Line Drawings from Volume Data

Michael Burns[1] Janek Klawe[1] Szymon Rusinkiewicz[1] Adam Finkelstein[1] Doug DeCarlo[2]
[1]Princeton University [2]Rutgers University

Abstract

Renderings of volumetric data have become an important data analysis tool for applications ranging from medicine to scientific simulation. We propose a volumetric drawing system that directly extracts sparse linear features, such as silhouettes and suggestive contours, using a temporally coherent seed-and-traverse framework. In contrast to previous methods based on isosurfaces or nonrefractive transparency, producing these drawings requires examining an asymptotically smaller subset of the data, leading to efficiency on large data sets. In addition, the resulting imagery is often more comprehensible than standard rendering styles, since it focuses attention on important features in the data. We test our algorithms on datasets up to 512^3, demonstrating interactive extraction and rendering of line drawings in a variety of drawing styles.

Keywords: visualization, volume, isosurface, NPR, silhouettes, suggestive contours

1 Introduction

Volumetric data sets are widely used in scientific and medical applications. They can arise both from scans of real-world phenomena (such as a CT or MRI scan of the brain) and from simulation (for example, fluid flow near an airplane engine intake). As scanning hardware and simulation algorithms have become more sophisticated, the size of these data sets has grown, pushing the limits of computing resources and taxing the ability of scientists to understand them. Effective visualization tools must therefore be both *efficient* on large data and *comprehensible* for the user, and visualization research has focused on developing techniques that address both of these criteria.

There are two major classes of rendering algorithms for volume data: those that render the volume itself (with nonrefractive transparency), and others that treat the volumetric data as an implicit function and render an isosurface. This distinction has implications on both efficiency and comprehensibility. In particular, since an isosurface is a two-dimensional manifold embedded in the three-dimensional space of the data, its size in bandlimited* datasets grows asymptotically more slowly than the size of the volume itself. Isosurface extraction algorithms can take advantage of this sparseness to achieve greater efficiency than is possible with methods that must visit each voxel [van Kreveld et al. 2004]. In addition, by hiding the clutter of multiple overlapping layers of data, isosurface renderings can be easier to understand and interpret.

Motivated by the above discussion, as well as developments in non-photorealistic line drawing algorithms for surfaces, we propose a further dimensionality reduction in volume rendering: our system *directly* extracts and renders linear features that lie on isosurfaces within the volume. Thus, we exploit not only the sparseness of surfaces within the volume, but also the sparseness of lines on a surface. We argue that this both reduces the amount of data that must be examined while rendering a frame (to a one-dimensional subspace of the volume) and leads to greater visual clarity, while conveying the same information as traditional shaded isosurface renderings. We note that the benefits of emphasizing linear features such as ridges and valleys or silhouettes in traditional volume renderings have been demonstrated by a number of researchers, e.g. [Ebert and Rheingans 2000; Interrante et al. 1995; Lu et al. 2003; Lum and Ma 2002; Nagy et al. 2002; Svakhine and Ebert 2003]. Our work carries this further by proposing a framework in which such lines are the fundamental rendering primitives.

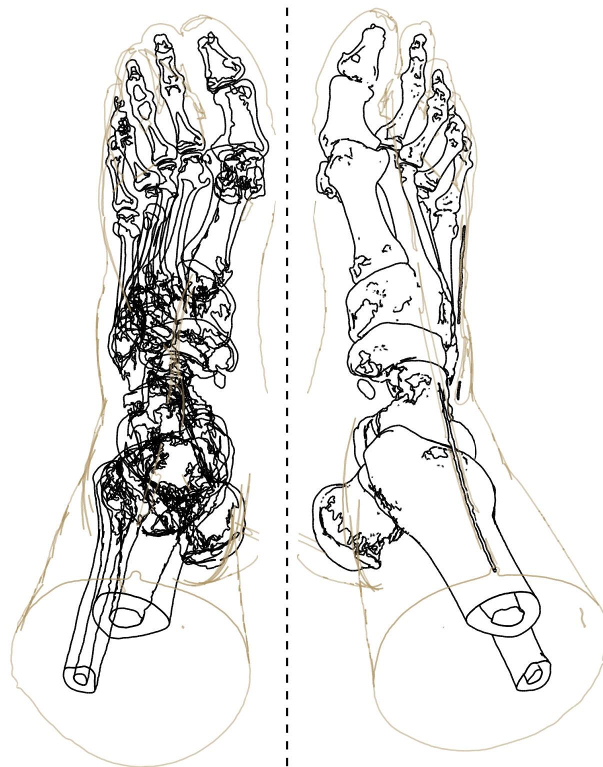


Figure 1: Two line drawings of isosurfaces extracted directly from a $512^2 \times 256$ voxel grid. Without hidden surface removal, the left drawing renders at interactive frame rates on a laptop.

We demonstrate a system that implements a volumetric rendering framework based on lines (Figure 1). Our system uses a randomized, temporally coherent seed-and-traverse algorithm inspired by one originally developed for silhouette extraction from meshes [Markosian et al. 1997]. Several types of lines are supported (Figure 2), including intersections with cutting planes, occluding contours (i.e., interior and exterior silhouettes), and suggestive contours, which are features recently introduced by DeCarlo et al. [2003] that help convey shape. Comprehensibility may

*By "bandlimited" we mean that the isosurface complexity does not grow with the size of the dataset; the same structures appear at higher resolution (albeit with greater fidelity).

