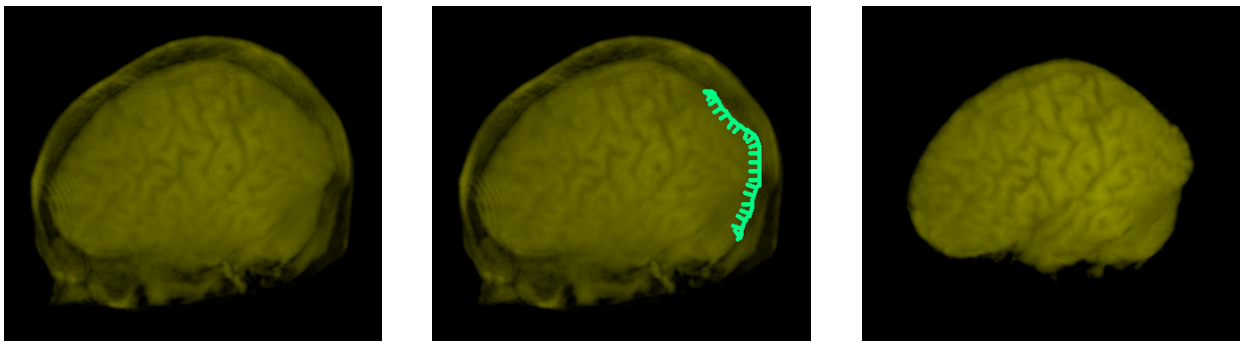
Volume Catcher

Shigeru Owada*
The University of Tokyo / Sony CSLabs, Inc.

Frank Nielsen†
Sony CSLabs, Inc.

Takeo Igarashi‡
The University of Tokyo / JST PRESTO



(a) (b) (c)

Figure 1: (a) $105 \times 73 \times 78$ head MRI data. (b) Drawing a 2D stroke along the contour of the brain. (c) Resulting 3D region. The system automatically computes the depth of the stroke and applies constrained segmentation.

Abstract

It is difficult to obtain a specific region within unsegmented volume data (region of interest, ROI). The user must first segment the volume, a task which itself involves significant user intervention, and then chooses a desired target within the 3D space. This paper proposes a simple and intuitive user interface for the task: the user traces the contour of the target region using a 2D free form stroke on the screen, and the system instantly returns a plausible 3D region inside the stroke by applying a segmentation algorithm. The main contribution is that the system infers the depth information of the ROI automatically by analyzing the data, whereas existing systems require the user to provide the depth information explicitly. Our system first computes the 3D location of the user-specified 2D stroke based on the assumption that the user traced the silhouette of the ROI, that is, the curve where the gradient is perpendicular to the viewing direction. The system then places constraint points around the 3D stroke to guide the following segmentation. Foreground constraints are placed inside the stroke and background constraints are placed outside the stroke. We currently use the statistical region-merging algorithm of Nock et al. [Nock and Nielsen 2004a] to perform the segmentation. We tested our system with real-world examples to verify the effectiveness of our approach.

CR Categories: D.2.2 [Software Engineering]: Design Tools and Techniques—User interfaces I.4.6 [Image Processing and Computer Vision]: Segmentation—Pixel Classification

Keywords: User interface, Volume graphics, Segmentation

*e-mail: sowd@acm.org
†e-mail: frank.nielsen@acm.org
‡e-mail: takeo@acm.org

1 Introduction

Volume segmentation is the process of splitting volume data into several perceptual or semantic units. It is a fundamental procedure that is required to obtain useful information from the volume region, such as its shape, topology, and various measurements (cubic volume, number of components, etc.). The importance of volumetric segmentation has been widely recognized for about thirty years and many sophisticated segmentation algorithms have been proposed. However, no fully automated method is yet available because segmentation is dependent on the observer's subjective interpretation, which is impossible to obtain without user intervention. Most segmentation methods have focused on low-level features, such as edge detection and texture analysis, and have achieved some degree of success. The difficulty is in high-level recognition that is related to the semantics of data. For example, suppose we have a scene that contains a bunch of grapes on a dish. Both "a bunch of grapes" and "a dish" are semantic elements, which may consist of more than one low-level feature. The user may want to carve out only one grape, or the entire bunch, or even the entire bunch along with the dish. These options are all probable, depending on the user's intent. Therefore, it is crucial for the user to give appropriate guiding information to get the desired segmentation result. From the user's perspective, one problem with 3D volumetric segmentation is that 3D guiding information is difficult to specify, as the typical input device is a mouse, which provides only 2D information. Recent work tried to deal with this problem by allowing the user to draw strokes on the cross-section of volume data that roughly indicate foreground and background regions [Tzeng et al. 2003]. This stroke information is used to train a classifier that is designed for segmenting voxels. One drawback of this system is that it requires much user interaction: the user has to specify the cutting

