Nonphotorealistic Rendering of Medical Volume Data

Feng Dong, Gordon J. Clapworthy, Hai Lin, and Meleagros A. Krokos
De Montfort University

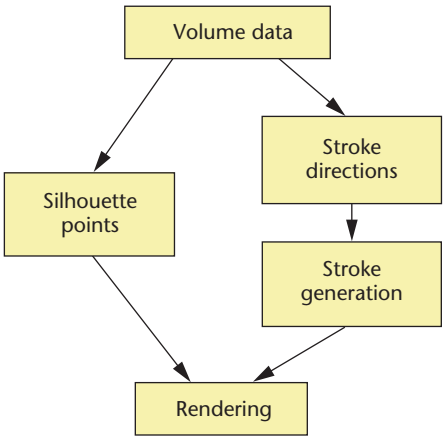
Volumetric hatching produces pen-and-ink illustrations suited for medical illustration. This approach accounts for data beneath the surface, producing images showing the subject's shape and reflecting its character.

Nonphotorealistic rendering, which produces images in the manner of traditional styles such as painting or drawing, is proving to be a useful alternative to conventional volume or surface rendering in medical visualization. Typically, such illustrations use simple lines to demonstrate medical shapes and features, omitting a great deal of detail. They frequently highlight the most relevant information better than glossy, colored images. Medical illustration has long regarded pen and ink as an important medium. Thus, because medical professionals are familiar with pen-and-ink illustrations, they could readily accept them as an alternative to conventional rendering of medical images.

Several authors, such as Treavett and Chen,[1] have begun using NPR in medical visualization (see the "Related Work in Nonphotorealistic Rendering" sidebar for a discussion of other visualization techniques). The general approach has been to apply existing surface-hatching techniques to surface models created from the volume data by marching cubes[2] or a similar method. Surface visualization, however, might not be well suited to portraying soft tissues, which are difficult to model with isosurfaces.

This article introduces volumetric hatching, a novel technique that produces pen-and-ink-style images from medical volume data. Unlike previous approaches that generate full-surface models, our technique uses the characteristics of the volume near the stroke being produced to generate a local intermediate surface. Because global isosurfaces can't exactly model many medical subjects, our volume-based method has considerable advantages. Our method is largely insensitive to surface artifacts. We focus on hatching with line strokes to portray muscles, intestines, brains, and so on. Hatching with line strokes requires determining not just the position of the line strokes, but also their orientation. Thus, the strokes not only illustrate the subject's shape, but also describe its character in some way—for example, by displaying fiber orientations for muscles.

Volumetric hatching can't replace conventional medical visualization. Rather, it can supply an alternative description that can enhance medical understanding. It might be particularly useful with the vector-based images of functional anatomy, where it could both describe motion sequences more compactly and focus on specific medical features demonstrated in the animation.



1 Overview of volumetric hatching. Volume data is the input for producing silhouette points and strokes, which feed into the rendering module.

Volumetric hatching

Figure 1 shows the volumetric hatching pipeline. Volume data is a 3D grid comprising a number of 2D images. A set of eight adjacent volumetric data points (VDPs)—sample points in the volume data—makes up a volume cube, the basic unit of volume data.

The silhouette, or outline of the subject, provides information about the subject—enough that some objects are recognizable from their silhouettes. The silhouette changes if the subject is viewed from a different direction.

Silhouette computation identifies a set of 3D points—

 Published by the IEEE Computer Society

Related Work in Nonphotorealistic Rendering

Work related to volumetric hatching includes surface hatching, volume data hatching, and more general uses of nonphotorealistic rendering in medical visualization. Although NPR can take many forms, we concentrate on pen-and-ink illustrations.

Surface hatching

Surface hatching in the pen-and-ink style illustrates a 3D surface using strokes instead of colors, with the hatching occurring on the surface.

A crucial problem in surface hatching is defining stroke direction to best illustrate surface shape and features. Although some authors have suggested isoparameter lines, in general, the two directions of principal curvature appears to be the favored approach.[1,2] The main obstacles to fast surface hatching are silhouette detection and visibility computation. Although many approaches address these problems, none stands out as a definitive solution.

We can apply surface hatching to many surface models, including parametric and implicit surfaces. To date, however, relatively few researchers have applied pen-and-ink illustration to 3D volume data. Treavett and Chen generated pen-and-ink images via volume rendering of 3D medical data sets.[3] They performed 3D drawing, which generates 3D strokes as textures in object space and then projects them on the image plane during rendering, and 2.5D rendering, which forms 2D strokes in 2D image space using information gained from the 3D object during prerendering.

Volume data hatching

Interrante used strokes to illustrate a surface shape within volume data.[2] She defined 3D scan-converted textures by computing directions associated with particles predistributed on the subject; she suggested the directions of principal curvature as the best candidates for stroke directions. Interrante then applied these textures to the volume to increase the opacity during the rendering stage. When displaying a transparent surface, the strokes help enhance surface shape. This technique is useful for visualizing layered isosurfaces in volume data.

NPR in medical visualization

Saito proposed an NPR-related method to preview volume data in real time.[4] He collected sample points uniformly from an isosurface and projected them on the image plane as geometric primitives such as lines. Because the primitives' orientation relies on the isosurface's local geometry, the method is restricted to isosurfaces.