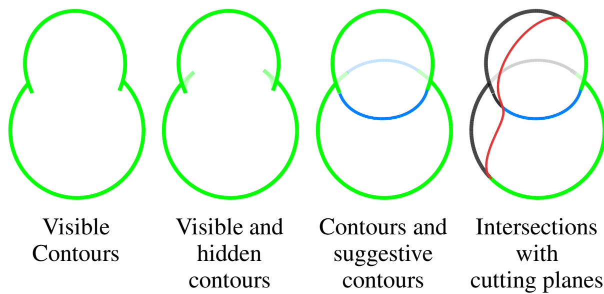


Figure 2: Types of lines supported by our system.

be further improved by choosing drawing styles based on factors such as visibility, and the selected isovalue may be placed in context by simultaneously drawing lines associated with other isovalues. By selecting which features to draw, we provide a flexible range of trade-offs between rendering quality and interactivity. At its most basic settings, our system achieves interactive rates on commodity hardware for data sets of roughly 512^3 samples, while placing few demands on the graphics card (which could be utilized to provide further stylization or to simultaneously display an extracted mesh).

2 Related work

As this paper focuses on rendering isosurfaces, we omit discussion of work on volume rendering using transfer functions, except where it involves non-photorealistic techniques (as discussed below). Dealing with the huge amount of data associated with a grid in 3D is a central challenge in rendering from volume data, and therefore most visualization techniques are not designed for interactive frame rates. However, several general strategies may be applied for **accelerated extraction and rendering** of isosurfaces. First, researchers have developed methods for extracting isosurfaces *without* traversing the entire volume, for example seed-and-traverse methods (e.g. [van Kreveld et al. 2004]) and hierarchical methods (e.g. [Cignoni et al. 1997]). Such algorithms exploit the property that the isosurface is typically sparse in the volume; our method also enjoys this property, but is further enhanced by the fact that the feature lines we draw are sparse on the isosurface. Second, view-dependent methods avoid drawing invisible portions of the isosurface, exploiting the fact that for some surfaces depth complexity can be high, and only the front layer needs to be drawn [Parker et al. 1998; Livnat et al. 1996]. Our algorithms are also view-dependent because of the nature of the features being drawn, but do not exploit this property for acceleration because computing visibility is expensive (Section 4.2). Finally, the stochastic method for searching the volume for isosurface feature lines described in Section 3.4 has a property enjoyed by previous methods [Liu et al. 2002; Parker et al. 1998] – the picture may not be 100% "correct" when the user first turns the camera or dials the isosurface threshold to a particular value; however, the response rate is interactive, and the "correct" picture is guaranteed to resolve soon thereafter.

The other aspect of rendering volumetric data addressed by this paper is **comprehensibility**, obviously a critical quality of effective scientific visualization. Researchers have applied various principles from non-photorealistic rendering (NPR) techniques to make renderings from grid data more effective. Kirby et al. [1999] showed that different forms of scalar and vector data organized in a 2D grid may be displayed simultaneously by adapting techniques from painting. For 3D data sets, researchers have created illustrations in the styles of pen and ink illustration [Dong et al. 2003; Treavett and Chen 2000], hatching and toon shading [Nagy et al. 2002], and stipple drawings [Lu et al. 2003] that can effectively convey volumetric data. Perhaps most similar to this work are the efforts for emphasizing linear features such as object boundaries by explicitly drawing or darkening silhouettes or by drawing "halos" to convey depth ordering (e.g, [Interrante and Grosch 1998; Ebert and Rheingans 2000; Csébfalvi et al. 2001; Lum and Ma 2002; Nagy et al. 2002; Svakhine and Ebert 2003; Kindlmann et al. 2003]). Our work also falls into the line of research of bringing NPR techniques to visualization, though the visual effect – line drawings of isosurfaces – is quite different than that of previous efforts, with a few exceptions.

Bremer and Hughes [1998] use a randomized algorithm similar to ours to render silhouettes from analytic implicit functions. Our technique, in contrast, operates on regularly-sampled volumetric data, and uses a voxel-based "marching lines" technique, as opposed to numerical integration, to trace contours.

Schein and Elber [2004] extract high-quality silhouettes from volumetric data at non-interactive frame rates by modeling the data with B-spline functions, using a lengthy preprocess for faster rendering. In contrast, our method renders high-quality contours at interactive frame rates without any preprocess.

Nagy and Klein [2004] render surface contours from volumes using modern programmable graphics hardware and the traditional volume rendering pipeline. As a result, volumes must be downloaded to the graphics card memory and rendering time is highly dependent on output resolution. Our approach, in contrast, generates resolution-independent line geometry from the volume in main memory, allowing for interactive rendering of very high-resolution images of large volumes.

3 Fast Line Extraction

This section describes how to extract sparse linear features from volumetric datasets. Although we make the discussion specific by focusing on occluding contours (interior and exterior silhouettes), the techniques we develop are general. Their applications to different line types are described in Section 4.1.

Our main algorithmic challenge is achieving extraction of lines in time proportional to the size of the extracted features (i.e., output sensitivity). While it is possible to design pathological datasets of size $n \times n \times n$ that produce isosurfaces of size $O(n^3)$ with silhouettes also of size $O(n^3)$ (for example a Peano curve in 3D), these shapes are not characteristic of typical datasets that one would want to visualize. In practice, isosurfaces describing a particular bandlimited shape represented by a volume of size n^3 will have size $O(n^2)$, at least at the isovalues one typically wishes to visualize. Furthermore, the contours of those isosurfaces will have size $O(n)$. Markosian et al. [1997] effectively argue for the latter statement by showing that silhouettes of a shape have size proportional to the square root of the number of faces representing that shape. Their argument might also be adapted to address the first claim (about the size of the isosurface), although it would be complicated by the nature of the surface extracted by the marching cubes algorithm. Nonetheless, these claims are borne out by our experiments, in particular the graph in Figure 7.