plane and provide several strokes to train the classifier. We propose a simple interface for specifying a region of interest. The user simply delineates the ROI on the rendered image using a 2D free-form stroke. The system automatically computes the missing depth information and returns a volumetric region inside the stroke by performing segmentation inside the stroke. The user no longer needs to specify foreground and background constraints in 3D space manually. This paper describes the user interface and the implementation details of our prototype system. We also present some segmentation examples using real-world datasets.

2 Previous work

Image segmentation has received much attention and has been explored intensively in the 2D domain. Many different approaches have been proposed, including thresholding, k-means clustering, deformable models, watershed segmentation, graphcut algorithms, level-set methods, and the Hough transform. We will not review all existing methods here. An interested reader should refer to the survey paper [Pham et al. 1999]. Instead, we will explain a few promising approaches. The most common algorithms optimize partitions of weighted neighborhood graphs [Yu and Shi 2004; Felzenszwalb and Huttenlocher 1998; Shi and Malik 2000]. These graph-partitioning problems can be solved either locally using fast greedy decision heuristics [Felzenszwalb and Huttenlocher 1998; Nock and Nielsen 2004a] or globally by computing decompositions [Yu and Shi 2004; Shi and Malik 2000] of matrices induced from these graphs. Segmentation algorithms designed primarily for 2D data are often applicable to 3D with little or no modification. The most frequently used volume classification tool, which is a superset of volume segmentation, is the transfer function [Pfister et al. 2001]. This technique is explored mainly in the context of volume visualization. Traditionally, a transfer function maps the original voxel values to color and opacity values directly [Drebin et al. 1988]. Recent work generates transfer functions that capture larger structures, such as higher-order gradients [Kniss et al. 2001] and topology [Takahashi et al. 2004].

Unfortunately, no completely automatic image segmentation algorithm is possible because segmentation depends on the semantic interpretation of an image, which is very difficult for computers to emulate. Therefore, recent systems seek to incorporate human control into the segmentation task. Examples of such systems for 2D imaging are the Lazy Snapping system [Li et al. 2004] and the Crayons system [Fails and Olsen 2003]. It is especially difficult to control 3D segmentation because the typical input device is a 2D mouse. The most reliable method is to isolate a slice of the volume and specify the contour of the ROI manually [Hastreiter and Ertl 1998]. This information is then propagated to adjacent slices using a region-growing technique. Alternatively, the user may place seed points for region growing on the cross-sectional plane [Sherbondy et al. 2003]. These techniques require a fair amount of user interaction. Setting transfer functions is indirect from the viewpoint of user interaction [He et al. 1996; Pfister et al. 2001]. Recently, Tzeng et al. presented a user interface for providing high-level classification information that involves drawing freeform strokes on a cross-sectional plane roughly [Tzeng et al. 2003]. The user cuts the data perpendicular to each axis and specifies foreground and background regions using a painting tool. By observing the gradient, location, and neighboring voxel values, as well as the voxel value itself, this system captures local texture and positional information on a voxel. Nock and Nielsen describe a fast, provably good, region-merging algorithm [Nock and Nielsen 2004b] based on the statistical analysis of regions. Their method easily generalizes [Nock and Nielsen 2004a] to incorporate user-defined constraints and is directly applicable to 3D.

