View Dependent Isosurface Extraction

Yarden Livnat Charles Hansen

University of Utah

Abstract

We propose a new approach to polygonal isosurface extraction that is based on extracting only the visible portion of the isosurface. The visibility tests are done in two phases. First, coarse visibility tests are performed in software to determine the visible cells. These tests are based on hierarchical tiles and shear-warp factorization. The second phase resolves the visible portions of the extracted triangles and is accomplished by the graphics hardware.

While the latest isosurface extraction methods have effectively eliminated the search phase bottleneck, the cost of constructing and rendering the isosurface remains high. Many of today's large datasets contain very large and complex isosurfaces that can easily overwhelm even state-of-the-art graphics hardware. The proposed approach is output sensitive and is thus well suited for remote visualization applications where the extraction and rendering phases are done on a separate machines.

1 Introduction

The *marching cubes* [11, 23] method demonstrated that isosurface extraction can be reduced to solving a local triangulation problem through a table lookup. To achieve this, the marching cubes method visits each and every cell of the data. Recently, we showed [10] that the extraction problem can be viewed as a search problem over the *span space* and proposed the *near optimal isosurface extraction* (NOISE) method. Both NOISE and the later *optimal isosurface extraction* method by Cignoni *et. al* [1], reduce the search to visit practically only the cells that *contain* the isosurface.

While in the past, the bottleneck in isosurface extraction was locating the cells that contain the isosurface, today's large datasets pose new challenges. Large datasets of 0.5-4GB can be found in medicine as well as geo-sciences applications. The size of isosurfaces extracted from these datasets can reach several million polygons, many of which are less than one pixel in size. Two problems emerge due to the vast number of polygons. First, due to the shear number of cells containing an isosurface, the computation of all the local triangulations can be very time consuming. Second, the huge number of polygons can easily overwhelm even the most powerful state-of-the-art graphics accelerators, leading to poor interactive rates. This problem is further compounded when a desktop machine is used to render the isosurface while a remote server, with potentially more memory, is used to extract the isosurface. This can occur over either a LAN or a WAN.

One current approach to the large number of polygons problem is to apply mesh reduction techniques [18, 14] to the isosurface either as a post process to the extraction phase or during the extracting phase itself [17]. However, mesh reduction is expensive and requires extracting the entire isosurface for examination. Furthermore, a change in the isovalue requires the *full* extraction of a new isosurface and the reapplication of the mesh reduction step.

A different approach is to employ ray-tracing techniques that do not need to create an intermediate polygonal representation. Ray-tracing, nevertheless, does not take advantage of graphics hardware and requires a large number of CPUs for interactivity [16].

In this paper we present a novel view dependent isosurface extraction approach, as illustrated in Figure 1, which further reduces the search, construction and display by only visiting the cells that contain the *visible* portion of the isosurface from a given view point. Our approach is based on a hierarchical front-to-back traversal of the dataset with dynamic pruning of sections that are hidden from the view point by previously extracted sections of the isosurface. Figure 2 shows the potential saving of such an approach. Note the large section of the isosurface, which represents the internal organs in the head, yet is not part of the view-dependent isosurface.

This work explores the middle ground between

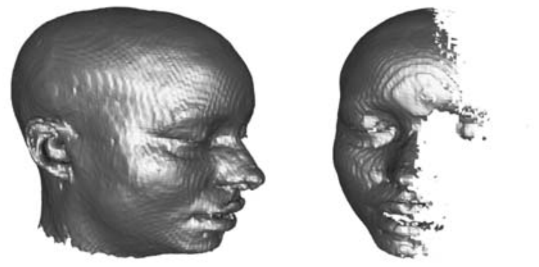

Figure 1: Left: The user view, Right: The same isosurface from a 90 degree angle to the user view, illustrating the incomplete reconstruction.

