

Published in final edited form as:

*Comput Graph.* 2011 June ; 35(3): 561–568. doi:10.1016/j.cag.2011.03.035.

## View-independent Contour Culling of 3D Density Maps for Far-field Viewing of Iso-surfaces

Powei Feng<sup>a</sup>, Tao Ju<sup>b</sup>, and Joe Warren<sup>a</sup>

Powei Feng: pfeng@rice.edu; Tao Ju: taoju@cse.wustl.edu; Joe Warren: jwarren@rice.edu

<sup>a</sup>Department of Computer Science, Rice University, Houston, Texas, USA

<sup>b</sup>Department of Computer Science and Engineering, Washington University in St. Louis, St. Louis, Missouri, USA

### Abstract

In many applications, iso-surface is the primary method for visualizing the structure of 3D density maps. We consider a common scenario where the user views the iso-surfaces from a distance and varies the level associated with the iso-surface as well as the view direction to gain a sense of the general 3D structure of the density map. For many types of density data, the iso-surfaces associated with a particular threshold may be nested and never visible during this type of viewing. In this paper, we discuss a simple, conservative culling method that avoids the generation of interior portions of iso-surfaces at the contouring stage. Unlike existing methods that perform culling based on the current view direction, our culling is performed once for all views and requires no additional computation as the view changes. By pre-computing a single visibility map, culling is done at any iso-value with little overhead in contouring. We demonstrate the effectiveness of the algorithm on a range of bio-medical data and discuss a practical application in online visualization.

## 1. Introduction

### 1.1. Motivation

Iso-surfaces are commonly used for visualizing 3D density maps, such as MRI and CT scans in bio-medical applications. With the increasing complexity of today's imaging data, contouring a density map easily yields iso-surfaces with high polygon counts that are costly to produce, store and render. Often times, however, the number of elements contained in the iso-surface does not correspond proportionally to visual complexity perceived by the viewer. Consider the iso-surface in Figure 1 (a) from the CT scan of a human foot. There are many interior surface pieces, which are highlighted in Figure 1 (b). These interior parts account for close to half of the total triangles in the iso-surface, and yet they are not visible if one views the foot from the outside.

One method for handling this issue is to cull invisible components of the iso-surfaces at rendering time using existing *visibility culling* methods [2]. While this approach is feasible, the extraction and storage of the original, un-culled iso-surface is still required. A better solution would be to perform culling at the contouring stage, thus avoiding the generation of invisible surface components in the first place. We call this second approach *contour culling*. While

© 2011 Elsevier Ltd. All rights reserved.

**Publisher's Disclaimer:** This is a PDF file of an unedited manuscript that has been accepted for publication. As a service to our customers we are providing this early version of the manuscript. The manuscript will undergo copyediting, typesetting, and review of the resulting proof before it is published in its final citable form. Please note that during the production process errors may be discovered which could affect the content, and all legal disclaimers that apply to the journal pertain.

there have been a number such methods proposed, these methods are all *view-dependent* (see a brief review in the next section). This means that culling has to be performed whenever the user changes the viewing angle. As a result, a significant overhead can be added to the run-time rendering pipeline.

## 1.2. Contribution

In this paper, we explore an alternative, *view-independent* contour culling approach that adds little computational overhead at run-time. We consider a specific but common scenario of iso-surfaces visualization, which we call *far-field viewing*. In this scenario, the user views the iso-surfaces from a distance (e.g., the view taken in Figure 1 (a)) and varies the iso-value associated with the iso-surface as well as the viewpoint and viewing direction to gain a sense of the general 3D structure of the density map. At each iso-value, our method culls parts of the iso-surface invisible to *any* viewpoints located outside the volume (e.g., highlighted parts in Figure 1 (b)). Once the iso-surface is generated, it can be viewed from different directions with no more computations needed other than rendering.

There has been active research into visibility culling methods that cull based on views from a region rather than from a single point (see a brief survey in [15]). However, these methods cannot be directly applied to iso-surface visualization as they usually require considerable pre-computation that is specific to the surface to be culled. When the user changes the iso-value, such pre-computation needs to be performed again.

In contrast, our method performs a single pre-processing step for iso-surfaces at *all* iso-values. The key observation is that values in a density map (especially bio-medical images) typically drop off at the boundary of the map. During far-field viewing of such a map, a point  $x$  in the map that is completely invisible when viewing from outside the map at some iso-value  $c$  will also stay invisible for iso-values lower than  $c$ . We call the minimal iso-value that  $x$  is visible to some outside views the *contour visibility function* (CVF), or  $g_f(x)$  (detailed in Section 2). With a pre-computed CVF, the visible parts of the iso-surface at any iso-value  $c$  can be easily extracted by contouring only in parts of the map where  $g_f(x) \leq c$ .

Our main contribution is a simple dynamic-programming algorithm for computing a discrete approximation of CVF for tri-linearly interpolated 3D density functions (see Section 3). We show that this piece-wise constant approximation can be easily utilized by a tri-linear contouring algorithm or typical polygonal contouring algorithms (such as Marching Cubes [11]) to perform culling, and the result is guaranteed to be *conservative*: the culled portions of the iso-surface are not to the viewer at any view angle for far-field viewing (see Section 4).

An example result of our method is shown in Figure 1 (c). Note that the visible parts of the iso-surface are well preserved, whereas a significant amount of interior pieces are removed (see (d)). When tested on several real-world bio-medical data (see Section 5), we observed that our view-independent culling approach performs well for density maps that contain large inner iso-surface pieces, sometimes achieving up to 80% surface reduction.

## 1.3. Application

The reduced run-time overhead of our view-independent culling approach applies naturally to online and mobile platforms with limited access to computational and rendering resources. As a concrete example, we incorporated the culling algorithm into an online visualization applet for 3D density maps of macromolecular structures (such as viruses) as part of the Electron Microscopy Databank (EMDB). The iso-surfaces of such data often contain large nested, interior portions, and the biologists most often take views from outside of the volume. Hence our culling method is particularly suited. In this application, both contouring (using Marching

Cubes) and culling are performed on the server side, and the resulting surface is transformed to the client-side applet for viewing. Due to bandwidth limits, the iso-surface needs to be simplified before transmission [6]. With culling turned on, simplification becomes more efficient (since there are fewer triangles to start with), and the simplified result better preserves the details on the visible parts of the iso-surface. This is demonstrated in Figure 2. Note that since our culling is view-independent, no extra computation is needed on the client-side applet as the user inspects the surface from different views.