3 User interface

We first describe the system from the user's perspective. After the user loads a volumetric model, the system renders it using a traditional volume-rendering method. Here, the user can apply any rendering technique to enhance the appearance of the model. Currently, our system allows the user to modify opacity-transfer, color-transfer, and gradient-enhancement functions [Lichtenbelt et al. 1998]. The user can change the view direction, scale, and these transfer functions interactively, to locate a target ROI in the volume.

To select a ROI, the user simply traces at least a part of the contour of the ROI on the screen using a 2D freeform stroke (Figure 1b). We currently assign dragging while pressing the left mouse button to draw freeform strokes. The current implementation requires that the user trace the contour in a clockwise direction. In other words, the area to the right of the stroke's drawing direction is recognized as the target region. The system shows hatching along the stroke to indicate this constraint. The width of the hatched region roughly indicates the allowable error margin. The system automatically computes the depth of the stroke so that the stroke is on a region boundary in the 3D space and applies segmentation based on the fact that the right hand side of the 3D stroke is inside the region. Finally, the system returns a volumetric region as either voxels in the volume (Figure 1c) or a boundary surface computed using the Marching Cubes algorithm [Lorensen and Cline 1987] (Figure 7d,f). The algorithm is based on the assumption that the user's stroke traces the contour of the ROI closely. Consequently, it may fail if the stroke is far from the contour or if the contour is too fuzzy.

Our current system supports two types of strokes: open and closed strokes. If the distance between the two ends is close (< 20 pixels in our current implementation), the system automatically connects the end points and processes it as a closed stroke. Otherwise, the stroke is treated as open. Closed strokes are useful when the entire boundary is visible and open strokes are useful when the boundary is too long or partially occluded.

4 Algorithm

4.1 From 2D freeform stroke to 3D path

Our algorithm first converts the 2D stroke on the screen into a 3D path by adding depth information. We assume that the ROI is distinct visually and that the user follows its contour. If this assumption is valid, the 3D path should be close to the silhouette of the ROI. In other words, the gradient vector of the volume data near the 3D stroke should be almost perpendicular to the view direction. We find such a 3D path as follows:

1. Sweep the 2D freeform stroke along the depth direction to create a 3D curved surface (Figure 2a). The sweep extent is set to cover the entire volume. This curved surface is called the *sweep surface*. In our implementation, this surface is represented by a piecewise planar surface (quadrangular polygons).

2. Parameterize the sweep surface. The system assigns the x coordinate axis to the depth direction and the y coordinate axis to the direction parallel to the stroke on the screen (Figure 2b). That is, $x = 0$ along the curved edge of the sweep surface near

the camera and $y = 0$ along the straight edge corresponding to the starting point of the stroke.

3. Set sampling points on the sweep surface. Sampling points are lattice points in the parameter space. We set the interval of the lattice as 0.3% of the diameter of the bounding sphere for the volume data[1]. In the following context, each lattice point is denoted by L_{ij} (i, j are indices along the x and y directions in the parameter space where $1 \leq i \leq X_{max}, 1 \leq j \leq Y_{max}$.).

4. On each lattice point L_{ij}, compute $S_{ij} = | N_{ij} \cdot G_{ij} |$ where N_{ij} is a unit normal vector of the surface while G_{ij} is a unit volume gradient (Figure 2c). S_{ij} is called a *silhouette coefficient*, and it indicates how much the point looks like a silhouette from the current viewpoint.

   The volume gradient G_{ij} is computed as follows. If the original data contain color channels, the system first converts the color values into grayscale values that represent the perceptual brightness of the colors using the standard equation $Gray = Alpha \times (0.299 \times Red + 0.587 \times Green + 0.114 \times Blue)$ [Russ 2002]. If the volume renderer filters the original data (e.g., using the opacity and color transfer functions), the post-filtered color should be used because we assume that the user wants to select a visually distinct structure in the current rendered image. Furthermore, to handle the user's imprecise input stroke and to suppress the effect of noise, the system computes G_{ij} from a blurred version of the original data. Any kind of blur filter can be used. We chose a discrete approximation of the Gaussian filter. The radius R of the filter in the data's coordinate system is matched to the extent of the stroke's error margin in the screen coordinate system[2]. The stroke's error margin is set as five pixels in our implementation. To blur and differentiate the data simultaneously, we convolve the data with the derivative of a blur filter. We construct our blur/differential filter kernel by first computing a blur filter and taking the forward difference along the x direction. The result can be used immediately for the y and z directions by rotating it by 90 degrees.