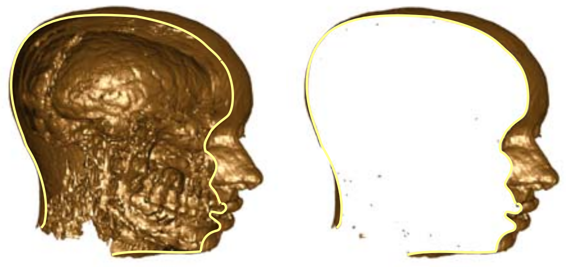

Figure 2: Extracted isosurface. A cut plane through the full and view-dependent isosurfaces extracted from the same view point as in Figure 1. Note the large internal structures that are part of the full isosurface but are not part of the view-dependent isosurface

mostly hardware based (e.g. marching cubes + Z-buffer) and purely software (e.g. ray-tracing) algorithm for isosurface extraction. Our goal is to reduce the load on the network and/or graphics hardware by performing *some* of the visibility tests in software. This approach leads to an output sensitive method that can reduce the load of other components in the visualization pipeline such as transmission of the isosurface geometry over a network.

In the next section we review previous approaches to isosurface extraction. In Section 3 we present our algorithm, drawing on recent innovations in the area of visibility algorithms. Sections 4–5 detail these algorithms as well as our modifications. Results are presented in Section 6. We conclude and discuss future directions in Section 7.

2 Related Work

Marching cubes [11, 23] is perhaps the most widely known 3D isosurface extraction algorithm. The novelty of this approach was the reduction of the global extraction problem to a large set of local triangulation (one per data cell) based on a lookup table. Since the introduction of the marching cubes method, much work [12, 13] was dedicated to the ambiguities introduced by the lookup table and other methods of representing the isosurface inside a single cell.

Another issue that has received attention was the localization of the cells that actually contained the isosurface. Wilhelms and Van Gelder [21, 22] were the first to point out the large amount of time that was spent analyzing empty cells. To alleviate this problem, they introduced an octree based hierarchy over the dataset and tagged each node (a *meta-cell*) in the tree with the minimum and maximum values of the data in their subtree. Given an isovalue, one can then descend the hierarchy, pruning empty subtrees based on this minmax scheme.

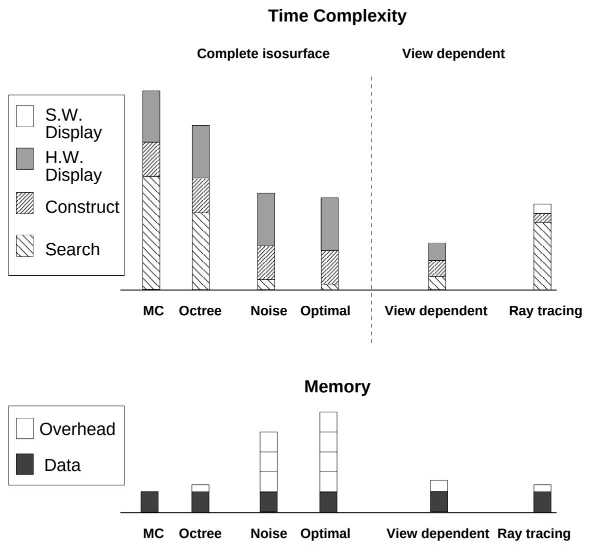

Figure 3: Complexity Comparison.

In recent years, many other methods for accelerating the search phase were proposed [3, 4, 6, 7, 15, 20, 19] all of which have a complexity of $O(n)$, where n is the size of the dataset. In 1996, we introduced the *span space* [10] as a mean for mapping the search onto a two-dimension space. Using the span space, we proposed a *near optimal isosurface extraction* (NOISE) algorithm that has a time complexity of $O(\sqrt{n} + k)$, where k is the size of the isosurface. Cignoni *et. al* [1] employed another decomposition of the span space leading to an optimal search method with a time complexity of $O(\log n + k)$. Figure 3 depicts a relative (not to scale) comparison between these methods with respect to the search, construction and display phases.

3 The Algorithm

The proposed method is based on the observation that isosurfaces extracted from very large datasets tend to exhibit high depth complexity for to two reasons. First, since the datasets are very large, the projection of individual cells tend to be sub-pixel. This leads to a large number of polygons, possibly non-overlapping, projecting onto individual pixels. Secondly, for some datasets, large sections of an isosurface are internal and thus, are occluded by other sections of the isosurface, as illustrated in Figure 2. These internal sections, common in medical datasets, can not be seen from any direction unless the external isosurface is peeled away or cut off. Therefore, if one can extract just the visible portions of the isosurface, the number of rendered polygons will be reduced resulting in a faster algorithm. Figure 4 depicts a two dimensional scenario. In a view dependent methods only the solid lines are extracted whereas in non view dependent isocontouring both solid and dotted are extracted.