3.1 Contours in Volumes

Our input data $\phi(i,j,k)$ is a 3D matrix of real-numbered data values, positioned at the nodes of a regularly spaced lattice. We begin by considering the task of rendering contours on isosurfaces of ϕ. Recall that an isosurface F can be defined as the zero-set of the function

$$f(i,j,k) = \phi(i,j,k) - \tau, \tag{1}$$

where τ is a threshold within the range of the data values. This equation $f = 0$ implicitly defines a 2D surface in 3D space.

Contours on a continuous surface are those locations where the surface normal is perpendicular to the view. That is, they are

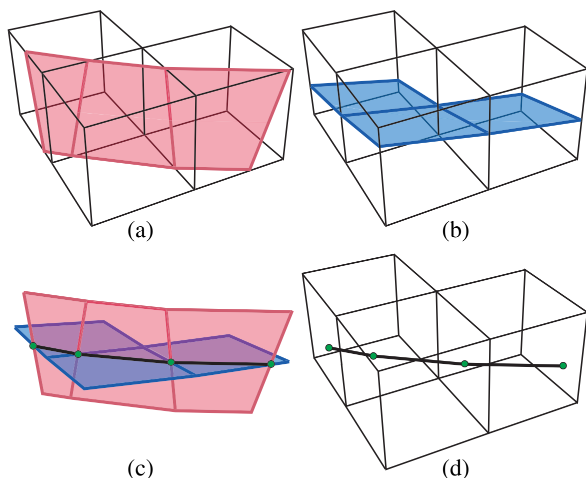


Figure 3: Extracting silhouettes. (a) Isosurface F in red traverses three neighboring voxels. (b) Contour surface C is blue. (c) F and C intersect in a curve, which can be traced through the volume (d).

locations at which $n \cdot v = 0$, where n is the surface normal and v is the view vector (i.e., a unit vector from a point on the surface to the camera). In our application of volume rendering, we could use this definition directly by first extracting an isosurface at some threshold τ, computing the surface normal as $n = -\nabla\phi$, and extracting the contour. Alternatively, we can consider the set of contours on *all* possible isosurfaces of ϕ, which is itself a 2D surface defined as the zero set C of the function

$$c(i,j,k) = -\nabla\phi(i,j,k) \cdot v(i,j,k). \tag{2}$$

The task of finding contours at a specific threshold τ then reduces to finding the intersection of the 2D surfaces F and C. Generically, this intersection takes the form of a set of 1D loops in 3D space.

In order to extract a contour, we first locate cubes containing zeros of both the f and c implicit functions; we describe fast methods for this below. Once a relevant voxel is found, we extract its contour segments using a variant of the Marching Lines algorithm [Thirion and Gourdon 1996]. This was originally presented as an adaptation of the well-known Marching Cubes technique [Lorensen and Cline 1987] to extracting crest lines on surfaces, but can be generalized to extract the intersection of isosurfaces of any two implicit functions.

Briefly, the algorithm finds intersections of the two implicit functions on the faces of the cube, using linear interpolation based on the values at the eight corners (Figure 3a-b). Then it finds any intersections between these two sets of lines on each face. The result is a set of points on the faces of the cube, which when connected yield segments of the contour (Figure 3c-d). Note that this method is completely general, in that it can extract the curves formed by the intersection of any two implicit functions on the volume. We rely on this property to extract other families of lines in addition to contours, as described in Section 4.1.

3.2 Walking Contours

The most basic method of finding contours would be to examine all cells in a volume, looking for cubes containing zeros of the implicit functions and extracting line segments in each. However, this approach is impractical for interactive visualization of large datasets due to its lack of output sensitivity. We instead use an accelerated technique that traces lines through the volume while attempting to avoid examination of the majority of cells.

Since the intersection of two 2D surfaces is in general a 1D loop, it is reasonable to extract the loop by following it through the volume. Given a starting *seed* cell, we extract a 1D segment of the contour with Marching Lines, then move to the cell adjacent to the face containing one of the intersection points and repeat (Figure 3). We continue this around the loop until we return to the seed cell. Walking through voxels that contain more than one line segment is performed in an arbitrary but consistent manner, to ensure that complete loops are formed.

Given this algorithm, and supposing we had at least one valid seed cell for every loop in the volume, we could then extract all contours in $O(m)$ time, where m is the total length of all loops. Thus, this algorithm is output sensitive, which is convenient since our output tends to be sparse for non-pathological cases. In the typical case, m is $O(n)$ in an n^3 volume, as argued above.

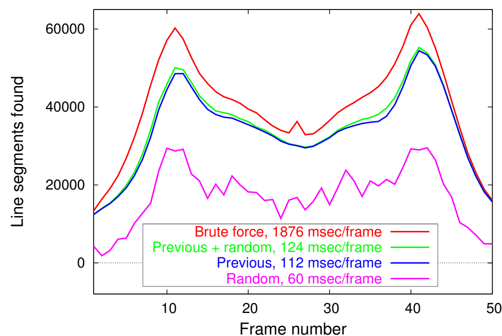


Figure 4: Efficiency of extracting contours using different seeding strategies, for an animation (with changing viewpoint and isosurface threshold) of the aneurysm dataset (Figure 8).

3.3 Exploiting Spatio-Temporal Coherency

We are then left with the problem of finding cells from which we can seed the contour extraction. While contours are view dependent and move along the surface as the viewpoint changes, they usually intersect contours of that surface from other viewpoints. To exploit this spatio-temporal coherency, we begin our search for contour seed points of a new frame by first examining cells that contained contours in the previously drawn frame – a method previously explored for finding contours [Markosian et al. 1997] and suggestive contours [DeCarlo et al. 2004] on surfaces. Since there are $O(m)$ cells containing contours from the previous frame, our search only adds $O(m)$ time to the extraction algorithm, leaving the overall running time as $O(m)$.