Girshick and colleagues used principal-direction line drawings to show curvature flow over the surface. The technique aims at surfaces represented by 3D volume data or a polygon surface mesh.[5]

Lu and colleagues presented an interactive direct-volume illustration system that simulates stipple drawing.[6] They explored several feature-enhancement techniques for creating effective, interactive visualizations of scientific and medical data sets, and introduced a mechanism for generating appropriate point lists for all resolutions.

Researchers have believed for some time that NPR could be useful in medical visualization. Levoy and colleagues proposed NPR in radiation treatment planning.[7] More recently, Interrante and colleagues enhanced transparent skin surfaces with ridge and valley lines.[8] Ebert and Rheingans used volume illustration—basically a feature-enhanced form of volume rendering—to highlight features such as boundaries, silhouettes, and depth and orientation cues.[9] The results were not pen-and-ink illustrations.

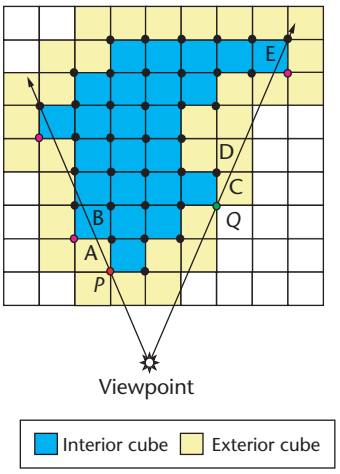
Other methods have enhanced object contours within the data.

References

1. G. Elber, "Line Art Illustrations of Parametric and Implicit Forms," *IEEE Trans. Visualization & Computer Graphics*, vol.4, no.1, 1998, pp. 71-81.
2. V. Interrante, "Illustrating Surface Shape in Volume Data via Principal Direction-Driven 3D Line Integral Convolution," *Proc. Siggraph*, ACM Press, 1997, pp. 109-116.
3. S.M.F. Treavett and M. Chen, "Pen-and-Ink Rendering in Volume Visualization," *Proc. IEEE Visualization*, IEEE CS Press, 2000, pp. 203-210.
4. T. Saito, "Real-Time Previewing for Volume Visualization," *Proc. Symp. Volume Visualization*, IEEE Press, 1994, pp. 99-106.
5. A. Girshick et al., "Line Direction Matters: An Argument for the Use of Principal Directions in 3D Line Drawings," *Proc. NPAR 2000: First Int'l Symp. Nonphotorealistic Animation & Rendering*, ACM Press, 2000, pp. 43-52.
6. A. Lu et al., "Nonphotorealistic Volume Rendering Using Stippling Techniques," *Proc. IEEE Visualization 2002*, IEEE CS Press, 2002, pp. 211-218.
7. M. Levoy et al., "Volume Rendering in Radiation Treatment Planning," *Proc. 1st Conf. Visualization in Biomedical Computing*, IEEE CS Press, 1990, pp. 4-10.
8. V. Interrante, H. Fuchs, and S. Pizer, "Enhancing Transparent Skin Surfaces with Ridge and Valley Lines," *Proc. IEEE Visualization,* IEEE CS Press, 1995, pp. 52-59.
9. D. Ebert and P. Rheingans, "Volume Illustration: Nonphotorealistic Rendering of Volume Models," *Proc. IEEE Visualization*, IEEE CS Press, 2000, pp. 195-202.

silhouette points—that pass to the rendering module to generate silhouette lines in the final image. During silhouette computation, we collect 3D silhouette points instead of lines. We project the points on the final image and generate 2D lines connecting the projections during rendering. We have had poor results with the alternative approach of tracing lines from one volume cube to another to obtain 3D lines and projecting these lines during rendering.

A direction field finds the overall stroke orientations. The direction field gives the stroke direction at each VDP and thus dictates how the hatching will occur. Computation or user interaction determines the direction field to be used.



2 Finding silhouette points. Because the two cubes (C and D) immediately following *Q* along the line are outside the subject, *Q* is a VDP. Because cube B is inside the subject, *P* is not a VDP.

The stroke generation module generates the 3D strokes following the directions indicated by the direction field. The illumination defined for 3D strokes filters the generated strokes, and their projections contribute to the final image. This provides visual coherence—if we move the viewpoint, we need only reproject the 3D strokes on a new image plane, thus maintaining consistency between images.

Detecting silhouette points

The silhouette conveys the most important information about a subject's shape, and might be its simplest portrayal. The silhouette depends on the viewpoint and exists only at the subject's visible boundary. We consider only VDPs for silhouette points because medical data volume cubes are quite small and therefore a set of VDPs represents silhouettes with sufficient accuracy.

To find silhouette points, we first mark the VDPs and volume cubes. VDPs belonging to the subject are *in*, and those outside the subject are *out*, based on the data segmentation. We further categorize *in* VDPs as follows:

- *Boundary point*. The VDP is on the subject's boundary—that is, it's neighboring an *out* VDP.
- *Inner point*. The VDP's neighbors are all *in*.

We categorize the volume cubes as follows:

- *Interior cube*. All eight VDPs are *in*.
- *Exterior cube*. At least one VDP is *out*.

Because these categorizations don't change with the viewpoint, we perform them only once.

During silhouette detection, we search only among the boundary points in the exterior cubes. Because these form a small fraction of the total VDPs, searching is fast.