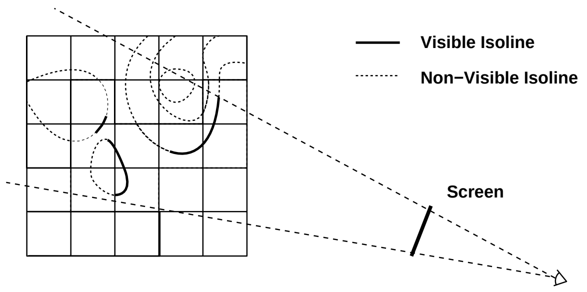

Figure 4: A two dimensional scenario.

The proposed algorithm, which is based on a hierarchical traversal of the data and a marching cubes triangulation, exploit coherency in the object, value, and image spaces, as well as balancing the work between the hardware and the software. We employ a three step approach depicted in Figure 5. First, we augment Wilhelms and Van Gelder's algorithm by traversing down the octree in a front-to-back order in addition to pruning empty sub-trees based on the min-max values stored at the octree nodes. The second step employs coarse software visibility tests for each [meta-] cell which intersect the isosurface. The aim of these tests is to determine whether the [meta-] cell is hidden from the view point by previously extracted sections of the isosurface (thus the requirement for a front-to-back traversal). Finally, the triangulation of the visible cells are forwarded to the graphics accelerator for rendering by the hardware. It is at this stage that the final and exact [partial-] visibility of the triangles is resolved. A dataflow diagram is depicted in Figure 6.

3.1 Visibility

Quickly determining whether a meta-cell is hidden and thus can be pruned, is fundamental to this algorithm. This is implemented by creating a virtual screen with one bit per pixel. We then project the triangles, as they are extracted, on to this screen and set those bits which are covered, providing an occlusion mask.

Additional pruning of the octree nodes is accomplished by projecting the meta-cell on to the virtual screen and checking if any part of it is visible, i.e. if any of the pixels it covers are not set. If the entire projection of the meta-cell is not visible, none of its children can be visible.

We note that it is important to quickly and efficiently classify a cell as visible. A hidden cell, and all of its children, will not be traversed further and thus can justify the time and effort invested in the classification. A visible cell, on the other hand, does not gain any benefit from this test and the cost of the visibility test is added to the total cost of extracting the isosurface. As such, the cell visibility test should not depend heavily on the projected screen area otherwise the cost would prohibit the use of the test for meta-cells at high levels of the oc-

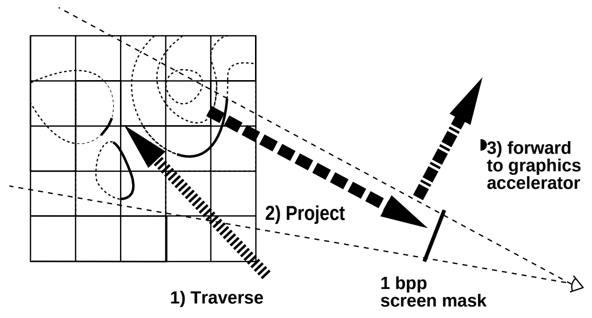

Figure 5: The three step algorithm.