To evaluate the performance of this strategy, we conducted an experiment in which we measured the number of contour segments found in an animation involving continuous change in viewpoint and isosurface threshold. The red curve at the top of Figure 4 shows the number of contour segments extracted by the brute-force algorithm (i.e., "ground truth") at each frame in the sequence. The third curve, drawn in blue, shows the performance of using contours from the previous frame as seeds (the other two curves involve a random seeding approach, and are discussed in the next section). In all cases, the first frame is initialized with the brute-force algorithm. As can be seen, the temporally-coherent algorithm extracted most of the contours in all frames, with an average of 85% and a minimum of 70%. Although this performance is not perfect, we have observed that this is not objectionable in practice. First, short segments are more likely to be missed than long ones, so it is unlikely that the most perceptually important loops will be omitted. In addition, the missing line segments are more likely to occur during periods of quick motion, so they are less likely to be perceived. Finally, when the user stops changing the viewpoint and threshold, the random probing discussed in the next subsection

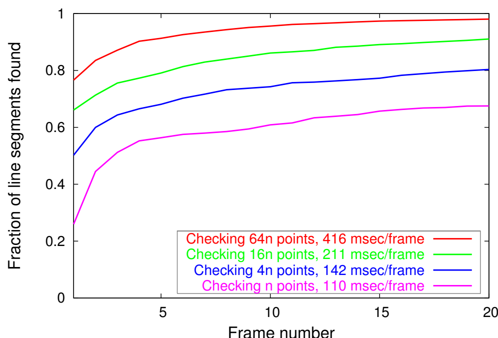


Figure 5: Efficiency of randomized probing, together with temporal coherence, for the skull isolevel of the head dataset (see Figure 10). The viewpoint and threshold are static. Because contours found on the previous frame are retained, the total number of contour segments found increases over time. We show results for different numbers of random probes, as a multiple of n (where n^3 is the number of voxels).

causes the missing segments to be located and drawn, with high probability, within a few frames.

3.4 Gradient Descent for Contour Location

Contours do appear and disappear through changes in view and isovalue. Therefore, in addition to the previous-frame seeding discussed above we search for new seeds using an iterative gradient descent method. This locates valid seed cells either nearby old contours or at random in the volume. As shown in the magenta (lower-most) curve of Figure 4, random sampling and gradient descent can be used alone to find a large percentage of new contours, though leveraging information from the previous frame accelerates this process.

Given a stating point p in the volume, we calculate the gradient $g = \nabla\phi(p)$ and the vector

$$v_{walk} = \frac{\tau - \phi(p)}{|g|}\;\frac{g}{|g|}. \quad (3)$$

The vector v_{walk} is thus a linear approximation of the vector from p to the closest zero of f. We proceed to the point $q = p + v_{walk}$, and repeat this calculation at q using the gradient of the $n \cdot v$ implicit function. We then test if this cell is a valid seed for a new contour.

We repeat this process of alternately walking along gradients of f and c and testing for contour seeds. In our experiments with a wide variety of datasets, we have found that this process typically either finds the surface or fails (by leaving the volume or getting stuck in a local minimum) within four to five iterations. We thus terminate our search after five iterations if no seed point is found.

To find new contours, we can perform this gradient descent starting either at contour-containing cells from the previous frame or at random cells in the volume. If we were to restrict ourselves to purely random probing, potentially a large number of probes would be required to locate most of the contour segments (Figure 5).

4 Comprehensible Rendering

To render volume data in a comprehensible manner, we can extract any of a number of different families of lines using the method described in section 3. We optionally test extracted line segments for visibility, then assign to each a rendering style based on the type of line, whether it is visible, and a selection of other properties. Finally, the extracted segments are rendered as OpenGL polylines.

4.1 Families of Lines

Contours: As described in section 3, we compute the contour for our surface by extracting intersections of the isosurface $\phi - \tau = 0$ and the contour surface $n \cdot v = 0$. The normals are computed as $n = -\nabla\phi$, and are evaluated as needed using a finite differences approximation over a local neighborhood. An alternative would be to precompute normals at program startup, which would increase run-time performance at a substantial ($4\times$) cost in memory usage.

As described below, we can omit or stylize the contours based on properties such as their length and visibility. Moreover, independently of their visibility, not all contours are worth drawing. Specifically, we can selectively omit those portions of the contour loops whose contour is never visible (from outside the surface). Such contours occur in locations such as the bottom of an indentation; they have negative radial curvature [Koenderink 1984]. We detect these "interior" contours by defining

$$s(i,j,k) = \nabla\big(\hat{n}(i,j,k) \cdot v(i,j,k)\big) \cdot v(i,j,k), \quad (4)$$

where $\hat{n}$ is the unit-length normal. As with the normal, the gradient is computed numerically, on demand. Since s has the same sign as radial curvature, interior contours are found by testing $s < 0$ once a segment has been extracted. Although most of the results in this paper omit these interior contours, they are visible as the hollow voids in the bones of the hand model (Figure 10, left).

Suggestive contours: Our renderings also include suggestive contours [DeCarlo et al. 2003], which complement contours in line drawings to produce more effective renderings of shape. Suggestive contours are defined as those surface locations where the radial curvature is zero, and its directional derivative towards the camera is positive. Thus, we extract suggestive contours as the zero set of s (as defined above), then check that the derivative of s towards the viewer is positive. In practice, we compare the derivative to a small positive threshold ($Ds > \varepsilon$) to filter out spurious zero crossings caused by noise [DeCarlo et al. 2004]. Furthermore, finding suggestive contours on backfaces requires the sign of the derivative test to be negated ($Ds < -\varepsilon$) in order to keep the analogous lines to those found on frontfaces. (DeCarlo et al. [2003] assumed opaque surfaces, hence did not address backfacing suggestive contours.)