To check whether a VDP is a silhouette point, we cast a view line from the viewpoint toward the VDP. If the line pierces any cubes belonging to the subject before it reaches the VDP, the VDP can't be a silhouette point. If it doesn't, we check the two cubes immediately following the VDP along the line, such as cubes A and B after the VDP *P* in Figure 2. If neither cube belongs to the subject, the VDP is a silhouette point. In Figure 2, *Q* is a silhouette point because cubes C and D are outside the subject; *P* is not a silhouette point.

This process ensures that a concave subject is dealt with properly. A view line could hit an interior cube positioned on the farther branch of the concavity—for example, the line through *Q* strikes interior cube E. The resulting image will show one part of the subject silhouetted against the other part.

In practice, it's more efficient to check the volume cubes along the view line from *P* to the viewpoint. So, we check the volume cube neighboring *P* first, then the next cube toward the viewpoint, and so on. This way, we don't have to find the first volume cube along the view line, which is typically quite time consuming.

Drawing the silhouette

To draw the silhouette, we project the 3D silhouette points on the final image. During projection, we check each point's visibility. Because many of the silhouette points come from the same volume cubes and are connected in the 3D volume, we can form silhouette lines by connecting the points' projections using straight lines.

Because we can project many silhouette points close to each other in the image, the projections can be dense in areas, and we might create many unnecessary lines. To overcome this problem, we remove some silhouette points in the dense areas before connecting the projections.

We define two thresholds to identify areas in which there are too many projections:

- A float number, *dist*, defines a minimum distance between projections. We use the volume cube's size as the unit of measure to keep *dist* independent of sample size.
- An integer, *Neigh*, defines *next*(*P*, *Neigh*), a set of VDPs that are close to silhouette point *P*.

If $Neigh = 1$, the set $next(P, 1)$ consists only of *P*'s neighboring VDPs. If $Neigh = 2$, then $next(P, 2)$ includes, in addition to *P*'s neighbors, the neighbors of the VDPs in $next(P, 1)$. We likewise extend the definition for larger values of *Neigh*.

Given a fixed *dist* and *Neigh* for each silhouette point *P*, we consider the set of silhouette points that are in *next*(*P, Neigh*) and discard the points that project within a distance *dist* of the projection of *P*. After removing these points, we join the remaining points using straight lines to create the silhouette. Removing the points can create some gaps in the silhouette. Increasing *dist* and *Neigh* removes more unnecessary lines, but creates larger gaps. Typically, *dist* is between 0 and 1 and *Neigh* is between 1 and 3. Because silhouette detection and drawing is quick, users can readily experiment with these values.

Determining stroke directions

Determining stroke directions relies on both the subject (for instance, to portray muscles, medical artists typically use strokes indicating muscle fiber direction) and the individual, as different artists have their own hatching styles. Hence, designing a

general algorithm to automatically define stroke directions for any subject is difficult. Rather, stroke direction decisions should depend on the character of the subjects.

For muscles, stroke orientation must follow the direction of the muscle fibers. In earlier work, we describe a method for detecting muscle fiber orientation from volume data.[3] The process involves quickly estimating an approximate fiber orientation for each VDP and refining it into a more accurate direction. We then associate each VDP inside the muscle volume with a direction indicating the fiber orientation at that point. For muscles, we perform this process only for those VDPs within a prescribed distance of the muscle surface.

For other subjects, such as a human brain, strokes follow the direction of principal curvature, which we calculate from the volume data using Thirion and Gourdon's method.[4] They derived principal curvature formulas that use only partial differentials of the 3D volume; hence, we can compute the principal curvature directly from the volume without extracting a surface. The stroke lies along one of the principal directions associated with the principal curvature.

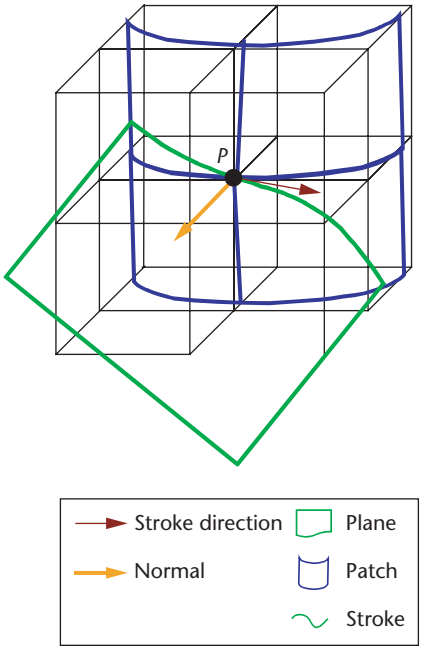
Both methods are very versatile—the principal-curvature method has been widely used to illustrate different subjects, and the other works on any subject with line textures.

To produce a hatching style for a particular subject, you should adopt an approach tailored to that subject. If this requires introducing a new approach, the rest of the algorithm will be unaffected, as hatching style is independent of other components.

Producing strokes

In volumetric hatching, unlike surface hatching, interior data make contributions. Some interior strokes are portrayed to improve rendering quality.

In general, to produce a stroke at a VDP, we fit a local surface patch approximating the geometry at the VDP and then intersect that patch with a normal plane following the stroke direction. The intersection of the patch and the plane defines the stroke.



3 Stroke generation. To produce a stroke, we intersect a linear surface patch with a plane and gradient at VDP *P*.

In Figure 3, we use a linear blue surface patch to estimate the geometry at VDP *P*. The patch intersects the green plane containing the stroke direction and gradient at *P* to produce the stroke. The final form of each stroke is a piecewise succession of straight lines across the patch's tessellated surface.