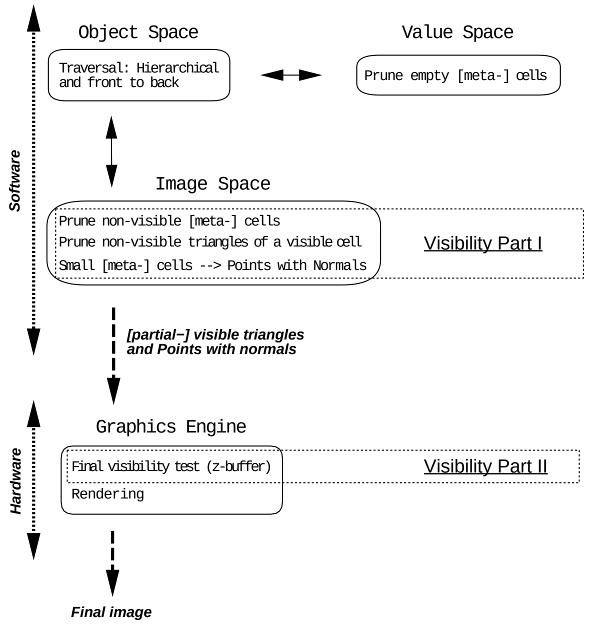
Object Space
Traversal: Hierarchical and front to back
Value Space
Prune empty [meta-] cells
Software
Image Space
Prune non-visible [meta-] cells
Prune non-visible triangles of a visible cell
Small [meta-] cells --> Points with Normals
Visibility Part I
[partial−] visible triangles and Points with normals
Graphics Engine
Final visibility test (z-buffer)
Rendering
Visibility Part II
Hardware
Final image

Figure 6: The algorithm data flow.

tree - exactly those meta-cells that can potentially save the most.

Two components influence the visibility cost, namely the cost of projecting a point, triangle, or a meta-cell on to the screen and the cost of either scan-converting triangles or determining if a meta-cell projected area contains any unset pixels.

In the next sections, we address these costs in two ways. First, we employ a hierarchical tiling for the virtual screen. Secondly, to reduce the cost of the projection we use a variation of the shear-warp factorization.

4 Image Space Culling

We employ hierarchical tiles [5] as a mean for fast classification of meta-cells and determining the coverage of extracted triangles. The hierarchical nature of the algorithm ensures that the cost of either of these two operations will not depend highly on their projected area.

4.1 Hierarchical Tiles

Hierarchical tiles provide a mechanism for accelerating software based rendering. The idea is to project one polygon at a time in a front-to-back order and render only those pixels that contribute to the final image. The front-to-back order ensures that each pixel is rendered only once.

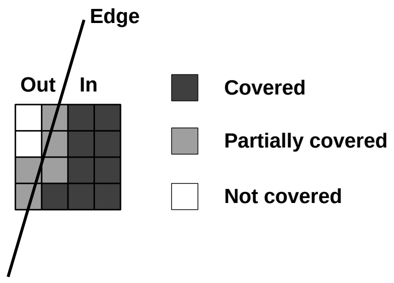

Figure 7: An Edge Tile.

A coverage map (a tile) is a rectangle bitmap (we use 8x8) in which each bit represents a pixel in the final image. The algorithms is based on the premise that all the possible coverage of a single edge crossing a tile can be precomputed and tabulated based on the points where the edge intersect the tile boarder, Figure 7. The coverage pattern of a convex polygon for a particular tile of the image is computed by combining the coverage maps of the polygon edges. The coverage map of a triangle can thus be computed from three precomputed tiles with no dependency on the number of pixels the triangle actually cover, Figure 8. We refer the reader to the work by Greene [5] for a detailed explanation on how the three states (Covered, Partially-covered and Not-covered) can be represented by two tile masks and the rules for combining coverage maps.

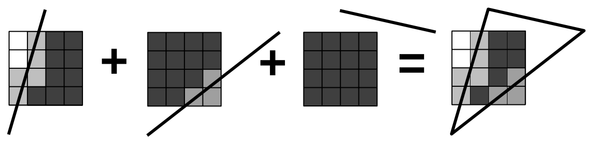

Figure 8: A triangle tile coverage map.

Rendering a polygon amounts to computing the coverage map of the polygon for each tile in the image and isolating only those pixels that are covered by the polygon and where already covered, Figure 9. In order to accelerate the rendering, the tiles are organized in a hierarchical structure in which each meta-tile represents a block of [meta-] tiles. Under this structure, a polygon is projected onto the top meta-tile and only those subtiles in which the polygon might be visible are checked recursively, leading to a logarithmic search.