#### 1.4. Related work

Visibility culling is a well-studied problem in computer graphics. Previous work on visibility culling has focused on the problem of culling the invisible portions of a polygonal mesh with respect to a view direction. Classic techniques that try to address the visibility culling problem include Z-buffer, back-face culling, view-frustum culling, and visibility space partitions (see the survey [2]). Our approach differs from traditional approach in that we focus on avoiding the generation of unnecessary polygons when extracting level-sets from a density map. Most of the previous methods are compatible with our method since they can be applied after a polygonal mesh has been extracted from the density map.

Several algorithms for culling iso-surfaces based on visibility have been proposed in the past. Given an iso-value, these algorithms generate only part of the iso-surfaces visible to the viewer. Livnat and Hansen [10] perform a front-to-back sweep of an octree representation of the density volume to quickly identify regions of the space occluded by the iso-surface. In a similar approach, Gao and Shen [4] improve the culling performance on multiple processors using an image space culling algorithm and a load-balanced workflow. Pesco et al. [16] uses an implicit culling scheme based on the scalar values at the voxels instead of actual triangles on the iso-surface. Recently, Gregorski et al. proposes a culling algorithm for iso-surfaces extracted from hierarchical tetrahedral meshes that exploits frame-to-frame coherence between consecutive views [7].

While these methods can dramatically reduce the number of polygons that need to be rendered, their view-dependence requires culling and surface extraction to be performed whenever the view is changed. In addition, culling in these methods can add significant overhead (in both implementation complexity and running time) to the contouring algorithm and is specific to the iso-value used to generate the iso-surface. In contrast, our approach does not require re-generation of iso-surfaces as the view changes, and our method adds little overhead to the contouring algorithm regardless of the iso-value used.

Our work is closely related to methods that compute occlusion maps to accelerate rendering. One of the earliest works is by Zhang et al., who propose computing a hierarchy of occlusion maps to accelerate the rendering of scenes with large number of primitives [17]. Our work differs from Zhang et al's work in that our method targets density maps. In particular, our visibility function can be viewed as an occlusion map, but it performs culling for all iso-values. Li et al. extended the idea of occlusion maps into the domain of volume rendering. In their method, the occlusion map is updated at every frame (i.e. view dependent). In contrast, we compute a single, conservative occlusion map for all-directions [9].

Also related to contour culling is the class of methods that cull occluded voxels for volume rendering [3, 5]. Like iso-surface culling, most of these methods are view-dependent. A notable exception, and one that is most related to our work, is the works by Mroz and colleagues [13, 12] that perform culling for Maximum Intensity Projection (MIP), a rendering method that displays the maximum intensity along each ray through the volume. Similar to us, the authors observe that a (possibly large) fraction of the voxels do not contribute to the MIP image at any viewing angle. Hence they identify such voxels in a pre-processing stage and preclude

them in the rendering stage. The identification is done either by recursive ray search at each voxel [13], or by a front-moving algorithm [12] that is similar in spirit to the dynamic programming algorithm proposed in this paper.

There are a number of key differences between our work and that of Mroz and colleagues. First, the two works are designed for different visualization modes (MIP versus iso-surfaces), hence the culling criteria is different. Second, while MIP culling creates a binary mask (kept or removed) for the voxels, our approach builds a floating-point visibility function over the volume that provides the culling information at all iso-values. Third, comparing to front-moving in [12], our dynamic programming algorithm offers the additional control over the aggressiveness of surface reduction while maintaining a conservative culling.

## 2. The contour visibility function

We proceed to provide a mathematical basis for our technique. We will first formulate the contour visibility function (CVF), which gives the exact range of level sets at which a point in space is visible when viewed at infinity. We will then consider approximations of such functions with conservative guarantees, which allow efficient computations (to be discussed in Section 3).

### 2.1. The definition

Defining the CVF of a given density function  $f(x)$  is fairly straightforward. Assuming again that the values of  $f(x)$  lie in the range  $[0, 1]$  and that tend towards zero as  $x \rightarrow \infty$ . Let  $r_x$  be a ray from  $x$  to infinity and  $f_{r_x}$  be the maximum of  $f$  taken at all points (including  $x$ ) along the ray  $r_x$  (i.e.  $\max_{z \in r_x} f(z)$ ). We define the CVF, denoted as  $g_f(x)$ , to be the minimum of  $f_{r_x}$  over all possible rays  $r_x$ .

We now claim that given some iso-surface  $f(x) = c$  where  $0 < c < 1$ , a point  $x$  (which may or may not be on the iso-surface) lies in the visible space when viewed from infinity in some direction if and only if  $g_f(x) \leq c$ . To prove this, note that a ray intersects with the iso-surface if and only if there are some values along the ray above  $c$  and some below  $c$  (assuming the intersection is in a general position and that the iso-surface avoids the singularity of  $f$ ). Since  $f$  tends to zero at infinity, a ray  $r_x$  intersects with the iso-surface if and only if  $f_{r_x} > c$ . As a result,  $x$  is un-blocked by the iso-surface in some direction if and only if  $f_{r_x} \leq c$  for some  $r_x$ , and likewise  $g_f(x) \leq c$ .

Figure 3 illustrates the CVF of a 1D function (shown in top-left). In 1D, there are only two possible rays associated with each point  $x$  that go toward either the positive or negative infinity, which we denote respectively as  $r_x^+$  and  $r_x^-$ . The ray maxima  $f_{r_x^+}$ ,  $f_{r_x^-}$  are plotted respectively in top-right and bottom-left for each  $x$ , and the CVF  $g_f(x)$ , the minimum of the two ray maxima, is plotted in bottom-right. Observe that, by our definition,  $f(x) \leq g_f(x)$ .

### 2.2. Conservative approximations

While computing CVF is relatively straightforward in 1D, it is much more difficult in 2D and 3D. Even in the case where  $f(x)$  has a simple structure, such as being piece-wise constant, the function  $g_f(x)$  is likely to have a much more complex structure that is challenging to compute both accurately and efficiently.

From the computational perspective, we are mostly interested in approximations of the exact CVF. In particular, we are interested in approximations  $g(x)$  that are *conservative*: given any iso-surface  $f(x) = c$ , a point  $x$  such that  $g(x) > c$  should never be visible in any direction when viewed from infinity. In this way, culling away all points  $x$  on the iso-surface where  $g(x) > c$

will not affect the visible part of the surface in any views (even though some hidden, interior parts may still remain).