In this step, the main work is generating a useful patch. Simply creating an isosurface patch that passes through the VDP gives unsmooth results. To generate a smooth patch around the VDP, we fit the patch from the smoothed volume gradient. The sidebar "Smooth Patches for Generating Strokes" gives technical details.

Smooth Patches for Generating Strokes

Before generating a smooth patch at a volumetric data point (VDP), we must decide the patch's primary orientation so we can represent the smooth patch in terms of a height field.

The patch's primary orientation is the main direction the patch faces—*x*, *y*, or *z*. We determine its orientation by checking the gradient at the VDP. If the patch's *z* component is greater than its *x* and *y* components, its primary orientation is *z*.

Here, we assume the primary orientation is *z*, but you can apply the procedure equally to *x* or *y* orientations. The regular grid of the height field's 2D domain is then in the *xy* plane.

Creating an isosurface patch

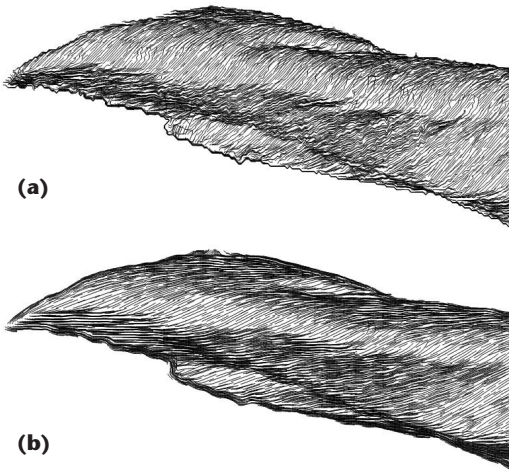
Because we describe the smooth patch as a height field, to create it we must find the discrete height values at grid points $G(x, y)$ in the 2D domain. From these, generating a mesh is straightforward.

At each $G(x, y)$, we compute the isosurface patch height, denoted $hiso(x, y)$, to produce the 3D points $S(x, y, hiso)$. We then obtain the normals to the isosurface by calculating the gradient at points $S(x, y, hiso)$ using linear interpolation. The normal n associated with grid point $G(x, y)$ is denoted $n(x, y)$.

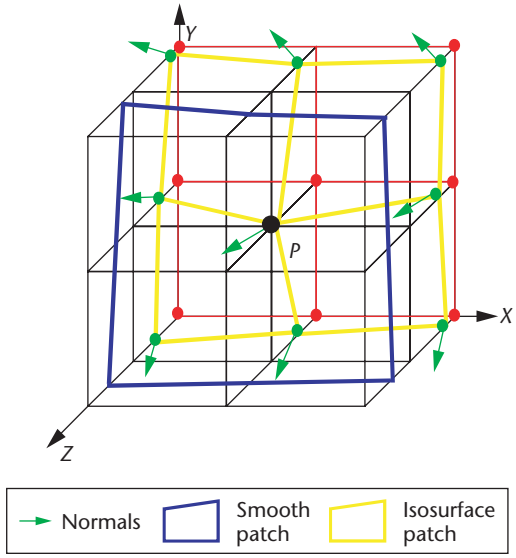
We first generate an isosurface patch local to the VDP, using the gradients on the isosurface to estimate its surface normals. Because these gradients are smooth, they provide a good foundation for generating the smooth patch.

Fitting a Smooth Patch

We fit a smooth patch such that the normals on the patch are close to those obtained in the previous step. We regard $n(x, y)$ as the normal of the smooth patch at $T(x, y, hsmo)$, so we can expand the *xyz* components of $n(x, y)$ as linear combinations of *hsmo*. We find height values *hsmo* for the grid points $G(x, y)$ using least squares fitting, thus defining the 3D points $T(x, y, hsmo)$. Once we've found all the $hsmo(x, y)$, making the surface patch is simple.



4 Volumetric smoothing of an image (a) without smoothing and (b) with smoothing.



5 Fitting a smooth patch. Fitting the yellow isosurface patch gives us the blue smooth patch.

Figure 4 shows the volumetric hatching of an image with and without smoothing.

Figure 5 shows how we would fit a smooth patch at VDP *P*. The yellow polygon is the isosurface patch at *P*, and the green arrows are surface normals generated on the isosurface patch. We fit the smooth patch from these normals. The red grid is the height field domain.

A height field $h(x, y)$ gives a height value for each point (x, y) in a 2D domain. It therefore defines a set of 3D points $(x, y, h(x, y))$. Obviously, we can easily make a mesh from the height field by joining these 3D points.

Strokes are produced only at VDPs. If users require more strokes to build their desired tones, we can insert more data points into the cubes (via trilinear interpolation) and produce more strokes from these points. If users prefer a lighter tone, they can filter out some of the strokes. Another alternative for tone building is illumination, as described in the next section.

In Figures 3 and 5, computing strokes at *P* uses $2 \times 2 \times 2$ neighboring cubes. In practice, if you prefer longer strokes, you can use more neighboring cubes. We've often used $4 \times 4 \times 4$.

Rendering

During the rendering stage, the 3D silhouette points and strokes are presented in a 2D final image. This involves several processes: illuminating the strokes, determining the contribution of the strokes to the final image, and drawing the silhouette.