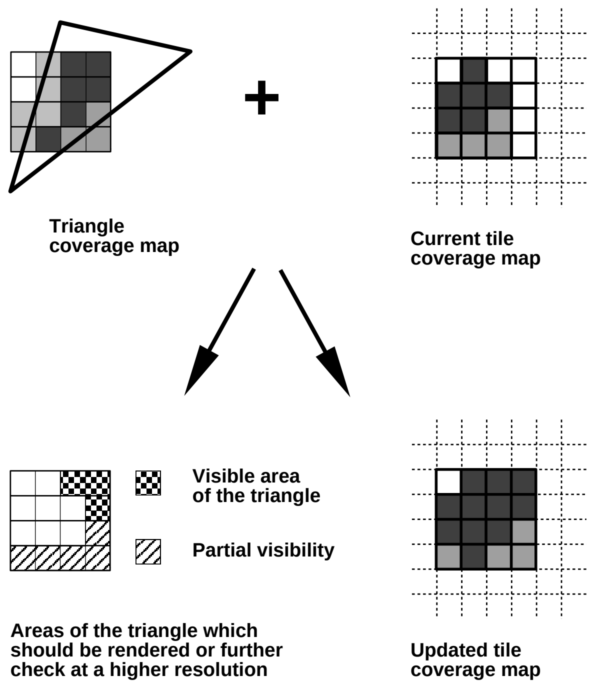
Triangle coverage map
+
Current tile coverage map
Visible area of the triangle
Partial visibility
Areas of the triangle which should be rendered or further check at a higher resolution
Updated tile coverage map

Figure 9: Application of a triangle coverage map and an image tile coverage map.

4.2 Hierarchical Visibility Mask

Our implementation differs from the one proposed by Greene in that we do not actually render the visible portion of a projected triangle. Rather, we mark the triangle as visible and forward it to the graphics hardware. It is then left to the graphics accelerator to determine which pieces of the triangle are actually visible and correctly render them.

One should note that it is not possible to determine *a-priori* the front-to-back relations between the triangles inside a single cell. It is therefore mandatory to accept all or none of the triangles, even though they need to be projected on the hierarchical tiles one triangle at a time. Figure 10 shows the classification of the cells as well as the portions of the isolines which are extracted. Note that the entire isoline section in a visible cell (shown in light gray) is extracted. The non-visible portions will be later removed by the graphics accelerator.

An additional feature we employ limits recursion down the octree once the size of a meta-cell is approximately the size of a single pixel. Instead, we forward a single point with an associated normal to the graphics hardware, similar to the dividing cubes method [2]. The normal is estimated by the gradient of the field.

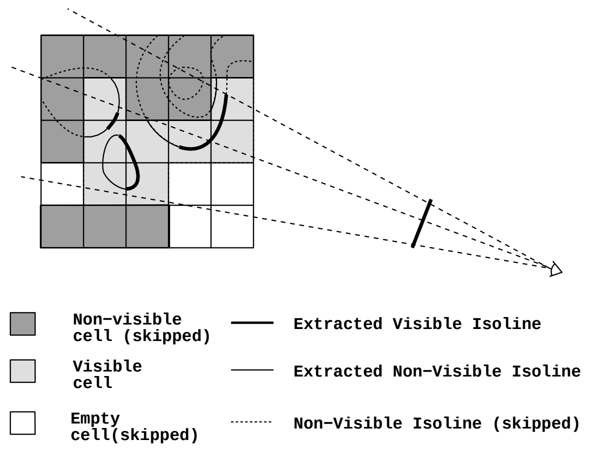
Non−visible cell (skipped)
Visible cell
Empty cell(skipped)
Extracted Visible Isoline
Extracted Non−Visible Isoline
Non−Visible Isoline (skipped)

Figure 10: Cells and isolines visibility.

The advantage of this method is that the single point potentially represents a large number of polygons since the meta-cell that projects to a pixel may still be high in the octree.

5 Warped IsoSurface Extraction *(WISE)*

A key component in the visibility test is the projection of a point, a triangle or a meta-cell onto the screen. In general, the perspective projection of a point is a 4x4 transformation followed by two divide operations, for a total of 16 multiplications 12 additions and 2 division per vertex. Clearly, the cost of performing such transformation for each and every vertex of the projected meta-cells and triangles is too high. In addition, the non-linearity of the perspective transformation prohibits the use of pre-computed transformation table. To accelerate this critical step, we take advantage of the shear-warp factorization of the viewing transformation.