5. The problem of computing the depth of the user-drawn 2D stroke is now the problem of finding a path in the parameter space that starts from a point on the edge $y = 1$ and ends at a point on the edge $y = Y_{max}$, where the sum of the silhouette coefficients along the path is maximized (Figure 2d). As this path has only one x value for each y, it is represented as a function $x = f(y)$. To maintain the continuity of the path, we impose the condition that $| f(y_i) - f(y_{i+1}) | \leq c$, where c is an integer constant value that controls the continuity of the path. We currently use $c = 1$. We introduce an additional condition of $| f(y_1) - f(y_{max}) | \leq c$ for a closed stroke. We use dynamic programming to solve this problem [Cormen et al. 2001]. For an open stroke, the dynamic programming is performed only once while for closed stroke, it must be performed for each point on the edge $y = 1$, to satisfy the additional condition $| f(y_1) - f(y_{max}) | \leq c$. The optimal path on the parameter space is then converted into a 3D path by connecting the corresponding 3D lattice points (Figure 2e). The matrix S_{ij} and optimal path for Figure 1 are shown in Figure 3.

[1] Strictly speaking, this parameterization is not uniform if we use perspective transformation. The space is slightly stretched in the y direction where $x = 1$.

[2] In perspective projection, R should depend on the depth. However, we use a constant value computed at the center of the voxels.

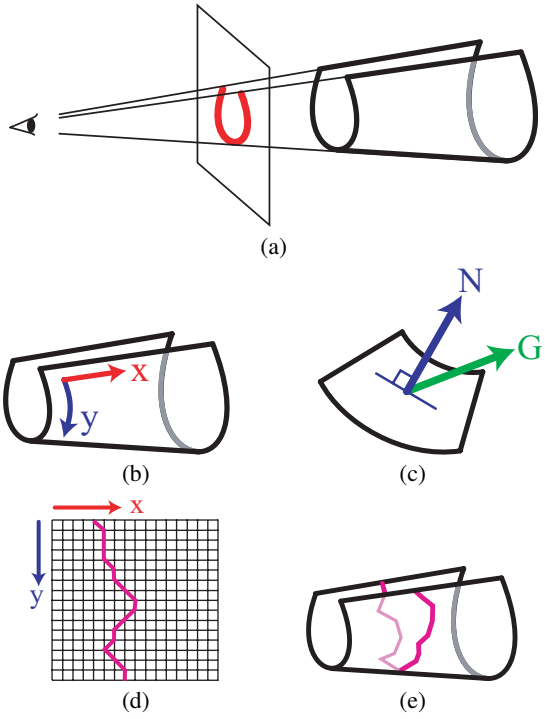


Figure 2: From a 2D stroke to a 3D path. (a) Construction of the sweep surface. (b) Parameterization of the sweep surface. (c) Computation of the silhouette coefficient. (d) Finding the optimal path. (e) The resulting 3D path.

4.2 Generating constraints and segmentation

The next task is to generate constraints to guide the segmentation. We currently use the region-merging algorithm of Nock et al. [Nock and Nielsen 2004a] for segmentation, which takes a set of constraint points that specify foreground and background regions as input and separates the volume into foreground and background regions that contain the corresponding constraints. The constraints can be used directly for other segmentation algorithms, such as those that use the Graphcut technique [Li et al. 2004]. We generate the constraints by offsetting the 3D path. The offset direction, D, is obtained simply by computing a cross-product of the view vector and the tangent vector of the path (Figure 4a). The displacement extent e is proportional to the radius of the blur kernel R computed in the previous section; therefore, it is also proportional to the stroke's error margin ($e = 2R$, in our implementation). Each point in the 3D path is offset by $\pm e\frac{D}{|D|}$ and the points on the right hand side become foreground constraints and those on the left hand side become background constraints (Figure 4b). The constraints generated for Figure 1 are shown in Figure 5.