Stroke illumination. Lighting is fundamental to providing a 3D feel to a subject. In pen-and-ink illustration, including more or fewer strokes in an area can produce darker or lighter intensities. Thus, adjusting the number of strokes that pass through an area controls the intensity associated with that area.

We apply a volumetric illumination method based in object space—that is, we calculate the volume cubes' lighting intensity. We linearly convert each cube's lighting intensity to the number of strokes in the cube. If the intensity is less than the number of existing strokes, we reduce the number of strokes in the cube until it corresponds to the lighting intensity at the cube.

Because the illumination occurs in the object space and results in filtered strokes, we can reuse the strokes even if we reposition the viewpoint, as long as the viewing distance and lighting sources remain approximately unchanged. The sidebar "Calculating Stroke Illumination" provides further details.

VDP contribution. A VDP's contribution to the final image is the projection of its associated stroke. As in volume rendering, exterior VDPs occlude the contributions of interior VDPs. We therefore only consider contributions from the interior VDPs near the subject's surface—that is, those within the user-defined distance depth of the surface. During volume data segmentation, identifying these VDPs, which form a set called the *shell,* is straightforward. We need strokes only at the VDPs within the shell.

In volume rendering, opacity controls the visibility of internal structures. Likewise, volumetric hatching presents only the data within a certain distance beneath the subject's exterior surface, as controlled by parameter *depth*. A proper *depth* choice lets users portray subject parts that lie just below the surface, but nevertheless influence the subject's appearance, while excluding subject parts that lie deep inside the subject. Because this part of the process is fast, we can manually adjust the *depth* value during rendering.

We further classify interior cubes as

- shell cubes, which have all eight VDPs in the shell, and
- core cubes, which have at least one VDP not in the shell.

To calculate the contribution of a VDP in the shell, given the viewpoint and image plane position, we again use a view line. If the line doesn't encounter any core cubes before it reaches the VDP, the VDP is visible and its stroke is projected. In Figure 6, the view line toward *P* hits only the shell cube A before it reaches *P*, so *P* is a visible point and its associated stroke contributes to the final image. If the line hits a core cube before it reaches the VDP, the VDP is invisible and doesn't contribute to the

Calculating Stroke Illumination

To perform stroke illumination, we first convert the intensity of a volume cube into the number of strokes in the cube. We define *cube intensity* as the average lighting intensity at the cube's eight VDPs. If we normalize the range of cube intensity values to [0, 1], cube intensity is related to the stroke number by

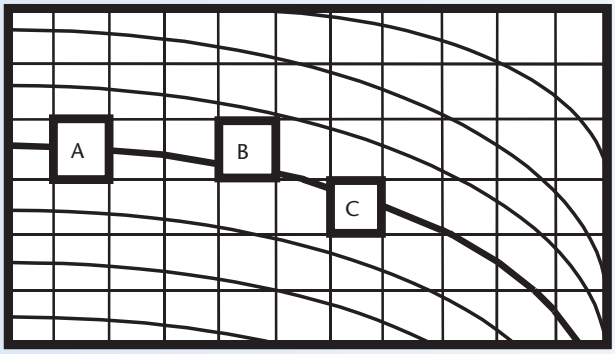
$$StrokesNum = (1 - \text{cube intensity}) \times \text{ratio} + \text{base} \quad (1)$$

where *ratio* defines the relation between the intensity and the number of strokes, and *base* is the ambient tone, which controls the number of strokes at the brightest cube. Increasing *ratio* gives greater contrast. We typically set *base* between 0 and 3.

We apply this illumination model to the strokes we've created. The strokes are fairly evenly distributed because we generate a stroke at each VDP. If we obtain the average number of strokes in the volume cubes before illumination and regard this as the number of strokes at the darkest cube (that is, where cube intensity is 0), we calculate *ratio* as *Ratio* = average stroke number – *base*.

The illumination process then removes strokes from the volume cubes in which cube intensity is greater than 0. The number to be removed at a cube is equal to the difference between the number of strokes before illumination and number of strokes at the cube, which we determine from Equation 1.

If the number of strokes in the volume cube is larger than *StrokesNum* (overtoned), we remove strokes to reduce the number of strokes at the cube. That is, we select a stroke and remove the segment that lies within the cube. In practice, it doesn't matter which stroke we cut, so we cut them randomly.



A 2D illustration of stroke removal.

If we select a stroke for removal using this method, we check the cubes through which it passes. If any of them are overtoned, we cut the stroke from it, too. Thus, if a stroke is cut from a volume cube, it becomes the first candidate to be cut from other volume cubes—in Figure A, the bold stroke is cut from cube A and also from the overtoned cubes B and C, through which it passes. This lessens the number of strokes to be cut and reduces the proliferation of small, scattered line segments.

When we reposition the viewpoint, we don't need to recompute the illumination if the distance between the subject and the viewpoint remains nearly unchanged. If the subject moves closer, however, we need more strokes. We create extra strokes inside the volume cubes by generating extra data points. If the subject moves away from the observer, we use a smaller ratio to generate a lighter tone.

final image (for example, VDP *Q* in Figure 6 is invisible).

For speed, we perform the calculation in reverse direction—from the cube neighboring the VDP toward the viewpoint—as in silhouette detection.