It is easy to show that an approximation  $g(x)$  is conservative if  $0 < g(x) \leq g_f(x)$  for any  $x$ . The closer  $g(x)$  approaches  $g_f(x)$ , the tighter the approximation, and larger portions of the iso-surfaces could be culled using  $g(x)$ . Note that  $f(x)$  itself is a conservative approximation, as  $f(x) \leq g_f(x)$ . Intuitively,  $f(x) > c$  meaning that  $x$  lies in the “inside” of the iso-surface  $f(x) = c$ , hence invisible to the outside.

### 3. Computing the approximated CVF

In many contouring applications, the density map is provided as scalar values associated with points on a 3D integer grid. These values are then interpolated to form  $f(x)$ , typically using tri-linear interpolation. Our goal in this section is to develop an efficient algorithm that computes a conservative approximation of the CVF in a tri-linearly interpolated density function.

#### 3.1. Motivation

The algorithm is motivated by the piece-wise nature of tri-linear interpolation. The interpolated function  $f(x)$  is made up of smooth pieces within each grid “cell” formed by eight grid points. Likewise, tri-linear contouring algorithms are typically performed in a cell-by-cell manner, producing one piece within a cell. If we know which cells are invisible, we can easily modify the contouring algorithm to skip the invisible cells. To make such decisions quickly and accurately, all we need is a conservative approximation to  $g(x)$  that assumes a *constant* value in each cell. If this value is greater than the current iso-value, the conservativeness guarantees that the entire cell is never visible in any view directions.

For ease of explanation, we will start by describing the algorithm for computing this piece-wise constant approximation of CVF in 2D. We will then show how the implementation can be extended to work on 3D volumes.

#### 3.2. Algorithm in 2D

Consider the problem of approximating the CVF for a 2D density function  $f(x)$  over an integer grid with  $n \times n$  squares such that the function gives a constant value  $g_{i,j}$  for each grid square  $p_{i,j}$ . To be a conservative approximation, we ask  $g_{i,j} \leq g_f(x)$  for any  $x \in p_{i,j}$  where  $g_f(x)$  is the actual CVF.

We start by examining the discrete path of grid squares, denoted as  $r_{i,j}$ , that is intersected by the ray  $r_x$  from some point  $x \in p_{i,j}$  to infinity (see the shaded squares in Figure 4 (a)). Observe that consecutive squares in the path share common edges (highlighted in the picture) that are intersected by the ray. Now, consider the minimum of  $f$  along each of these common edges, and let  $f_{r_{i,j}}$  be the maximum of such minima over all edges along the path  $r_{i,j}$ . We first point out that  $f_{r_{i,j}}$  is a lower-bound of  $f_{r_x}$ , which is the maximum of all values of  $f$  along the continuous ray  $r_x$ . With this observation, we can see that a conservative choice of  $g_{i,j}$  would be the minimum of  $f_{r_{i,j}}$  over all possible discrete paths  $r_{i,j}$  whose defining rays come out of some point  $x \in p_{i,j}$ .

In the parlance of digital geometry, the path of grid squares intersected by a straight ray is known as a *digital straight line* [8]. To compute  $g_{i,j}$  as described above, we could just enumerate all digital straight lines starting from  $p_{i,j}$ . Unfortunately, as shown in [8] and [1] there are  $O(n^3)$  digital straight lines that pass through a given grid square, so that approach does not seem to be a feasible method for computing  $g_{i,j}$  for all  $n^2$  grid squares. In 3D (our ultimate goal), the situation appears to be even worse. Instead, our approach will compute a lower bound of  $g_{i,j}$  as described above, by taking the minimum of  $f_{r_{i,j}}$  over a larger set of edge-adjacent square

paths  $r_{i,j}$  coming out of  $p_{i,j}$  that includes all the digital straight lines. Unlike digital straight lines, this expanded set of paths can be enumerated much more efficiently using local operations.

The algorithm starts by partitioning all view angles into a finite number of *view cones*. For each view cone, we compute at each grid square  $p_{i,j}$  a lower bound of  $f_{r_{i,j}}$  for all digital straight lines  $r_{i,j}$  whose defining rays lie in the view cone. This is done using an efficient two-part dynamic programming algorithm. Performing the algorithm for each view cone yields one value at each grid square  $p_{i,j}$ , and finally  $g_{i,j}$  is taken as the minimum of such values computed at  $p_{i,j}$  over all view cones. The algorithm is detailed below and summarized in the pseudo-code in Program 1.

**Partitioning the view angles**—To partition all view angles into a set of view cones, consider a  $2m \times 2m$  integer grid centered at the origin as shown in Figure 4(b) (where  $m = 3$ ). Forming vectors from the origin to the boundary grid points yields a set of  $8m$  vectors. Each pair of adjacent vectors defines a view cone for those rays whose slopes lie in the given cone. For example, the shaded cone in Figure 5 is bounded by the vectors  $(3, 1)$  and  $(3, 2)$ .

**Approximating CVF within each view cone**—Without loss of generality, we now focus on the computation for a single view cone. For the sake of simplicity, we consider the view cone that lies in the lower right half of the first quadrant and is bounded by the rays  $(m, l)$  and  $(m, l + 1)$  where  $0 \leq l < m$ . (The remaining cases can be converted to this case via the appropriate horizontal, vertical or diagonal reflection of  $f$ .)

Program 1: Pseudo-code for approximating the CVF for a 2D piece-wise density function  $f$

```

Partition the view cone into  $8m$  cases /*  $m$  is a small non-negative
integer*/
For each view cone
Flip  $f$  vertically, horizontally or diagonally such that the view cone is
bounded by the vectors  $(m, l)$  and  $(m, l + 1)$  where  $0 \leq l < m$ 
Let  $S$  be set of squares whose intersection with the convex hull of  $p_{0,0}$ ,
 $p_{m,l}$  and  $p_{m,l+1}$  is non-empty
Let  $g_{i,j} = 0$  for  $n \leq i < n + m$  and  $n \leq j < n + l + 1$  /*  $g$  is an  $n \times n$  density
grid */
For  $i = n - 1$  to  $0$  by  $-1$  and  $j = n - 1$  to  $0$  by  $-1$ 
Let  $h_{0,0} = 0$ 
For  $c = 0$  to  $m$  by  $1$  and  $d = 0$  to  $l + 1$  by  $1$ 
If  $p_{c,d} \in S$ 
Let  $h_{c,d} = \min(\max(f_{(i+c)^-}, j+d, h_{c-1,d}), \max(f_{i+c, (j+d)^-}, h_{c,d-1}))$ 
Let  $g_{i,j} = \min(\max(h_{m,l}, g_{i+m, j+1}), \max(h_{m,l+1}, g_{i+m, j+1+1}))$ 
Perform the inverse of the flips applied  $f$  to  $g$ 
Let the final  $g$  be the minimum of the  $g$  computed for each view cone