Cutting plane intersections: We also render lines for the intersection of the isosurface with each of six axis-aligned cutting planes, with two planes per axis representing positive and negative bounds on the axis. Note that the same method as described in Section 3.1 for finding loops at the intersection of two surfaces addresses this problem, using the implicit function

$$p(i,j,k) = pos(i,j,k) \cdot n_{plane} - off_{plane}. \quad (5)$$

Cutting planes can also affect the visibility of other lines, with user control over omitting or applying stylization to lines on either the near or far side (Figure 10, center).

Multiple thresholds: As a final effect, we can simultaneously render lines from each of these families at multiple settings of the isosurface threshold τ. As shown in Figure 1 and Figure 10, center, this effect can unobtrusively provide context and enhance comprehensibility of the spatial relationship between multiple isosurfaces in the volume data.

4.2 Visibility

A significant aspect of rendering lines is determining which are visible and which are hidden. If we were extracting the entire isosurface, visibility could be obtained at little or no additional cost using a hardware z-buffer. However, if we are interested in drawing only the lines, this would impose at least the obvious $O(n^2)$ cost to download the mesh to the graphics card, not to mention the additional complexity of implementing an accelerated marching

cubes algorithm to extract the mesh in less than $O(n^3)$ time. Since we prefer to avoid extracting the entire isosurface in favor of finding sparse features on the surface, we need some method to determine which parts of these lines would be occluded by the surface.

We have implemented a simple solution, as follows: for each vertex of each rendered loop, we trace a ray through the volume from the vertex to the camera position (possibly using a "depth offset" heuristic to prevent immediate detection of self-intersection). As the ray passes through each cell in the volume, we determine whether it intersects the isosurface in that cell by examining the intersection of the ray with the face by which it leaves the cell. At that intersection, we use linear interpolation from the four corners to determine the function value and see if it has changed sign (indicating the ray has passed from outside to inside the isosurface). If it has, we mark the originating vertex as occluded. Note that, as a hint to our randomized drawing code, we could add the point of occlusion as a seed point for the randomized search, to ensure that the occluder is drawn.

Occluded vertices can be drawn in a different style, or not drawn at all, according to the user's preferences. The visibility algorithm can also be extended to handle multiple thresholds or cutting planes, simply by keeping track of all intersections against the isosurfaces and cutting planes of interest, as well as the order in which they occur. Examples of the use of visibility for one or more isosurfaces and cutting planes may be seen in figures throughout the paper.

The disadvantage to this method is that it is asymptotically as expensive as the z-buffer approach. In a volume of size n^3, the number of vertices rendered by our system is in $O(n)$, as is the number of cells touched by an average ray. Therefore, the time complexity for this visibility algorithm is $O(n^2)$. In practice, this method is still fast enough to be interactive, but it doesn't scale as well as the other algorithms in this application. For large data sets, if this algorithm is not fast enough for real-time rendering, one can compromise by only doing visibility calculations when the program is idle; when the user is manipulating the view, visibility is not computed, and so occluded contours are drawn in the same style as non-occluded contours.

There exist ways in which visibility could be computed more quickly for our application. For example, ray-to-ray caching and fast rejection techniques would reduce the constant factors in our implementation, hierarchical approaches such as the one used by Parker et al. [1998] would reduce the complexity to $O(n \log n)$, and approaches based on finding the image-space intersections of occluding contours could reduce complexity further. Instead, we have implemented a simple approach for computing approximate visibility as follows. Rather than checking visibility for every vertex, we check every k-th vertex as we traverse a loop. Whenever there is a change in visibility between two tested vertices, we check the intervening vertices as well to determine where the transition occurred. If no transition occurs between two tested vertices, we assume the intervening vertices have the same visibility. In non-pathological cases this method reduces the algorithm's running time by a factor of at most k, possibly at the cost of a greater number of mislabeled vertices. One is still guaranteed not to miss details larger than k voxels, which may be acceptable for large data sets, especially where a large number of voxels can map to just one pixel. An evaluation of the impact of this algorithm is shown in Figure 6. The figure shows a closeup of a rendering from the aneurysm dataset, which we expect to be challenging for this algorithm because of the abundance of narrow occluders. Note that the sequences in the accompanying video were created without the approximate visibility algorithm.

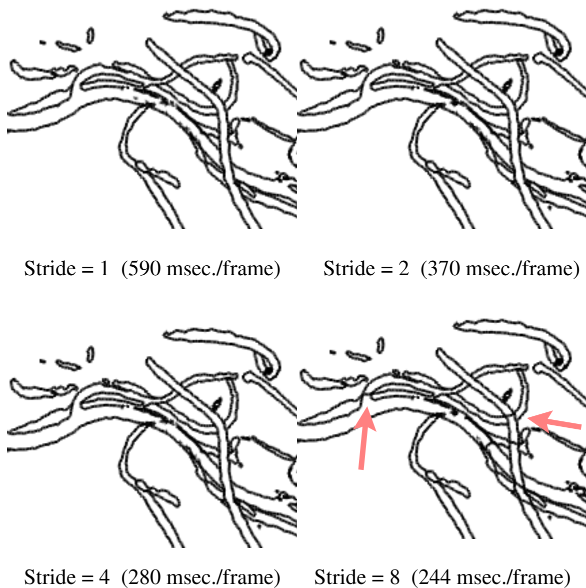


Figure 6: Effect of stride length k on approximate visibility algorithm. Note that the first three images are essentially identical (while resulting in a 2× speedup), while the last shows visible artifacts in areas with narrow occluders.

