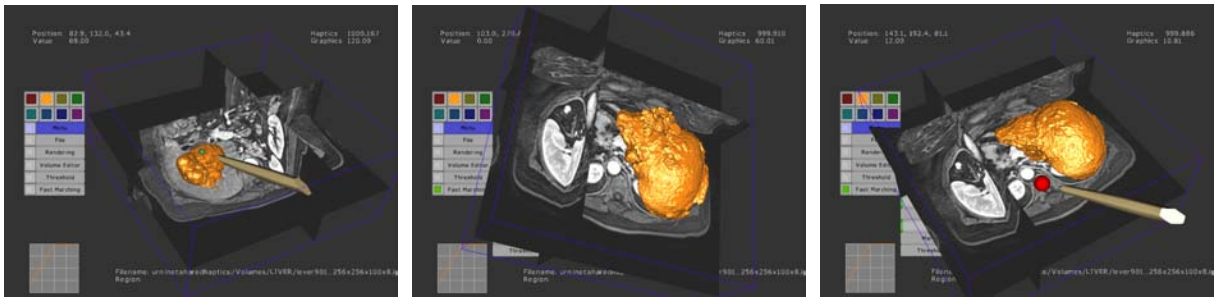
Fast surface rendering for interactive medical image segmentation with haptic feedback

Erik Vidholm* Jonas Agmund†
Centre for Image Analysis
Uppsala University



(a) Initialization of a seed-region inside the liver.

(b) Initial segmentation result that contains artifacts due to the low contrast between the liver and the surrounding tissues.

(c) Result after editing.

Figure 1: Screenshots from the interactive segmentation environment.

Abstract

In this work, we present a haptic-enabled application for interactive editing in medical image segmentation. We use a fast surface rendering algorithm to display the different segmented objects, and we apply a proxy-based volume haptics algorithm to be able to touch and edit these objects at interactive rates. As an application example, we show how the system can be used to initialize a fast marching segmentation algorithm for extracting the liver in magnetic resonance (MR) images and then edit the result if it is incorrect.

CR Categories: I.3.6 [Computer graphics]: Methodology and techniques—Graphics data structures and data types, Interaction techniques. I.4.6 [Image processing and computer vision]: Segmentation.

Keywords: marching cubes, surface tracking, volume haptics

1 Introduction

Image segmentation is the task of finding a certain object in an image and label all the voxels inside the object as foreground, and all other voxels as background. In medical segmentation, objects should be extracted from different data sets obtained through, e.g., Computed Tomography (CT) or Magnetic Resonance Imaging (MRI). An object to be segmented could typically be a part of the brain or the liver. Even though many methods have been proposed for automatic segmentation, it is still seen as an unsolved problem since the methods are not general enough. In semi-automatic methods, some degree of manual interaction is involved to improve the result. Ideally, the user should give an initialization to the algorithm and then examine the final result and if necessary edit it. The efficiency of this interactive part is highly dependent on the quality of the user interface. The user needs to be provided with proper tools for the specific task, and the learning threshold should not be too high. When working with volume images it is a huge step just to map interaction in 2D to events in 3D. By using more advanced input devices combined with different depth cues (e.g., stereo), it is possible to overcome this problem.

Interactive editing and manipulation of volume data for design and modeling purposes has been referred to as *sculpting* by previous authors. In general, a sculpting system consists of a set of modeling tools together with fast surface rendering and/or haptic rendering algorithms for data display. In [Galyean and Hughes 1991], a sculpting system with various free-form tools was developed. An octree-based system was proposed in [Bærentzen 1998] where "spray-tools" and constructive solid geometry (CSG) tools were used. The use of haptic feedback in volume sculpting was suggested already in [Galyean and Hughes 1991], but realized first in the work described in [Avila and Sobierajski 1996]. In the recent paper [Kim et al. 2004], a combined geometric/implicit surface representation is used along with tools for haptic painting based on texture techniques. The connection between haptic volume sculpting and interactive volume image segmentation is close, but not much work has been done in this area. Haptic interaction was used by [Harders and Székely 2002] for centerline extraction during segmentation of

*e-mail: erik.vidholm@cb.uu.se
†e-mail: jonas.agmund.4911@student.uu.se

tubular structures, and in [Vidholm et al. 2004] haptic feedback was used to facilitate the placement of seed-points in MR angiography data sets for vessel segmentation. Examples of non-haptic interactive segmentation tools for volume images that have inspired our work are found in [Kang et al. 2004].

In this paper, we propose the use of editing tools based on morphological image processing operators in combination with haptic feedback, stereo graphics, and a fast surface rendering algorithm to interactively edit and manipulate segmented data. Haptics provides the possibility of simultaneous exploration and manipulation of data. In our work, realistic feedback is not the most important issue. More important for us is that the user works more efficiently with guidance by haptic feedback than without. The aim is to considerably reduce the amount of user time required in the segmentation process.