```

Our goal is to compute at each grid square (i.e.  $p_{0,0}$ ) a lower bound of  $f_{r_{0,0}}$ , noted as  $g_{0,0}$ , for all digital straight lines  $r_{0,0}$  defined by rays in the view cone. Our key observation here is any segment of a digital straight line in the view cone is also a digital straight line in the same view cone. We thus compute the desired lower bound  $g_{0,0}$  in two stages. In the first stage, we consider all digital straight lines  $r_{0,0}$  with horizontal span  $m$  in the view cone. Note that such lines must



end either at the grid square  $p_{m,l}$  or  $p_{m,l+1}$ . We obtain the lower bound of  $f_{r_{0,0}}$  over all  $r_{0,0}$  ending at  $p_{m,l}$  and  $p_{m,l+1}$  respectively as  $h_{m,l}$  and  $h_{m,l+1}$  (elaboration to follow). In the second stage, we obtain the final lower bound  $g_{0,0}$  for all digital straight lines in the view cone (with horizontal spans no smaller than  $m$ ) using a simple dynamic programming pass. Assuming  $g_{(m,l)}$  and  $g_{(m,l+1)}$  have been computed correctly, we compute

$$g_{(0,0)} = \min(\max(h_{m,l}, g_{m,l}), \max(h_{m,l+1}, g_{m,l+1})) \quad (1)$$

To obtain the lower bounds  $h_{m,l}$ ,  $h_{m,l+1}$ , one could enumerate the digital straight lines starting at  $p_{0,0}$  and ending at either  $p_{m,l}$  or  $p_{m,l+1}$ . Since  $m$  is small, such enumeration wouldn't be so costly. Alternatively, one can use a dynamic programming scheme similar to the above description to compute a more conservative bound, which is implemented here. Specifically, let us consider the set of all edge-adjacent square paths  $r_{0,0}$  starting from grid square  $p_{0,0}$ , moving only rightward or upward, ending at either  $p_{m,l}$  or  $p_{m,l+1}$ , and visiting only those grid squares intersecting with the convex hull of squares  $p_{0,0}, p_{m,l}, p_{m,l+1}$ . An example of the allowed squares for  $m = 3$ ,  $l = 2$  is shown shaded in Figure 5). Note that these paths include all digital straight lines with slope bounded by the view cone. These paths can be enumerated, and the minimum of  $f_{r_{0,0}}$  over these paths can be updated in a single dynamic programming pass. Starting with  $h_{0,0} = 0$ , we compute  $h_{c,d}$  for all allowed squares  $p_{c,d}$  inductively as

$$h_{c,d} = \min(\max(f_{c^-,d}, h_{c-1,d}), \max(f_{c,d^-}, h_{c,d-1})) \quad (2)$$

where  $f_{c^-,d}$  denotes the minimum of  $f$  along the edge shared by squares  $p_{c,d}$  and  $p_{c-1,d}$ .

Overall, the dynamic programming sweep in Equation 1 processes the squares  $p_{i,j}$  in decreasing values of  $i$  and  $j$ . To apply this recurrence near the right and upper boundaries of the grid, we pad the grid with  $l + 1$  rows and  $m$  columns of zeros. At each square  $p_{i,j}$ , the dynamic programming pass in Equation 2 is performed to supply the outer dynamic programming sweep with the necessary quantities  $h_{m,l}$ ,  $h_{m,l+1}$  (see the pseudo-code in Program 1).

### 3.3. Generalization to 3D

To extend the algorithm to a 3D density function  $f(x)$ , we will similarly compute an approximation to the CVF  $g_f(x)$  that is constant within each grid cell. The value  $g_{i,j,k}$  for a cell  $p_{i,j,k}$  is computed as a lower bound of  $f_{r_{i,j,k}}$  over all digital straight lines  $r_{i,j,k}$  (consisting of face-adjacent cells intersecting some ray) passing through  $p_{i,j,k}$ . Here,  $f_{r_{i,j,k}}$  is the maximum of the minimum of  $f$  on each grid face between successive cells on  $r_{i,j,k}$ . Using a similar argument as in 2D, such choice of  $g_{i,j,k}$  is a conservative approximation, i.e.,  $g_{i,j,k} \leq g_f(x)$  for any  $x \in p_{i,j,k}$ .

To compute  $g_{i,j,k}$ , the 2D dynamic programming algorithm only needs to be slightly modified. To partition the view angles, we use a cube of size  $2m \times 2m \times 2m$  centered at the origin, whose boundary is divided into  $24m^2$  unit squares. Each unit square defines a view pyramid bounded by four vectors of the form  $(m, l, o)$ ,  $(m, l+1, o)$ ,  $(m, l, o+1)$ , and  $(m, l+1, o+1)$  where  $0 \leq l, o < m$ . The rest of the algorithm proceeds as before.

**Complexity**—The most time-consuming step is the inner dynamic programming pass (Equation 2), which has the complexity of  $O(m)$  (equalling the number of cells intersecting with the convex hull of the view pyramid, which is bounded by  $mk$  for some constant  $k$ ). This pass needs to run once for each grid cell and once for each view pyramid, hence the total asymptotic complexity is  $O(n^3 m^3)$ . The algorithm has an “embarrassingly parallel” structure, as the computation for each of the  $24m^2$  view pyramids is completely independent of each

other. Hence the practical performance can be easily improved by employing multiple processors (or cores).

#### 4. Contour culling using CVF

Our algorithm outputs a 3D visibility volume containing one value  $g_{i,j,k}$  for each grid cell  $p_{i,j,k}$  in the input volume. For a tri-linear contouring algorithm to take advantage of this output, simply skip cells whose  $g$  values are greater than the iso-value. The conservativeness of our computation ensures that culling does not affect the visible part of the iso-surfaces in any views. This is illustrated in 2D in Figure 6. Note that the interior part of the iso-curve (red curve in (d)) is culled during contouring, since the  $g_{i,j}$  values of the enclosing grid squares are greater than the iso-value (such grid squares are outlined by purple dots).