4.3 Stylization

Our system provides the user with full control over which kinds of lines are drawn, and in which style. In particular, the user can control the width and color of rendered lines based on any of the following attributes:

- the type of line (contour, suggestive contour, or cutting plane);
- the originating isosurface (primary or secondary);
- the result of the $s < 0$ test for finding "inside" contours;
- the result of visibility testing;
- the side of a cutting plane on which the segment lies; and
- the length of the loop from which a segment originates (since omitting short lines provides a simple form of detail elision).

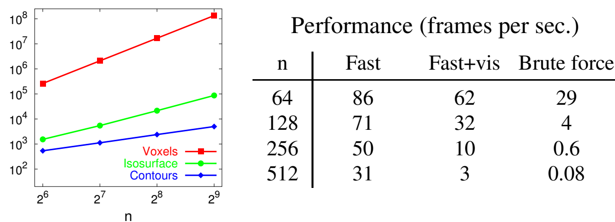
Alternatively, lines can be omitted based on any of these tests. The renderings throughout the paper demonstrate the stylistic flexibility afforded by these controls.

4.4 Results

Figures 1 and 8-10 show examples of line drawings made from data sets with a range of sizes from $150 \times 110 \times 60$ (simulation) to $512^2 \times 256$ (feet). These are drawn in a variety of styles to demonstrate rendering flexibility.

Figure 7 reports timing results for the stochastic search for silhouette loops as well as computing visibility on them. These data were collected as an average over a set of frames where the isosurface was swept through its meaningful range while the camera circled twice around the data – re-sampled versions of the aneurysm data set shown in Figure 8. To ensure that the datasets were band limited (as per the discussion in Section 3), we downsampled the original 256^3 dataset to 64^3, which was then upsampled to produce the 126^3, 256^3, and 512^3. Without hidden surface removal, the algorithm runs at interactive frame rates, even for a 512^3 dataset. These times were measured on a laptop with a 1.8 GHz. Pentium M processor and 1 GB of memory.

Computing suggestive contours takes more time, and, as reported in Figure 7, visibility takes even more time. Therefore our application has a mode (demonstrated in the accompanying video) wherein only contours are drawn during interaction (camera movement or sweeping the isosurface); when motion stops the full drawing with suggestive contours and hidden surface removal is rendered.



Performance (frames per sec.)

n	Fast	Fast+vis	Brute force
64	86	62	29
128	71	32	4
256	50	10	0.6
512	31	3	0.08

Figure 7: Experimental verification of $O(n)$ scaling for different-resolution versions of aneurysm dataset (Figure 8). *Left:* the total number of voxels grows cubically with dimension, the number of voxels on an isosurface grows quadratically, while the number of voxels on contours grows linearly. *Right:* the rendering performance of the fast seed-and-traverse algorithm, the same algorithm with visibility testing, and brute-force extraction. Interactive performance is maintained even for 512^3 datasets.

5 Conclusion and Future Work

We have described a method for creating line drawings from volumetric datasets by extracting linear features such as contours and suggestive contours directly from the data. Because such features are sparse, they offer the potential to be extracted more efficiently than by traversing the entire dataset. Furthermore they can produce more comprehensible renderings either by identifying important structural relationships (as exploited by previous research on traditional volume rendering that has emphasized silhouettes, for example) or by producing more spare drawings based on such lines alone. We have demonstrated both the efficiency and comprehensibility aspects of these algorithms by creating a variety of figures based on a variety of datasets using a working system. This project leads to several possible areas for future work, such as:

Efficient visibility: Visibility is the most expensive phase of the system we described, both in absolute terms and in terms of asymptotic complexity. As mentioned in Section 4.2, we believe that we could achieve an asymptotic improvement by using hierarchical methods to intersect a ray with the volume. Perhaps more interesting would be to find a way to adapt classical algorithms for hidden-line removal [Griffiths 1978] to our problem.

Enhanced stylization: Section 4.3 describes several ways that we use stylized rendering of the lines to aid comprehensibility. Nonetheless, there are a number of more sophisticated or abstract rendering algorithms one might try. In the spirit of [Kirby et al. 1999], there are strategies for conveying more information about the data or other scalar or vector values in the volume by further stylization of the lines. As another example, rather than drawing lines we could draw narrow “ribbons” that convey speed of motion with respect to changing either viewpoint or isovalue, producing an effect similar to motion blur. Finally, one might prefer to draw textured lines such as those drawn for silhouettes by Kalnins et al. [2003], which would then require investigating how to do so with temporal coherence when such lines are extracted from volumes.

Acknowledgments

The authors thank Gordon Kindlmann and Allen Sanderson for advice and assistance during the development of this project. We also thank Philip Shilane and Chris DeCoro for last minute proofreading before the submission deadline. The human CT, aneurysm, and the bonsai data sets are attributed to the NLM Visible Human Project, Philips Research, and Stefan Roettger, respectively, and we thank them for making such volumetric data sets publicly available. This work is partially supported by the National Science Foundation through grants HLC 0308121 and CCF 0347427.

References

BREMER, D., AND HUGHES, J. 1998. Rapid approximate silhouette rendering of implicit surfaces. In *Implicit Surfaces 98*, 155–164.

CIGNONI, P., MARINO, P., MONTANI, C., PUPPO, E., AND SCOPIGNO, R. 1997. Speeding up isosurface extraction using interval trees. *IEEE Trans. on Visualization and Computer Graphics 3*, 2 (Apr.), 158–170.