5.1 Shear-Warp Factorization

In 1994, Lacroute [8, 9] presented a volume rendering method that was based on the shear-warp factorization of the viewing transformation. The underlyng idea is to factor the viewing transformation into a shear followed by a warp transformation. The data is first projected into a sheared object space that is used to create an intermediate, albeit warped, image. Once this image is complete a warping transformation is applied to create the correct final image. Figure 11 illustrates the shear-warp transformation for an orthographic projection. For a perspective projection the slices need also to be scaled as seen in Figure 12.

The advantage of this method is that the intermediate image is aligned with one of the dataset faces. This alignment enables the use of a parallel projection of the 3D dataset. The warp stage is then applied to a 2D image rather than to each data point.

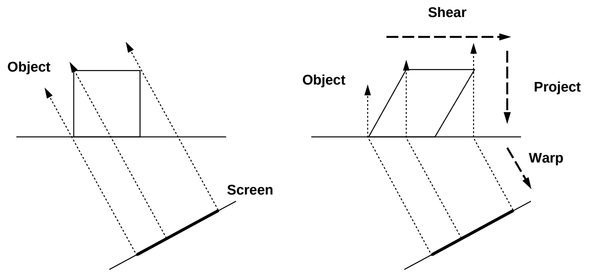

Figure 11: Shear-warp in orthographic projection.

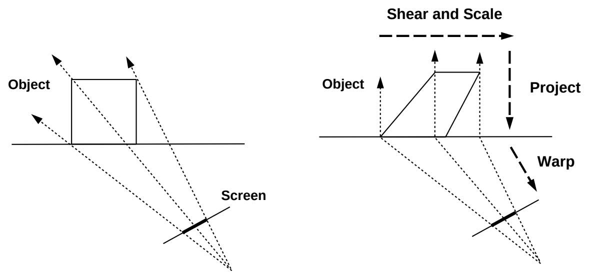

Figure 12: Shear-warp in perspective projection.

5.2 Shear But No Warp

We now note that the visibility on the image plane and on the warped projection plane are the same, see Figure 13. In other words, any point in the dataset that is visible on the image plane is also visible on the warped projection plane and similarly, points which would be occluded on the image plane are also occluded on the warped plane. It is therefore sufficient to perform the visibility tests on the warped projection plane. The advantage of this approach is two fold. First, the perspective projection is removed. Second, since the shear and scale factors are, with respect to the current view point, constant for each slice we can pre-compute them once for each new view point.

Let $[X, Y, Z]$ be the coordinate system of the dataset and let $[s_x, s_y, s_z]$ be the scaling vector of the data

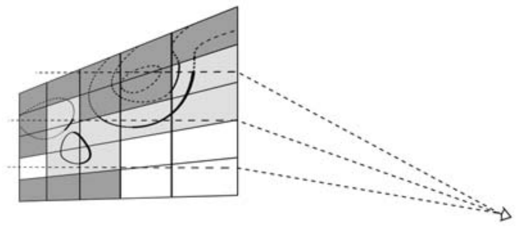

Figure 13: Warped space.

with respect to this coordinate system. Let us assume, without loss of generality, that the current warped projection plane is $Z = 0$. We first transform the current eye location onto the $[X, Y, Z]$ coordinate system and then pre-compute the shear and scale coefficients,

```
foreach $Z$
    $s = Z * s_z / (Z * s_z - eye_z)$
    $scale_x[Z] = (1 - s) * s_x$
    $scale_y[Z] = (1 - s) * s_y$
    $shear_x[Z] = s * eye_x$
    $shear_y[Z] = s * eye_y$
```

The projection of any grid point $p(x, y, z)$ can now be computed as,

```
Project$(p)$ $\equiv$
    $x = p_x * scale_x[p_z] + shear_x[p_z]$
    $y = p_y * scale_y[p_z] + shear_y[p_z]$
```

for a total of two multiplications and two additions per vertex.