We perform the actual volume segmentation using these constraints. If the user-given stroke is a closed stroke, only those voxels inside the sweep surface are returned.

5 Results

We applied our technique to several datasets. Figure 6 shows an example of a high-potential iron protein dataset. Note that our system works even when only part of the target region is visible (Figure 6b). We also applied our system to color data (Figure 7). We

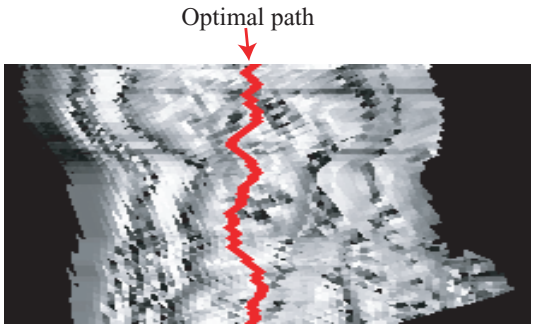


Figure 3: Silhouette coefficient matrix and computed optimal path for Figure 1.

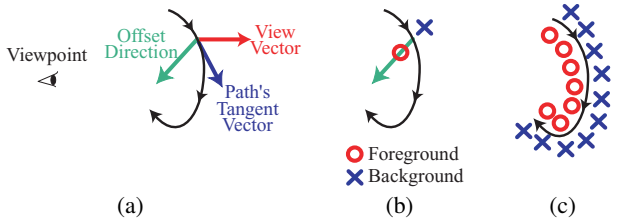


Figure 4: Constraint locations

captured this full color volumetric data using the slicer system developed by Ogawa et al. [Ogawa et al. 1999] (Figure 8). This slicer can cut a $3cm \times 5cm \times 2cm$ object frozen in OCT compound or a paraffin-embedded object into $10\mu m$ thick slices. The green matter in Figure 7 is OCT compound and the slice interval for the chocolate crisp data is $42\mu m$ thick.

To evaluate the speed of our system, we asked three subjects to use it. Their task was to select various desired regions 20 times for two datasets (high-potential iron protein($66 \times 66 \times 66$) and head MRI($105 \times 73 \times 78$)), respectively. Figure 9 is the chart that shows the time required to convert a 2D stroke into a 3D path and to generate constraints. The horizontal axis represents the number of sample points on the sweep surface, while the vertical axis represents the elapsed time in milliseconds. We used a desktop computer with an Intel Xeon(TM) 3.2-GHz processor and 2 GB of RAM. The chart shows that the required time is almost proportional to the number of sample points for both open and closed strokes, although closed stroke requires running dynamic programming multiple times. We suppose it is because running time is dominated by the resampling cost on the sweep surface. This chart does not contain the time for segmentation, because it is strongly affected by which algorithm is used. In our case [Nock and Nielsen 2004a], it took 0.5 to 10 seconds, depending on the size of the data and the number of constraints.

6 Limitations and future work

Although our system works for many real-world examples, there are several cases in which our system does not work properly. An intrinsic limitation of our approach is that the system carves out only one object when two objects have almost the same contour from the current viewpoint (Figure 10a). Another limitation is that the generated constraints can be outside the ROI when the ROI is too thin or has a complicated boundary shape near the 3D path (Figure 10b). We believe that we can solve this by carefully analyzing