CSÉBFALVI, B., MROZ, L., HAUSER, H., KÖNIG, A., AND GRÖLLER, E. 2001. Fast visualization of object contours by non-photorealistic volume rendering. *Computer Graphics Forum 20*, 3, 452–460.

DECARLO, D., FINKELSTEIN, A., RUSINKIEWICZ, S., AND SANTELLA, A. 2003. Suggestive contours for conveying shape. *ACM Transactions on Graphics 22*, 3 (July), 848–855.

DECARLO, D., FINKELSTEIN, A., AND RUSINKIEWICZ, S. 2004. Interactive rendering of suggestive contours with temporal coherence. In *Third International Symposium on Non-Photorealistic Animation and Rendering (NPAR)*, 15–24.

DONG, F., CLAPWORTHY, G. J., LIN, H., AND KROKOS, M. A. 2003. Nonphotorealistic rendering of medical volume data. *IEEE Computer Graphics and Applications 23*, 4, 44–52.

EBERT, D., AND RHEINGANS, P. 2000. Volume illustration: non-photorealistic rendering of volume models. In *IEEE Visualization 2000*, 195–202.

GRIFFITHS, J. G. 1978. Bibliography of hidden-line and hidden-surface algorithms. *Computer-Aided Design 10*, 3, 203–206.

INTERRANTE, V., AND GROSCH, C. 1998. Visualizing 3d flow. *IEEE Computer Graphics and Applications 18*, 4, 49–53.

INTERRANTE, V., FUCHS, H., AND PIZER, S. 1995. Enhancing transparent skin surfaces with ridge and valley lines. In *Proceedings of the 6th conference on Visualization '95*, IEEE Computer Society, 52.

KALNINS, R. D., DAVIDSON, P. L., MARKOSIAN, L., AND FINKELSTEIN, A. 2003. Coherent stylized silhouettes. *ACM Transactions on Graphics 22*, 3 (July), 856–861.

KINDLMANN, G., WHITAKER, R., TASDIZEN, T., AND MOLLER, T. 2003. Curvature-based transfer functions for direct volume rendering: methods and applications. In *IEEE Visualization 2003*, 513–520.

KIRBY, M., MARMANIS, H., AND LAIDLAW, D. H. 1999. Visualizing multivalued data from 2D incompressible flows using concepts from painting. In *IEEE Visualization 1999*, 333–340.

KOENDERINK, J. J. 1984. What does the occluding contour tell us about solid shape? *Perception 13*, 321–330.

LIU, Z., FINKELSTEIN, A., AND LI, K. 2002. Improving progressive view-dependent isosurface propagation. *Computers & Graphics 26*, 2 (Apr.), 209–218.

LIVNAT, Y., SHEN, H.-W., AND JOHNSON, C. R. 1996. A near optimal isosurface extraction algorithm using the span space. *IEEE Trans. on Visualization and Computer Graphics 2*, 1 (Mar.), 73–84.

LORENSEN, W., AND CLINE, H. 1987. Marching cubes: A high resolution 3D surface construction algorithm. In *Proc. SIGGRAPH 1987*, 163–169.

LU, A., MORRIS, C. J., TAYLOR, J., EBERT, D. S., HANSEN, C., RHEINGANS, P., AND HARTNER, M. 2003. Illustrative interactive stipple rendering. *IEEE Trans. on Visualization and Computer Graphics 9*, 2 (Apr.), 127–138.

LUM, E. B., AND MA, K.-L. 2002. Hardware-accelerated parallel non-photorealistic volume rendering. In *Proceedings of the 2nd international symposium on Non-photorealistic animation and rendering (NPAR)*, ACM Press, 67–74.

MARKOSIAN, L., KOWALSKI, M. A., TRYCHIN, S. J., BOURDEV, L. D., GOLDSTEIN, D., AND HUGHES, J. F. 1997. Real-time nonphotorealistic rendering. In *Proceedings of SIGGRAPH 97*, Computer Graphics Proceedings, Annual Conference Series, 415–420.

NAGY, Z., AND KLEIN, R. 2004. High-quality silhouette illustration for texture-based volume rendering. In *Proc. WSCG*, V. Skala and R. Scopigno, Eds., 301–308.

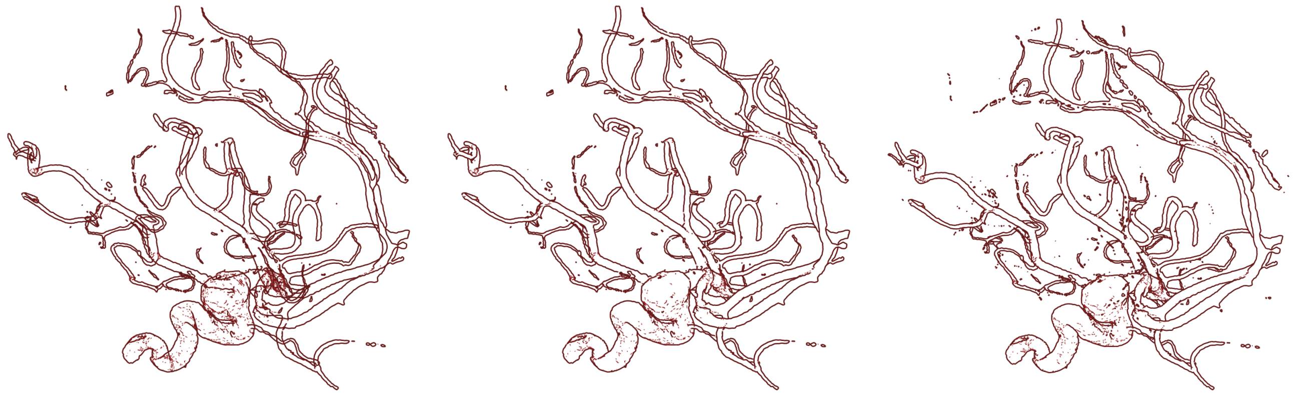


Figure 8: Aneurysm (256^3), *left to right:* without visibility; with visibility; using brute force for the search. Note that the stochastic search misses a few short, visible lines that are found by brute force search. If these features persist over time they will be found in future frames.