While the Z coordinate of every grid point is known in advance and thus the shear and scale factor can be pre-computed for each new view point, the same does not hold true for the vertices of the isosurface triangles. However, since the projection onto the warped projection plane is orthographic it can be shown that a vertex projection is,

```
Project$(p)$ $\equiv$
    $s = p_z / (z - eye_z)$
    $x = p_x + s * (eye_x - p_x)$
    $y = p_y + s * (eye_y - p_y)$
```

for a total of two multiplication, five additions and one division.

6 Results

We tested our method with two datasets. The first is a small dataset (64^3) of a CT scanned head where most of the internal structure, e.g. the brain, was removed by hand segmentation. This results in lower depth complexity. The other larger dataset (256^3) has a large number of internal structures that normally would be extracted as isosurfaces when one extracts the skin. We ran experiments in two scenarios. In one scenario, the isosurfaces were extracted and rendered on the same high end machine, an SGI Reality Monster (using a single CPU). In the second scenario, we use a lower-end machine (SGI Indigo2 Extreme connected with a 100-BaseT switched Ethernet) for the rendering phase and the Reality Monster for the isosurface extraction.

The results for the first scenario, Table 1, shows that for larger and more complex isosurfaces the new method still outperforms extraction of the full isosurface, though the performance falls short when only a change of the view point is considered.

The results for the second scenario, Table 2, demonstrate the advantage of this method. Even when the time for the extraction is added to each new view position, the new method out performs a regular full octree traversal and extraction due to the LAN bandwidth limitations.

It is important to note the large reduction in the size, up to a factor of 15, of the extracted isosurface.

Table 1: Scenario I: Local Visualization

method	view	extr. time	polygons	points	rend. time
Small dataset					
Octree	any	0.48	46,222		0.16
VD	normal	0.79	9036		0.06
VD	closeup	0.59	7995		0.02
VD	zoom out	0.60	5938	1,112	0.02
Large dataset					
Octree	any	3.86	353,868		1.28
VD	normal	2.56	22,405		0.04
VD	closeup	0.85	5,588		0.03
VD	zoom out	0.99	1,080	7,888	0.02

7 Conclusions

We have developed a new approach to isosurface extraction which relies on extracting only the visible portion of the isosurface. It was demonstrated that the reduction in the isosurface size can be substantial (up to 93%) which makes it attractive for remote visualization.

Table 2: Scenario II: Remote Visualization

method	view	extr. time	polygons	points	rend. time
Small dataset					
Octree	any	0.42	46,222		1.35
VD	normal	0.79	9184		0.27
VD	closeup	0.31		2735	0.05
VD	zoomout	0.40	2154	2319	0.11
Large dataset					
Octree	any	3.57	342,640		10.58
VD	normal	2.31	20,330		0.60
VD	closeup	1.14	6,078	5,374	0.30
VD	zoomout	0.38		7,364	0.12

We note that in some cases the algorithm did not perform as well. These are cases in which the depth complexity is not high and the size of the full isosurface is within the capability of the graphics adaptor.

In the future, we plan on optimizing and parallelizing our code as well as applying it to much larger datasets such as the visible human project and geological seismic survey. We are currently looking at other projection methods to accelerate the sofware visibility tests. For fater view changes, we intend to employ image based rendering techniques on the user side to enable warping between several distinct view dependent isosurfaces with the same isovalue.

8 Acknowledgments

This work was supported in part by the National Science Foundation. Special thanks to Chris R. Johnson for his helpful comments and suggestions. The authors would like to thank the reviewers for their insightful comments. The head dataset was provided by the Los Alamos National Laboratory. Furthermore, we appreciate access to facilities that are part of the NSF STC for Computer Graphics and Scientific Visualization, the C-SAFE DOE ASCI Alliance Center and the DOE Advanced Visualization Technology Center.

References

[1] P. Cignoni, C. Montani, E. Puppo, and R. Scopigno. Optimal isosurface extraction from irregular volume data. In *Proceedings of IEEE 1996 Symposium on Volume Visualization*. ACM Press, 1996.

[2] Harvey E. Cline, William E. Lorenson, Sigwalt Ludke, Carl R. Crawford, and Bruce C. Teeter. Two algorithms for the three-dimensional reconstruction of tomograms. *Medical Physics*, 15(3):320–327, May 1988.