The visibility volume can also be utilized by polygonal contouring algorithms that approximate the tri-linear contours and proceed in a similar cell-by-cell fashion, such as Marching Cubes. Culling in these algorithms based on  $g_{i,j,k}$  is also conservative (i.e. the visible part of the polygonal surface is not affected in any views) if the polygonal contours satisfy a mild condition: a grid face whose scalar values at all four corners are above the iso-value always lies inside the polygonal iso-surface. Note that this condition holds in most algorithms like Marching Cubes.

#### 5. Results

We demonstrate our algorithm on a suite of synthetic data as well as real-world bio-medical images (e.g., MRI, CT, cryo-EM). To quantify the effectiveness of culling, we consider the percentage of total triangles generated by Marching Cubes [11] that are culled using our approach (as described in Section 4) and refer to this as the *reduction rate*.

##### Choice of $m$

This is the only parameter of our algorithm, which determines the number of view partitions. Increasing  $m$  would result in a better approximation to the actual CVF because more and finer view cones (pyramids) are explored and because longer segments of discrete paths are used to approximate digital straight lines. However, larger  $m$  would also significantly increase the computational cost due to the  $m^3$  part in the algorithm complexity (see Section 3). In practice, we notice that higher values of  $m$  typically do not yield significantly better reduction rate. This is demonstrated in Figure 7, which shows the computational cost for increasing  $m$  on the Foot example (Figure 1) and the reduction rate at all iso-values for  $m = 1$  and  $m = 6$ . In our examples, we used  $m = 2$  which provides a good balance between the amount of reduction and running time. Figure 8 shows the corresponding visual impact of the  $m$  parameter.

##### Results

Figure 9 shows the graph of reduction rate for all test examples in this paper. Each curve represents a single density map. The “test128” example is synthetically created by randomly perturbing a constant density volume. The graph shows that the reduction can react sharply with respect to the iso-value, with best rates typically occurring in the middle range. In some cases, it is possible to achieve up to an 80% reduction in polygon counts.

Table 1 visually shows the amount of reduction (for one iso-value) in these test examples. To visualize the hidden surface parts, we used depth-peeling and line drawing to reveal the inner layer of each mesh extracted with Marching Cubes. This rendering technique, called blueprint rendering, was described by Niehaus and Dollner [14]. We modified their original algorithm to produce red lines for the inner layers of the depthpeeled image. Generally, the inner layers



reveals the inner polygons that are invisible when viewed from infinity. Table 2 shows culling on one example (a cryo-EM image of a virus) at different iso-values.

## Performance

Our algorithm is tested on an Intel Xeon machine of 8 cores with clock-rate of 2.5Ghz. As mentioned in the complexity analysis, our algorithm is easily parallelizable. Table 3 shows the execution time with respect to the number of cores used in the computation, exhibiting an expected linear speed-up. Table 4 shows the timing results for all examples, using 4 cores with our implementation. Observe that the computation on our largest dataset ( $256 \times 256 \times 397$ ) finishes in less than 10 minutes. Note that the computation is done once for each volume, after which the only operation needed during contouring is a scalar value comparison for each cell to determine its visibility.

## 6. Conclusion and discussion

We have described a novel contour culling method for far-field viewing of density maps that reduces the complexity of surfaces extracted by standard contouring techniques without altering their visual appearance. The algorithm pre-computes a visibility function (CVF) that is independent of view directions and considers all iso-values. The function can be used in conjunction with contouring to cull internal surface components at any iso-value with little overhead added to the contouring method. The effectiveness of the approach is demonstrated on several bio-medical data, and an application for online viewing of molecular density maps is presented.

Obviously, our culling method is not a replacement for existing view-dependent methods, which work for any viewpoints and also cull more triangles from any particular view point than our method. Our method offers some unique features in the far-field viewing scenario, such as the minimal run-time computational cost. It is also simple to incorporate it into any contouring implementations (the only addition is a comparison test between values in the data and values in the visibility map). These features make our method suited for light-weight visualization tools, which are explored in this paper, and also allow potential integration of our method into existing view-dependent culling systems to offer improved culling efficiency when the viewpoint is taking from outside the volume (since fewer triangles need to be considered for culling at any particular view).

For future work, we will explore other methods for enumerating digital straight lines that provide a better coverage and lower complexity than the current dynamic programming algorithm. In this work, we were not able to provide an error bound on our approximation to the ideal CVF, and proving such bound might be another possibility for future research. Additionally, we would like to further explore the trade-off space of the accuracy of rendering versus the amount of culling. While we only consider conservative culling in this paper, in most visualization applications the conservative constraint can be possibly relaxed to allow more aggressive culling with a small sacrifice in visual accuracy.

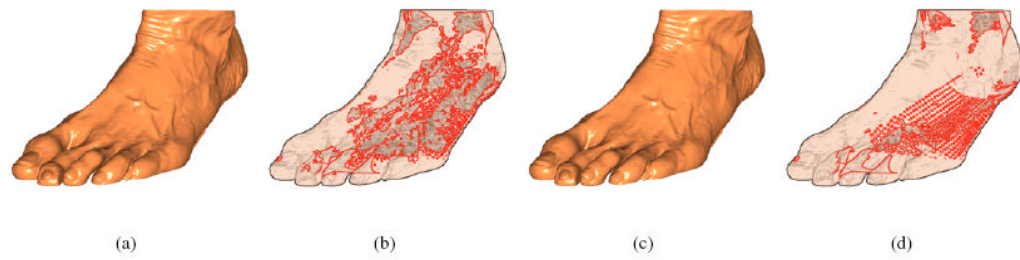
## Acknowledgments

We would like to thank the reviewers for their helpful comments. We thank Digital Morphology Library, Electron Microscopy Data Bank, Volume Library, and volvis.org for providing the datasets. This work is supported in part by NSF grants IIS-0705538 and IIS-0846072, and NIH grants R21DK79457 and R01GM079429.