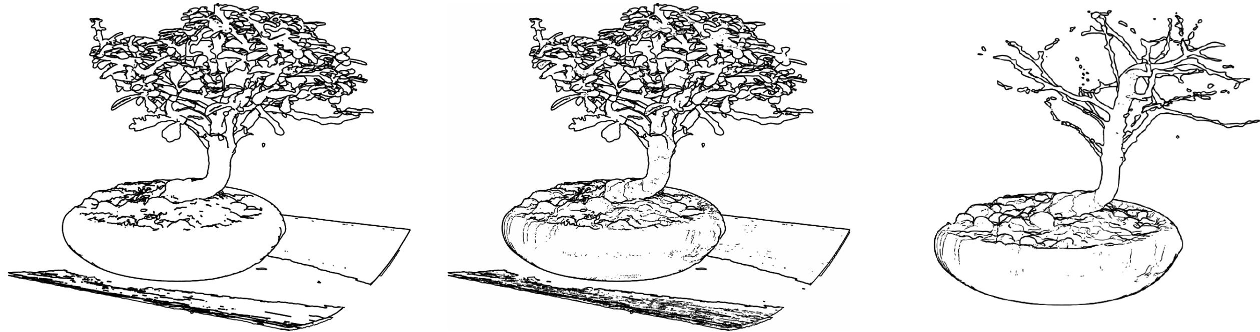


Figure 9: Bonsai (256^3), *left to right:* silhouettes alone; with suggestive contours; a different isosurface threshold.

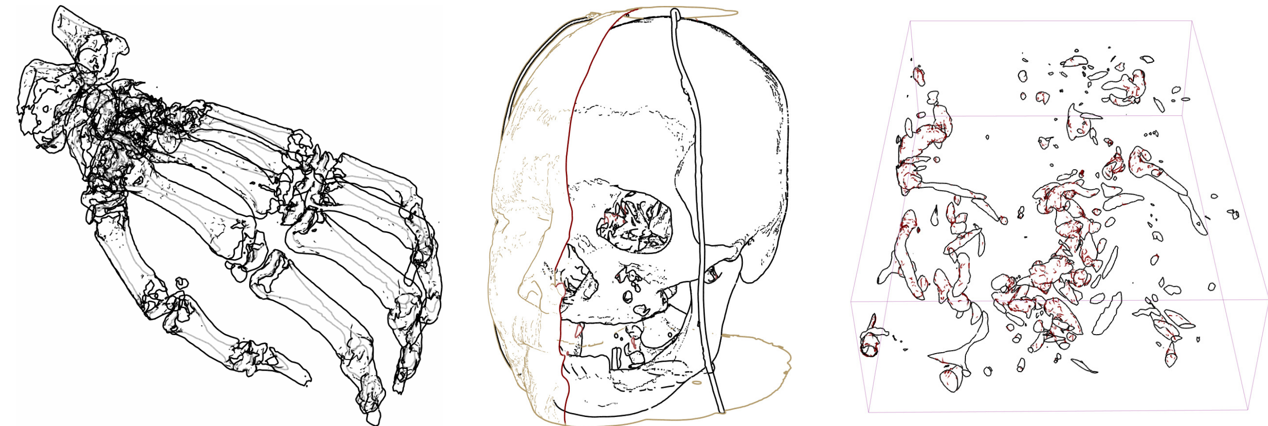


Figure 10: *Left to right:* Hand (150^3) without visibility draws at interactive rates and reveals hollow bones as thin backfacing contours; head ($512^2 \times 209$) with skin, bone and cutting plane; hypersonic turbulent flow simulation ($150 \times 110 \times 60$).

NAGY, Z., SCHNEIDER, J., AND WESTERMANN, R. 2002. Interactive volume illustration. In *Proc. Vision, Modeling and Visualization Workshop*.

PARKER, S., SHIRLEY, P., LIVNAT, Y., HANSEN, C., AND SLOAN, P.-P. 1998. Interactive ray tracing for isosurface rendering. In *IEEE Visualization '98*, 233–238.

SCHEIN, S., AND ELBER, G. 2004. Adaptive extraction and visualization of silhouette curves from volumetric datasets. *Vis. Comput. 20*, 4, 243–252.

SVAKHINE, N. A., AND EBERT, D. S. 2003. Interactive volume illustration and feature halos. In *PG '03: Proceedings of the 11th Pacific Conference on Computer Graphics and Applications*, IEEE Computer Society, 347.

THIRION, J.-P., AND GOURDON, A. 1996. The 3d marching lines algorithm. *Graphical Models and Image Processing 58*, 6 (Nov.), 503–509.

TREAVETT, S. M. F., AND CHEN, M. 2000. Pen-and-ink rendering in volume visualization. In *VISUALIZATION '00: Proceedings of the 11th IEEE Visualization 2000 Conference (VIS 2000)*, IEEE Computer Society.

VAN KREVELD, M., VAN OOSTRUM, R., BAJAJ, C., PASCUCCI, V., AND SCHIKORE, D. 2004. Chapter 5: Contour trees and small seed sets for isosurface generation. In *Topological Data Structures for Surfaces*, John Wiley & Sons, Ltd, Reading, Massachusetts, S. Rana, Ed., 71–86.