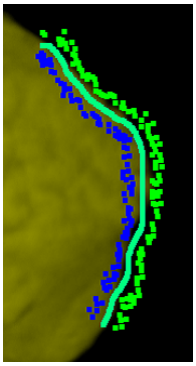


Figure 5: The constraints generated for Figure 1

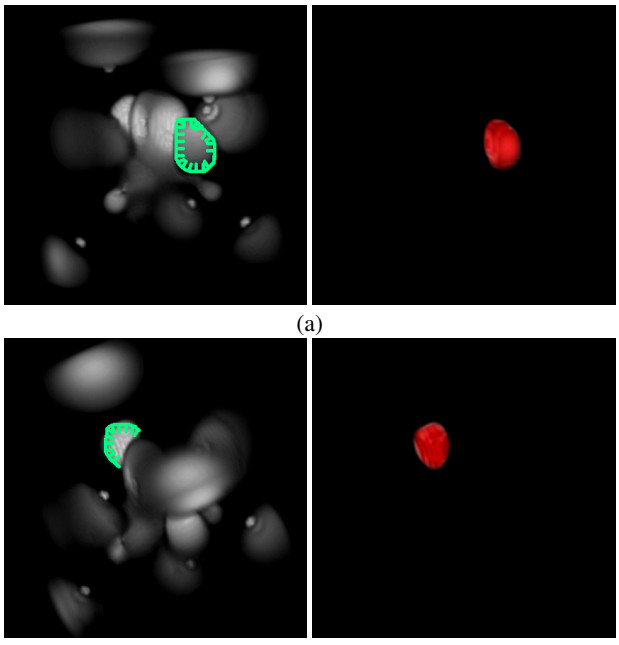


Figure 6: An application using 66^3 high-potential iron protein data. The segmented region is rendered as opaque, red voxels.

the structure around the path when placing the constraints. Finally, the performance of the system is dependent on the specific segmentation algorithm being used. For example, our current implementation fails when it is necessary to consider positional information for obtaining correct segmentation because our current segmentation algorithm [Nock and Nielsen 2004a] only considers color information (Figure 10c).

Our current implementation is very basic and we plan to extend it in various ways. First, we plan to extend it to adjust transfer functions automatically. When the user traces the silhouette of very faint or partially occluded objects, the system can adjust the transfer function automatically to increase their visibility. This process hopefully reduces the user's "trial-and-error" loop [Pfister et al. 2001]. Second, we plan to experiment with other segmentation algorithms, such as snakes. A key issue is that we need to provide different information to the segmentation algorithm depending on the type. For example, snakes require an initial boundary, which we need to generate from the user-drawn contour. Other miscellaneous future work includes the implementation of a mechanism to fix partially

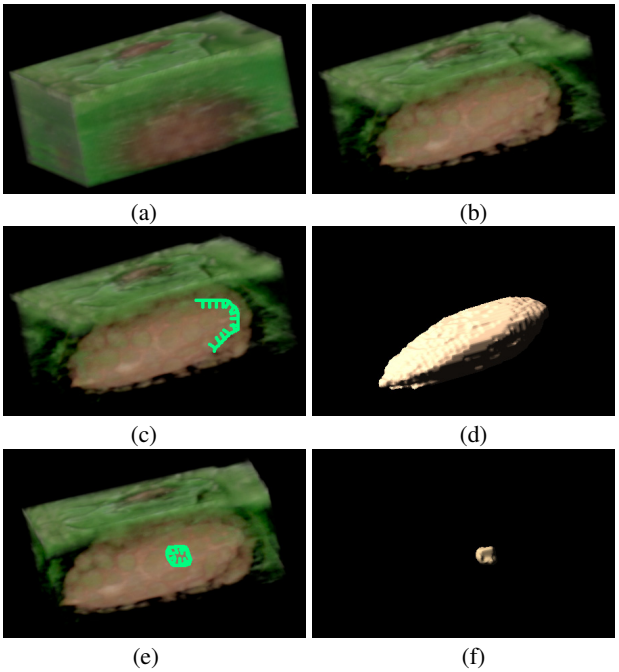


Figure 7: Application to a chocolate crisp dataset ($126 \times 89 \times 53$, color data). The opacity transfer function is applied to the original data (a) to distinguish the almond (b). When the user draws a stroke (c), the almond region is segmented and rendered as a surface (d). The surface color is set to the mean color of the selected region. The user can also pick small masses (e, f).

failed results, application to surface representations, and designing better user feedback.

7 Acknowledgements

It is difficult to obtain color volumetric data, other than the Visible Human Dataset [Banvard 2002]. Consequently, we are grateful to Taiki Kinoshita, Kenichi Kudoh, and Junichi Sugiyama for capturing the color chocolate crisp data. Ryo Haraguchi also provided his own MRI scan data. Tomer Moscovich helped us to check the draft on submission. We also thank Issei Fujishiro and Yutaka Ohtake for their useful suggestions and comments.