## References

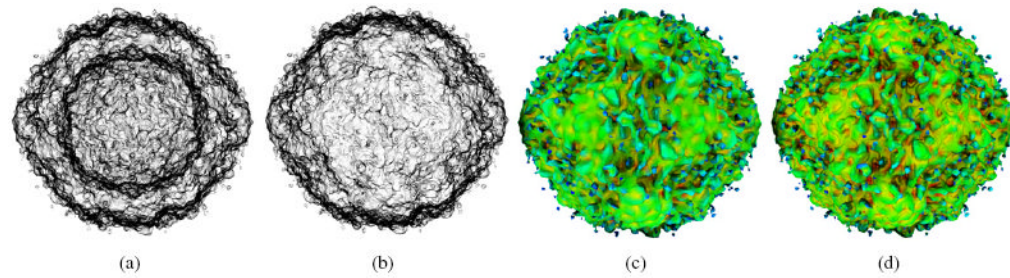
1. Andres E, Acharya R, Sibata C. Discrete analytical hyperplanes. *Graphical Models and Image Processing*. 1997; 59(5):302–309.

2. Cohen-Or D, Chrysanthou YL, Silva CT, Durand F. A survey of visibility for walkthrough applications. *IEEE Transactions on Visualization and Computer Graphics*. 2003; 9:412–431.
3. Gao, J.; Huang, J.; Shen, H-W.; Kohl, JA. VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03). IEEE Computer Society; Washington, DC, USA: 2003. Visibility culling using plenoptic opacity functions for large volume visualization; p. 45
4. Gao, J.; Shen, H-W. PVG '01: Proceedings of the IEEE 2001 symposium on parallel and largedata visualization and graphics. IEEE Press; Piscataway, NJ, USA: 2001. Parallel view-dependent isosurface extraction using multi-pass occlusion culling; p. 67-74.
5. Gao, J.; Shen, H-W.; Huang, J.; Kohl, JA. VIS '04: Proceedings of the conference on Visualization '04. IEEE Computer Society; Washington, DC, USA: 2004. Visibility culling for time-varying volume rendering using temporal occlusion coherence; p. 147-154.
6. Garland, M.; Heckbert, PS. Proceedings of the 24th annual conference on Computer graphics and interactive techniques. SIGGRAPH '97. ACM Press/Addison-Wesley Publishing Co.; New York, NY, USA: 1997. Surface simplification using quadric error metrics; p. 209-216. URL <http://dx.doi.org/10.1145/258734.258849>
7. Gregorski B, Senecal J, Duchaineau M, Joy KI. Compression and occlusion culling for fast isosurface extraction from massive datasets. *Mathematical Foundations of Scientific Visualization, Computer Graphics, and Massive Data Exploration*. 2009:303–323.
8. Klette R, Rosenfeld A. Digital straightness—a review. *Discrete Applied Mathematics*. 2004; 139(1-3): 197–230. the 2001 International Workshop on Combinatorial Image Analysis.
9. Li, W.; Mueller, K.; Kaufman, A. VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03). IEEE Computer Society; Washington, DC, USA: 2003. Empty space skipping and occlusion clipping for texture-based volume rendering; p. 42
10. Livnat, Y.; Hansen, C. VIS '98: Proceedings of the conference on Visualization '98. IEEE Computer Society Press; Los Alamitos, CA, USA: 1998. View dependent isosurface extraction; p. 175-180.
11. Lorensen, WE.; Cline, HE. SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques. ACM; New York, NY, USA: 1987. Marching cubes: A high resolution 3d surface construction algorithm; p. 163-169.
12. Mroz L, Hauser H, Groller E. Interactive high-quality maximum intensity projection. *Comput Graph Forum*. 2000; 19(3)
13. Mroz, L.; Konig, A.; Groller, E. *Data Visualization '99*. Springer; 1999. Real-time maximum intensity projection; p. 135-144.
14. Nienhaus M, Dllner J. *GPU Gems II: Programming Techniques for High Performance Graphics and General-Purpose Computation*. Addison-Wesley Professional, Ch *Blueprint Rendering and Sketchy Drawings*. 2005:235–252.
15. Nirenstein S, Blake E, Gain J. Exact from-region visibility culling. *Proceedings of the 13th Eurographics workshop on Rendering EGRW '02*. 2002:191–202.
16. Pesco S, Lindstrom P, Pascucci V, Silva CT. Implicit occluders. *Volume Visualization and Graphics, IEEE Symposium on*. 2004:47–54.
17. Zhang, H.; Manocha, D.; Hudson, T.; Hoff, KE, III. Proceedings of the 24th annual conference on Computer graphics and interactive techniques SIGGRAPH '97. ACM Press/Addison-Wesley Publishing Co.; New York, NY, USA: 1997. Visibility culling using hierarchical occlusion maps; p. 77-88. URL <http://dx.doi.org/10.1145/258734.258781>



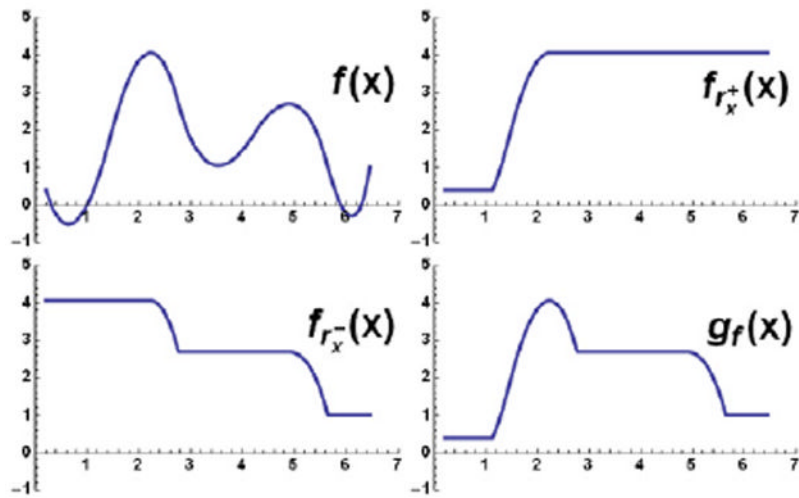
**Figure 1.**

(a,c): Iso-surfaces of a CT foot scan at the same iso-value generated by Marching Cubes without and with culling, containing 1020570 and 592456 triangles respectively. (b,d): Transparent rendering of the polygons in (a,c) (see details in Section 5), showing internal structures in the original iso-surface that have been largely culled away using our technique.