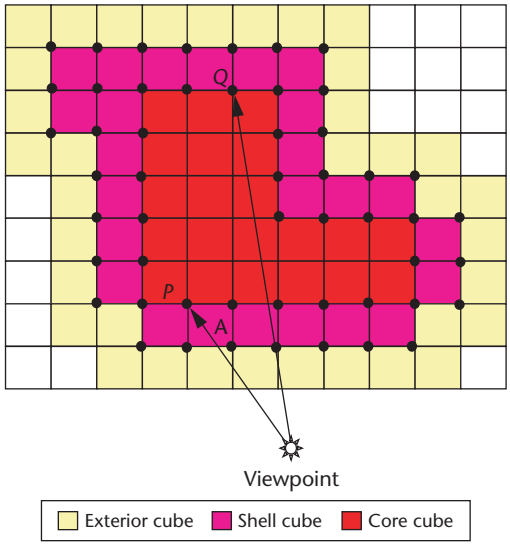
Experimental results

We've applied volumetric hatching to various sets of medical data, including segmented muscles from visible human data, a human brain from a magnetic resonance imaging data set, and part of the human digestive system from a cat scan data set.

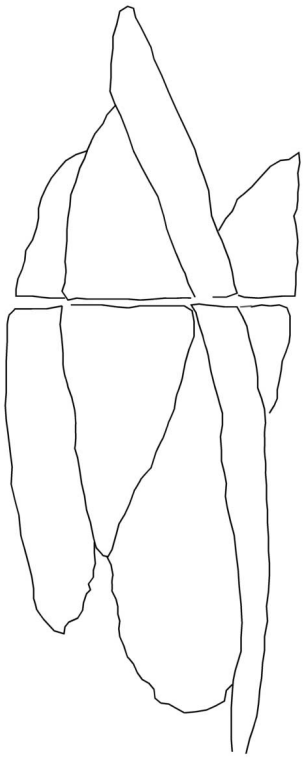
Figure 7 (next page) gives the results of a silhouette computation applied to a muscle data set; the silhouette lines display the subject in a simple form. The data set is $190 \times 162 \times 500$ pixels. Figure 8 compares our method for creating silhouettes with directly projecting 3D silhouette lines on the final image. Figure 8a shows a result from 3D silhouette lines, while Figure 8b shows the results for our method, with parameters *Neigh* = 2, *dist* = 0.6. Figure 8b looks much better, as Figure 8a has too many unnecessary lines.

Figure 9 illustrates the muscles at the front of an upper leg. As we segmented the data, we removed muscles that were not to be displayed from the data. We calculated strokes using $4 \times 4 \times 4$ neighboring cubes and a depth of 16 for the rendering shell, because the effects of surface muscle fibers penetrate a few volume cubes beneath the surface.

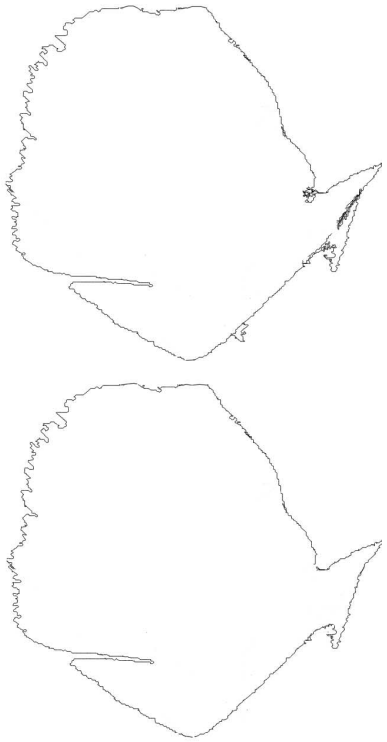
In Figure 9a, the ratio of the cube's lighting intensity to the number of strokes in the cube is smaller than in



6 Computing VDP contribution. Because the view line hits only the shell cube A before reaching *P*, *P* is a visible point and its associated stroke contributes to the image.

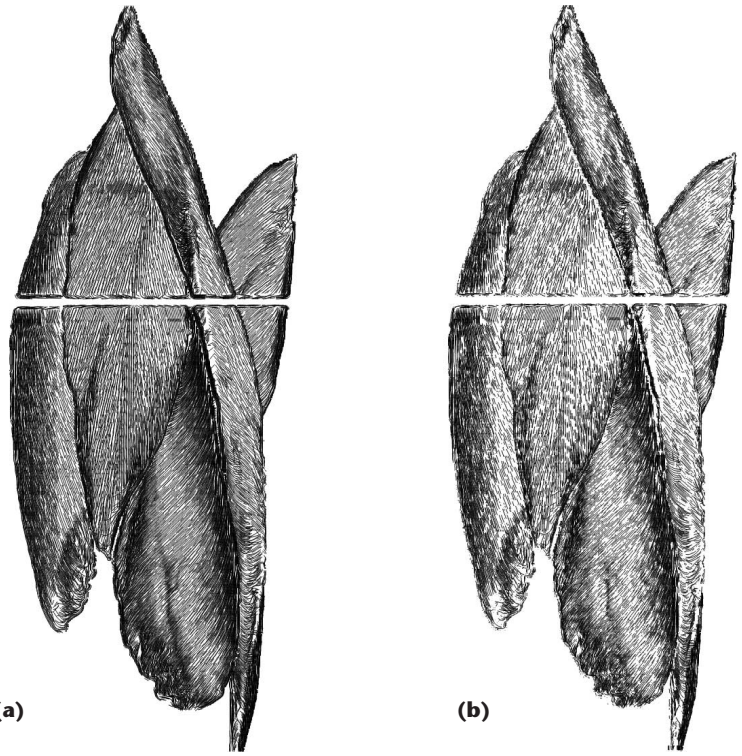


7 Muscle data presented using silhouettes. The silhouette lines display the basic outline of the subject.



(a) **(b)**

8 Comparison of silhouette generation methods on medical data: (a) 3D silhouette lines and (b) volumetric hatching using silhouette points.