[3] R. S. Gallagher. Span filter: An optimization scheme for volume visualization of large finite element models. In *Proceedings of Visualization '91*, pages 68–75. IEEE Computer Society Press, Los Alamitos, CA, 1991.

[4] M. Giles and R. Haimes. Advanced interactive visualization for CFD. *Computing Systems in Engineering*, 1(1):51–62, 1990.

[5] Ned Greene. Hierarchical polygon tiling with coverage masks. In *Computer Graphics*, Annual Conference Series, pages 65–74, August 1996.

[6] T. Itoh and K. Koyamada. Isosurface generation by using extrema graphs. In *Visualization '94*, pages 77–83. IEEE Computer Society Press, Los Alamitos, CA, 1994.

[7] T. Itoh, Y. Yamaguchi, and K. Koyyamada. Volume thining for automatic isosurface propagation. In *Visualization '96*, pages 303–310. IEEE Computer Society Press, Los Alamitos, CA, 1996.

[8] Philippe Lacroute and Marc Levoy. Fast volume rendeing using a shear-warp factorization of the viewing transformation. In *ACM SIGGRAPH 94*, Annual Conference Series, pages 451–458, July 24-29 1994.

[9] Philippe G. Lacroute. Fast volume rendering using shear-warp factorization of the viewing transformation. Technical report, Stanford university, September 1995.

[10] Y Livnat, H. Shen, and C. R. Johnson. A near optimal isosurface extraction algorithm using the span space. *IEEE Trans. Vis. Comp. Graphics*, 2(1):73–84, 1996.

[11] W.E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. *Computer Graphics*, 21(4):163–169, July 1987.

[12] Sergey V. Matveyev. Approximation of isosurface in the marching cubes: Ambiguity problem. In *Visualization '94*, pages 288–292, October 1994.

[13] C. Montani, R. Scateni, and R. Scopingo. Discretized marching cubes. In *Visualization '94, Los Alamitos, CA*, pages 281–287, October 1994.

[14] Kwang-Man Oh and Kyu Ho Park. A type-merging algorithm for extracting an isosurface from volumetric data. *The Visual Computer*, 12:406–419, 1996.

[15] J.S. Painter, P. Bunge, and Y. Livnat. Case study: Mantle convection visualization on the cray T3D. In *Visualization '96*, pages 409–412. IEEE Computer Society Press, Los Alamitos, CA, 1996.

[16] Steven Parker, Peter Shirley, Yarden Livnat, Chuck Hansen, and Peter-Pike Sloan. Interactive ray tracing for isosurface rendering. In *Visualization '98*, 1998.

[17] Tim Poston, H.T. Nguyen, Pheng-Ann Heng, and Tien-Tsin Wong. 'skeleton climbing':fast isosurfaces with fewer triangles. In *Pacific Graphics'97*, pages 117–126, Seoul, Korea, October 1997.

[18] Raj Shekhar, Elias Fayyad, Roni Yagel, and J. Fredric Cornhill. Octree-based decimation of marching cubes surfaces. In *Proceedings of Visualization '96*, pages 335–342. IEEE Computer Society Press, 1996.

[19] H Shen, C. D. Hansen, Y Livnat, and C. R. Johnson. Isosurfacing in span space with utmost efficiency (issue). In *Proceedings of Visualization '96*, pages 287–294. IEEE Computer Society Press, 1996.

[20] H. Shen and C. R. Johnson. Sweeping simplicies: A fast iso-surface extraction algorithm for unstructured grids. *Proceedings of Visualization '95*, pages 143–150, 1995.

[21] J. Wilhelms and A. Van Gelder. Octrees for faster isosurface generation. *Computer Graphics*, 24(5):57–62, November 1990.

[22] J. Wilhelms and A. Van Gelder. Octrees for faster isosurface generation. *ACM Transactions on Graphics*, 11(3):201–227, July 1992.

[23] Geoff Wyvill, Craig McPheeters, and Brian Wyvill. Data structure for soft objects. *The Visual Computer*, 2:227–234, 1986.