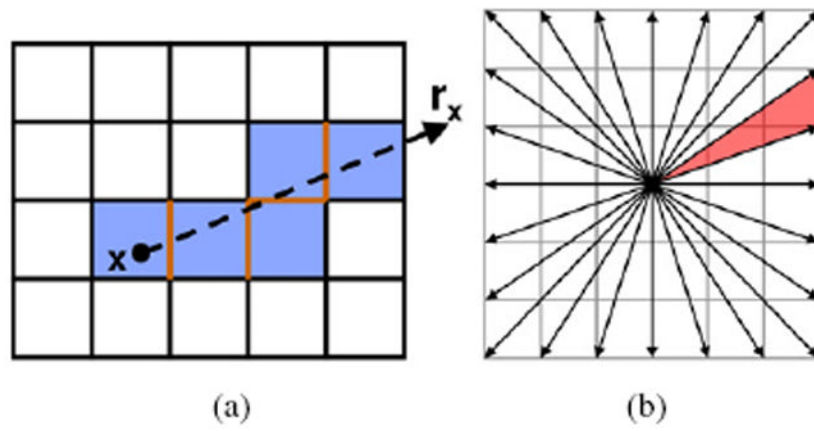
**Figure 2.**

This is a poliovirus with an inner surface not visible from a far-field view. All meshes are generated from a  $201^3$  volume and simplified to 100K triangles. The mesh generated from the original density data contains an inner surface (a,c), and the one generated from our contour-culled density shows that the inner surface has been removed (b,d). Images (c) and (d) show the meshes colored by curvature. By comparing the variation in curvature, we see that the contour-culled version retains more surface details than the mesh from the original volume.



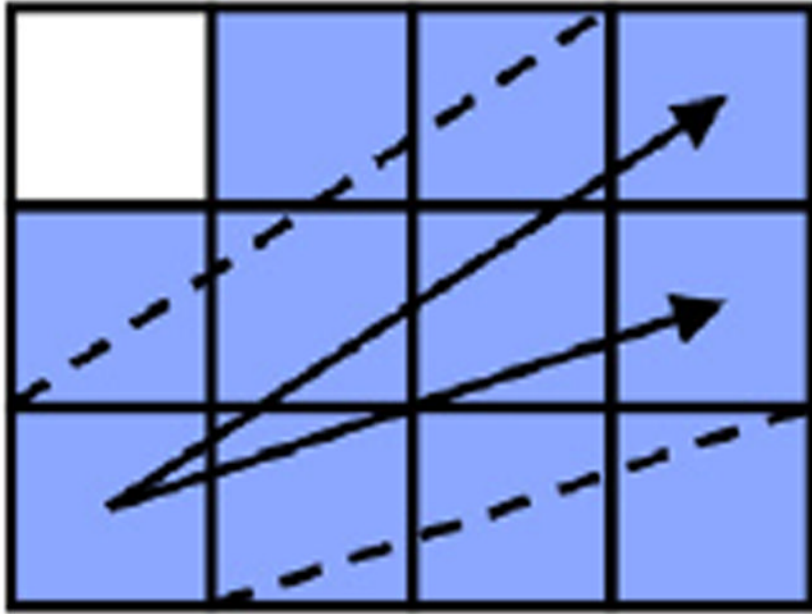
**Figure 3.**

The contour visibility function in 1D:  $f(x)$  is the original function,  $f_{r_x^+}$ ,  $f_{r_x^-}$  are maximum values along the two rays respectively from  $x$  to  $+\infty$  and  $-\infty$ , and  $g_f(x)$  is the minimum of  $f_{r_x^+}$  and  $f_{r_x^-}$ .

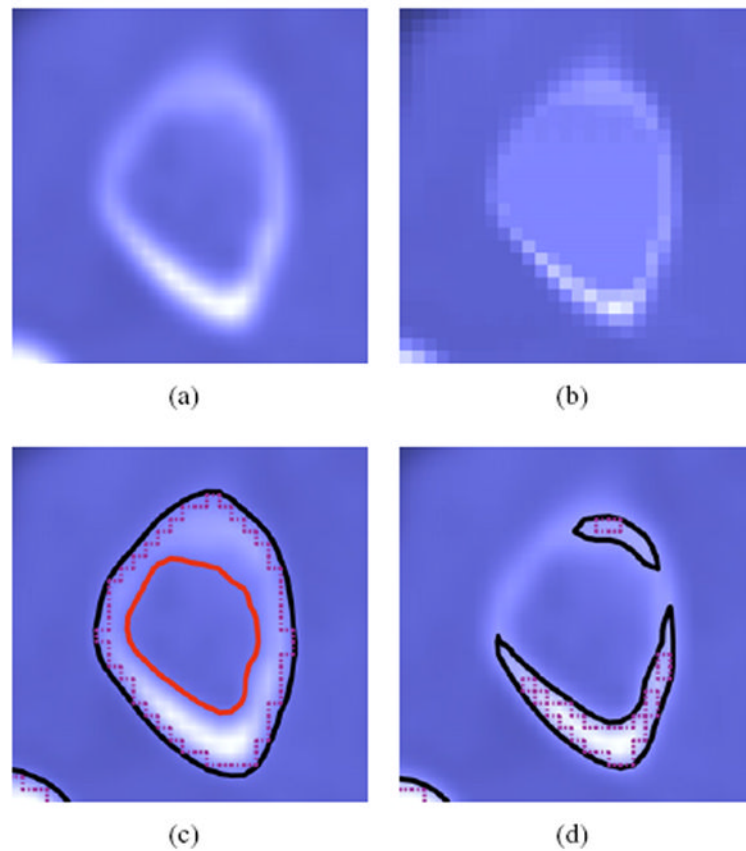


**Figure 4.** (a) A ray  $r_x$  and the digital straight line it defines (shaded) which consists of edge-adjacent squares. (b) The partition of the view angles into view cones. The highlighted triangle is a single view cone.

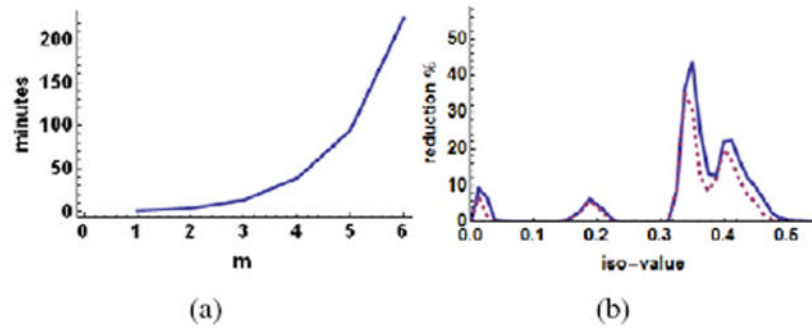




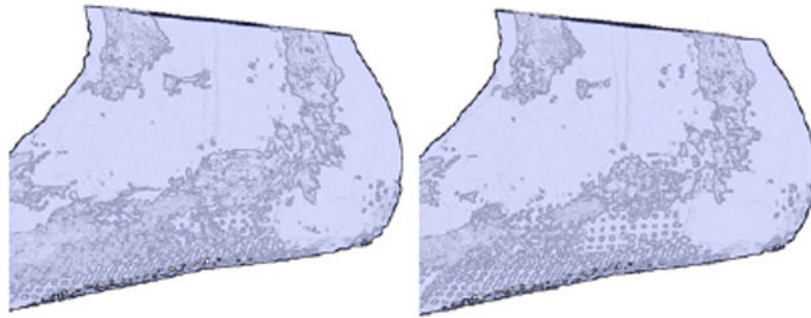
**Figure 5.** Grid squares that intersect with the convex hull of a view cone (marked by the dotted line).