(a) **(b)**

9 Volumetric hatching of a human leg using different ratio of the lighting intensity to the number of strokes. Because the ratio in (a) is smaller than in (b), (b) has greater contrast.

Figure 9b, giving the image greater contrast.

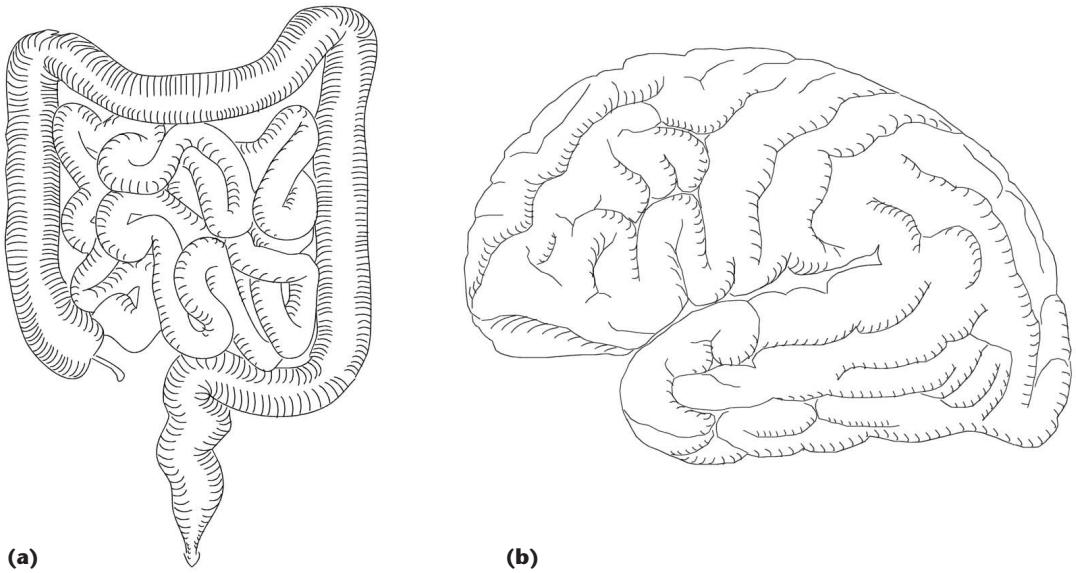
Figure 10a shows part of the human digestive system, and Figure 10b is a human brain. Because illustrations of these organs rarely use many strokes, we set a small value to the intensity–stroke ratio to reduce the number of strokes, and set base (the ambient tone) to 0. We calculated strokes using principle curvature directions and $2 \times 2 \times 2$ neighboring cubes. Because we needed a light tone and were interested only in VDPs close to the surface, we used a rendering shell depth of 1.

Table 1 gives the computational times associated with some images in Figures 9 and 10. As the table shows, stroke generation is the most computationally expensive part of the process. Fortunately, because a user can store strokes, subsequent reexamination of the data wouldn't require regenerating them, making viewing much faster.

Rendering consists of two parts: the first corresponds to illumination calculation, and the second to VDP projection. Although illumination calculation is rather time consuming (see the "Calculating Stroke Illumination" sidebar), the results are reusable as long as the viewing distance and lighting sources remain approximately unchanged, in which case the user could skip this part of the process. Part 2, VDP projection, is relatively fast. Although it must be performed each time the viewpoint changes, the rerendering time is acceptable as long as the viewing distance and lighting sources don't change greatly.

Volumetric hatching is also efficient in terms of storage because the images are based purely on the projection of 3D strokes, and thus can be stored in vector form.

Table 2 compares the storage requirements of images in vector and raster forms, both compressed and uncompressed. Storing the images in vector form saves a lot of



(a) (b)

10 Volumetric hatching examples: (a) human digestive system from a cat scan data set and (b) human brain from a magnetic resonance imaging data set.

space, even though the raster images are not very large. The space required for images stored in raster form increases with the size of the image, but doesn't change for images stored in vector form.

