coverage masks. In *Proceedings of SIGGRAPH 96*, pages 65–74. ACM SIGGRAPH, 1996.

[5] Takayuki Itoh and Koji Koyamada. Isosurface generation by using extreme graphs. In *Proceedings of Visualization '94*, pages 77–83. IEEE Computer Society Press, Los Alamitos, CA, 1994.

[6] Yarden Livnat and Charles Hansen. View dependent isosurface extraction. In *Proceedings of Visualization '98*, pages 175–180. IEEE Computer Society Press, Los Alamitos, CA, 1998.

[7] Yarden Livnat, Han-Wei Shen, and Christopher R. Johnson. A near optimal isosurface extraction algorithm using the span space. *IEEE Transactions on Visualization and Computer Graphics*, 2(1):73–84, 1996.

[8] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *Computer Graphics*, 21(4):163–169, July 1987.

[9] Kwan-Liu Ma, James S. Painter, Charles D. Hansen, and Michael F. Krogh. Parallel volume rendering using binary-swap compositing. *IEEE Computer Graphics and Applications*, 14(4):59–68, 1994.

[10] Steven Molnar, Michael Cox, David Ellsworth, and Henry Fuchs. A sorting classification of parallel rendering. *IEEE Computer Graphics and Applications*, 14(4):23–32, 1994.

[11] Steven Parker, Peter Shirley, Yarden Livnat, Charles Hansen, and Peter-Pike Sloan. Interactive ray tracing for isosurface rendering. In *Proceedings of Visualization '98*, pages 233–238. IEEE Computer Society Press, Los Alamitos, CA, 1998.

[12] Rudrajit Samanta, Thomas Funkhouser, Kai Li, and Jaswinder P. Singh. Hybrid sort-first and sort-last parallel rendering with a cluster of pcs. In *Proceedings of Eurographics Woskshop on Graphics Hardware*. ACM SIGGRAPH, August 2000.

[13] William J. Schroeder, Jonathan A. Zarge, and William E. Lorensen. Decimation of triangle meshes. *Computer Graphics*, 26(2):65–70, 1992.

[14] Han-Wei Shen, Charles D. Hansen, Yarden Livnat, and Christopher R. Johnson. Isosurfacing in span space with utmost efficiency (ISSUE). In *Proceedings of Visualization '96*, pages 287–294. IEEE Computer Society Press, Los Alamitos, CA, 1996.

[15] Han-Wei Shen and Christopher R. Johnson. Sweeping simplices: A fast isosurface extraction algorithm for unstructured grids. In *Proceedings of Visualization '95*, pages 143–151. IEEE Computer Society Press, Los Alamitos, CA, 1995.

[16] Jane Wilhelms and Allen Van Gelder. Octrees for faster isosurface generation. *ACM Transactions on Graphics*, 11(3):201–227, 1992.