References

BANVARD, R. 2002. The visible human project® image data set from inception to completion and beyond. In *CODATA 2002: Frontiers of Scientific and Technical Data , Track I-D-2: Medical and Health Data*.

CORMEN, T. H., STEIN, C., RIVEST, R. L., AND LEISERSON, C. E. 2001. *Introduction to Algorithms*. McGraw-Hill Higher Education.

DREBIN, R. A., CARPENTER, L., AND HANRAHAN, P. 1988. Volume rendering. In *SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, ACM Press, 65–74.

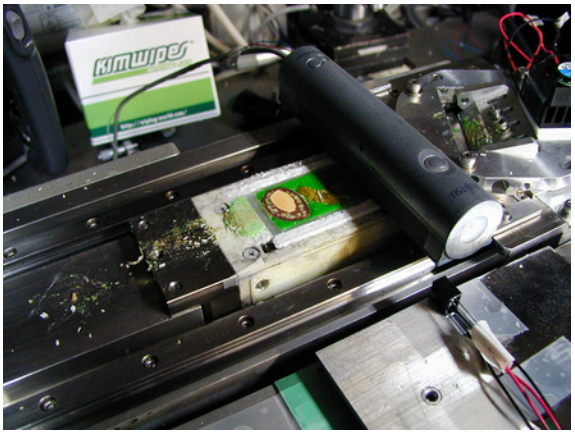


Figure 8: A cutting device

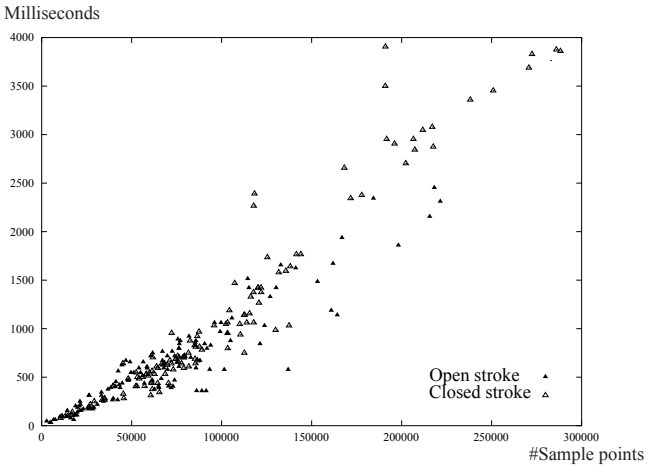


Figure 9: The time required to generate constraint points for the high-potential iron protein and head MRI datasets. This chart does not contain the time for segmentation.

FAILS, J., AND OLSEN, D. 2003. A design tool for camera-based interaction. In *Proceedings of the conference on Human factors in computing systems*, ACM Press, 449–456.

FELZENSZWALB, P., AND HUTTENLOCHER, D. 1998. Image segmentation using local variations. *IEEE Computer Vision and Pattern Recognition*, 98–104.

HASTREITER, P., AND ERTL, T. 1998. Fast and interactive 3D–segmentation of medical volume data. In *Proceedings of Workshop on Image and Multi-dimensional Digital Signal Processing (IMDSP)*, H. Niemann, H.-P. Seidel, and B. Girod, Eds., 41–44.

HE, T., HONG, L., KAUFMAN, A., AND PFISTER, H. 1996. Generation of transfer functions with stochastic search techniques. In *Proceedings of the 7th conference on Visualization '96*, IEEE Computer Society Press, 227–234.

KNISS, J., KINDLMANN, G., AND HANSEN, C. 2001. Interactive volume rendering using multi-dimensional transfer functions and direct manipulation widgets. In *Proceedings of IEEE Visualization '01*, IEEE, 255–262.

LI, Y., SUN, J., TANG, C.-K., AND SHUM, H.-Y. 2004. Lazy snapping. *ACM Trans. Graph. 23*, 3, 303–308.