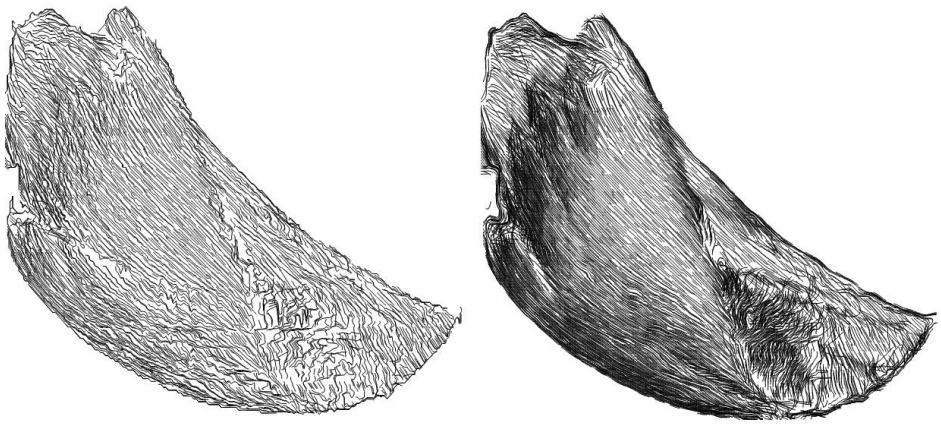
Because volumetric hatching deals directly with volume data, it differs greatly from most existing techniques, which are based on surface hatching. As we mentioned at the start of this article, a straightforward approach to hatching volume data is to generate isosurfaces from the data using marching cubes and then apply standard surface-hatching techniques. Figure 11 compares this method with our volumetric hatching. Hatching on the isosurface generated a poor result (Figure 11a) compared with volumetric hatching (Figure 11b). We applied the same lighting and stroke illumination to the surface and volume. Stroke illumination failed to generate a good result for the strokes embedded on the surface. The result from volumetric hatching is more impressive because the volumetric strokes (including those underneath the surface) better describe the subject.

Table 1. Timing results for volumetric hatching.

Figure	Data Size	Stroke Generation (seconds)	Rendering (seconds) Part 1	Part 2
9a	$190 \times 162 \times 500$	305	52	5
9b	$190 \times 162 \times 500$	305	63	2
10a	$120 \times 82 \times 300$	78	13	2
10b	$150 \times 102 \times 200$	82	15	2

Table 2. Storage requirements comparison.

Figure	Size (pixels)	Raster Form (Kbytes) Uncompressed	Raster Form (Kbytes) Compressed	Vector Form (Kbytes) Uncompressed	Vector Form (Kbytes) Compressed
9a	319×783	734	213	240	116
9b	319×783	734	206	220	102
10a	326×444	424	49	42	21
10b	467×317	434	45	43	21



(a) (b)

11 Comparison of (a) surface hatching with (b) volumetric hatching. Volumetric strokes better describe the subject, resulting in a more defined image.

Conclusions and future work

The silhouette computation technique still requires improvement. The current method depends too much on the sample distance of vol-

ume data. Because we choose the silhouette points in a discrete space, errors can't be ignored if the sample distance increases.

To date, the images we've produced have been static. A possibility for future work is to consider visual coherence, particularly in animated sequences. Because we treat illuminated strokes as 3D objects, the techniques have a built-in visual coherence. If the viewpoint moves much closer, however, we will have to generate more strokes in the focused area to retain the required detail.

Another limitation is that volume hatching only works with segmented volume data. Because line strokes are designed to indicate a subject's shape, we must identify the subjects before hatching can occur.

Pen-and-ink illustration using line strokes is just one of many NPR styles used in medical illustrations and books. Thus, our future work will focus on a more general approach incorporating many NPR styles. ■

Acknowledgments

The European Commission, within the MultiMod project no. IST-2000-28377 and the Chinese Natural Science Foundation, award no. 60003009, supported the work presented in this article.

References

1. S.M.F. Treavett and M. Chen, "Pen-and-Ink Rendering in Volume Visualization," *Proc. IEEE Visualization*, IEEE CS Press, 2000, pp. 203-210.
2. W.E. Lorensen and H.E. Cline, "Marching Cubes: A High Resolution 3D Surface Construction Algorithm," *Computer Graphics,* vol. 21, no. 4, 1987, pp. 163-169.
3. F. Dong, G.J. Clapworthy, and M. Krokos, "Volume Rendering of Fine Details Within Medical Data," *Proc. IEEE Visualization*, IEEE CS Press, 2001, pp. 387-394.
4. J.P. Thirion and A. Gourdon, "Computing the Differential Characteristics of Isointensity Surfaces," *Computer Vision and Image Understanding*, vol. 61, no. 2, 1995, pp. 190-202.

Feng Dong *is a research fellow in computer graphics in the Department of Computer and Information Sciences, De Montfort University, UK. His research interests include fundamental computer graphics algorithms, medical visualization, volume rendering, human modeling, and virtual reality. Dong received a PhD in computer science from Zhejiang University, China. He is a member of the UK Virtual Reality Special Interest Group (VRSIG).*

Gordon J. Clapworthy *is a professor of computer graphics in the Department of Computer and Information Sciences, De Montfort University, UK. His research interests include medical visualization, computer animation, biomechanics, virtual reality, surface modeling, and fundamental computer graphics algorithms. Clapworthy received a PhD in aeronautical engineering from the University of London. He is a member of the ACM, ACM Siggraph, Eurographics, and the UK-VRSIG, and is secretary of the British Chapter of the ACM.*

Hai Lin *is a research fellow in computer graphics in the Department of Computer and Information Sciences, De Montfort University, UK. His research interests include medical visualization, volume rendering and virtual reality. Lin received a PhD in computer science from Zhejiang University, China.*

Meleagros A. Krokos *is a research fellow in computer graphics in the Department of Computer and Information Sciences, De Montfort University, UK. His research interests include computer-aided geometric modeling of curves and surfaces, medical visualization, and virtual reality. Krokos was educated at the University of London. He is a member of the IEEE Computer Society, ACM Siggraph, and the UK-VRSIG.*

Readers may contact Feng Dong at the Dept. of Computer and Information Sciences, De Montfort Univ., UK, MK7 6HP; fdong@dmu.ac.uk.