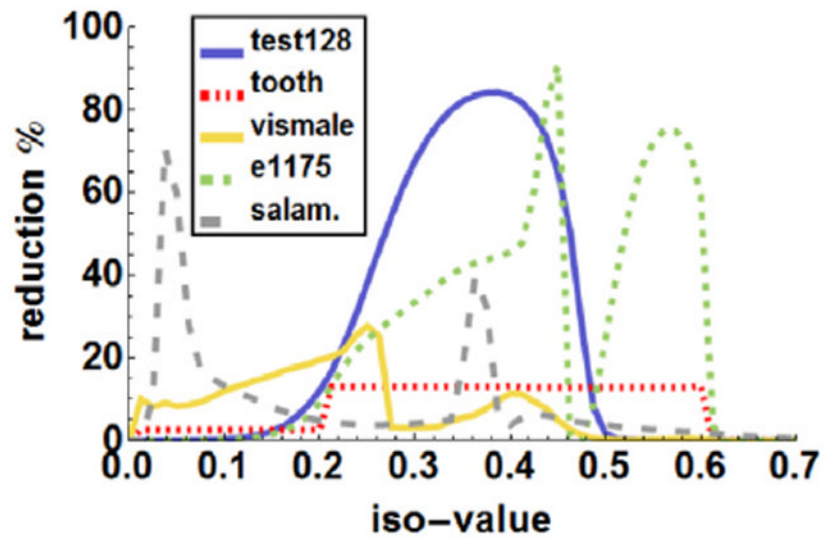
**Figure 6.** 2D example of contour culling. (a): A tri-linearly interpolated density function (from the Foot data). (b): The  $g_{i,j}$  computed by our algorithm for each grid square. (c,d): iso-curves of (a) at low and high iso-values, where interior curve parts invisible to the outside (red) are culled away. Purple dots outline those grid squares whose  $g_{i,j}$  are greater than the iso-value, and hence where the culling takes place.



**Figure 7.** (a): Running time of our algorithm on the Foot data as  $m$  increases. (b): Reduction rate at all iso-values for  $m = 6$  (blue solid line) and  $m = 1$  (purple dashed line).




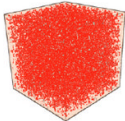

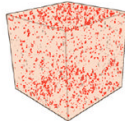









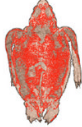


**Figure 8.** The foot data iso-contoured for  $m = 2$  (a) and  $m = 6$  (b). The chosen iso-value is .351. Modified Marching Cubes generated 884135 and 778579 triangles for  $m = 2$  and  $m = 6$  respectively.



**Figure 9.**  
Reduction rate of culling on several datasets at varying iso-values.

**Table 1**

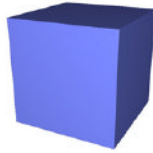
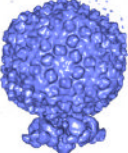
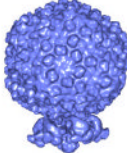
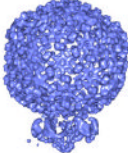
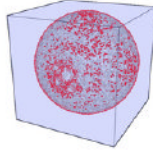
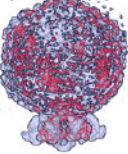
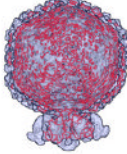
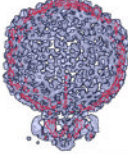
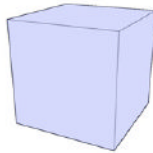
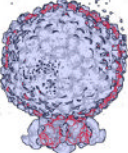
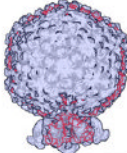
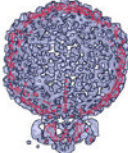
Comparisons of iso-surfaces extracted by Marching Cubes on several data sets without culling (first column) and with culling (third column) at a single iso-value. The transparent renderings (second and fourth columns) reveal the internal portions.

	original	original+transparent	culled	culled+transparent	reduction
test128					
	989984 triangles		265048 triangles		73.2%
tooth					
	128908 triangles		112232 triangles		12.9%
vismale					
	703636 triangles		526792 triangles		25.1%
turtle					
	2174216 triangles		1361838 triangles		37.4%



**Table 2**

Contour culling on virus particle *e1175* (Epsilon 15) from the Electron Microscopy Data Bank (EMDB) at various iso-values, showing the iso-surface extracted by Marching Cubes (top row), internal portions before culling (second row) and after culling (last row).

iso-value	0.44	0.50	0.55	0.74
far-field view				
without culling	 5935276 triangles	 597426 triangles	 1504356 triangles	 527112 triangles
with culling	 988428 triangles	 446166 triangles	 430450 triangles	 527112 triangles
reduction	83.3%	25.3%	71.4%	0%

**Table 3**

Running time on the Foot data using multiple cores (with  $m = 2$ ).

# of cores	1	2	4
time (min)	13.43	6.91	3.64

**Table 4**

The running time of our algorithm on all datasets. The timings for running regular marching cubes and our modified marching cubes are shown in the rightmost two columns. The results suggest no significant difference in running time.

dataset	size	contour culling (min)	MC (sec)	mod-MC (sec)
test128	$128 \times 128 \times 128$	.46	.229	.168
vismale	$256 \times 256 \times 128$	1.86	.092	.100
tooth	$256 \times 256 \times 161$	2.42	.172	.150
foot	$203 \times 418 \times 189$	3.64	.159	.156
e1175	$288 \times 288 \times 288$	5.29	.204	.178
turtle	$256 \times 256 \times 397$	6.10	.185	.191