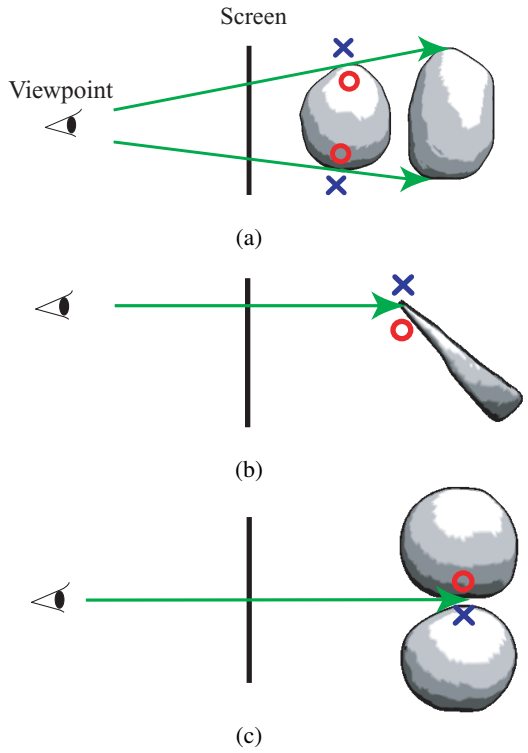


Figure 10: Failed examples. (a) Two regions have almost the same contour. (b) The target region is too thin. (c) A similar region is near the catching contour. In those figures, target regions are illustrated by boundary surfaces.

LICHTENBELT, B., CRANE, R., AND NAQVI, S. 1998. *Introduction to volume rendering*. Prentice-Hall, Inc.

LORENSEN, W. E., AND CLINE, H. E. 1987. Marching cubes: A high resolution 3d surface construction algorithm. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, ACM Press, 163–169.

NOCK, R., AND NIELSEN, F. 2004. Grouping with bias revisited. In *IEEE International Conference on Computer Vision and Pattern Recognition*, IEEE CS Press, A. B. L.-S. Davis, R. Chellapa, Ed., 460–465.

NOCK, R., AND NIELSEN, F. 2004. Statistical region merging. *IEEE Trans. Pattern Anal. Mach. Intell. 26*, 11, 1452–1458.

OGAWA, Y., OHTANI, T., SUGIYAMA, J., HAGIWARA, S., KOKUBO, M., KUDOH, K., AND HIGUCHI, T. 1999. Three dimensional visualization of internal constituents in a rice grain. *ASAE/CSAE-SCGR Annual International Meeting*, 993059.

PFISTER, H., LORENSEN, B., BAJAJ, C., KINDLMANN, G., SCHROEDER, W., AVILA, L. S., MARTIN, K., MACHIRAJU, R., AND LEE, J. 2001. The transfer function bake-off. *IEEE Comput. Graph. Appl. 21*, 3, 16–22.

PHAM, D. L., XU, C., AND PRINCE, J. L. 1999. A survey of current methods in medical image segmentation. In *Technical Report JHU/ECE 99-01, The Johns Hopkins University*.

RHEINGANS, P., AND EBERT, D. 2001. Volume illustration: Nonphotorealistic rendering of volume models. *IEEE Transactions on Visualization and Computer Graphics 7*, 3, 253–264.

RUSS, J. C. 2002. *The Image Processing Handbook Fourth Edition*. CRC Press.

SHERBONDY, A., HOUSTON, M., AND NAPEL, S. 2003. Fast volume segmentation with simultaneous visualization using programmable graphics hardware. In *Proceedings of IEEE Visualization 2003*, IEEE, 171–176.

SHI, J., AND MALIK, J. 2000. Normalized cuts and image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell. 22*, 8, 888–905.

TAKAHASHI, S., TAKESHIMA, Y., AND FUJISHIRO, I. 2004. Topological volume skeletonization and its application to transfer function design. *Graphical Models 66*, 1, 24–49.

TZENG, F.-Y., LUM, E. B., AND MA, K.-L. 2003. A novel interface for higher-dimensional classification of volume data. In *Proceedings of IEEE Visualization 2003*, IEEE, 505–512.

YU, S. X., AND SHI, J. 2004. Segmentation given partial grouping constraints. *IEEE Trans. Pattern Anal. Mach. Intell. 26*, 2, 173–183.