The paper is organized as follows: In Section 2, we give an overview of our visuo-haptic environment for interactive segmentation. A brief description of the volume visualization based on 3D texture mapping is given in Section 3. In Section 4, we present the fast surface renderer and some implementation issues. Section 5 gives an introduction to volume haptics and describes how we use haptic feedback for editing. An example application is given in Section 6 and in Section 7 we present our results. Finally, we summarize the paper with conclusions and future work in 8.

2 System overview

In this section, we give an overview of our environment and the interactive segmentation application.

2.1 Hardware and software

Our setup consists of a Reachin desktop display [Thurfjell et al. 2002] which combines a 3 degrees of freedom (DOF) PHANToM desktop haptic device with a stereo capable monitor and a semi-transparent mirror to co-locate graphics and haptics. See Figure 2. The workstation we use is equipped with dual 2.4 GHz Pentium 4 processors and 1GB of RAM. The graphics card is a NVidia Quadro 900XGL with 128MB of RAM. For stereo display, Crystal Eyes shutter glasses are used. The software has been implemented in the Reachin API, a C++ API that combines graphics and haptics in a scene-graph environment based on the VRML97 standard.

Figure 2: The Reachin desktop display.

2.2 Interactive segmentation

Most of the user interaction is performed with the PHANToM device through 3D widgets and volume editing tools. A Magellan space mouse is used for additional input. The haptic/graphic user interface is used for interaction with the main parts of the system, i.e., the 3D texture mapper, the image processor, the volume editor, the surface renderer, and the haptic renderer. All of these share access to a volume image $\mathbf{V}$ that we want to extract objects from. This image is typically obtained through MRI or CT. In the 3D texture mapper, we visualize the data in $\mathbf{V}$ by utilizing the hardware accelerated 3D texture mapping features of the graphics card. The image processor contains a set of different segmentation algorithms that has $\mathbf{V}$ as input and produce segmented volumes $\mathbf{S}$ as output. A segmented volume $\mathbf{S}$ is integer valued, and can contain several objects labeled between 1 and N, where N is the number of objects. Object no. j consists of the voxels $\Omega_j = \{\mathbf{x}|\mathbf{S}(\mathbf{x}) = j\}$. The background is labeled 0. In the surface renderer, fast surface reconstruction of the segmented objects in $\mathbf{S}$ is performed. The haptic renderer computes forces based on the data in $\mathbf{S}$, and is closely connected to the volume editor which contains various editing tools. As an option, the haptic feedback can also be based on $\mathbf{V}$ for enhanced navigation. Figure 3 illustrates the structure of the system.

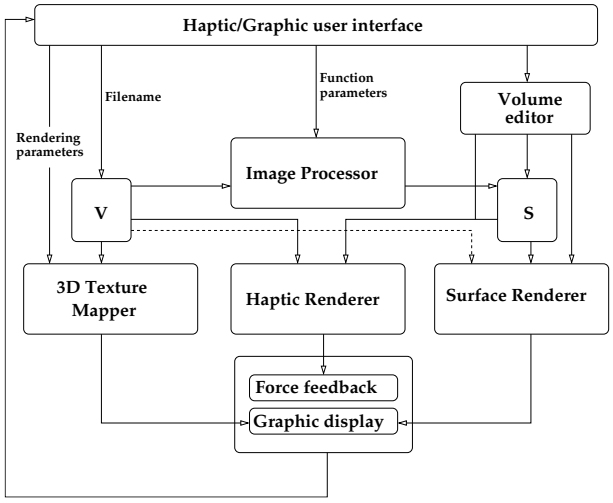


Figure 3: An overview of the interactive segmentation system.

3 3D texture mapping

Two different volume visualizations are used in our system. The surface rendering algorithm described in Section 4 is used to display segmentation results, while the original (medical) volume images are visualized through 3D texture mapping. The basic idea is to regard the whole volume image $\mathbf{V}$ as a texture map defined over $[0,1]^3$, and the texture mapping as the interpolation of the values in this domain. The default visualization in our application is a multiplanar reconstruction (MPR) consisting of three texture mapped or-

thogonal slice planes that can be moved along the corresponding coordinate axis. It is also possible to view maximum intensity projections (MIPs) of the data. We construct the MIPs by mapping the 3D texture onto a stack of view-plane aligned polygons that are rendered in back-to-front order and blended together.

To adjust contrast and brightness in these different projections, we use texture shading and the register combiner features of the graphics card. Two textures are loaded into texture memory: the volume **V** and a 1D texture **CB** that we use to store the contrast/brightness transfer function. By re-programming the register combiners, we can use the texture value from **V** as a texture coordinate for **CB**, and use that value in the rasterization. The same technique can also be used in ordinary volume rendering to implement transfer functions for opacity and color. Since the texture shading and register combinations are performed before the blending operations, any contrast/brightness adjustment affects both the slice planes and the MIP.

4 Surface rendering

A common way to use surface rendering of volume images is iso-surface extraction, i.e., a surface along which the volume image is equal to a certain threshold value, or iso-value. Interpolation is often used to achieve a smoother surface, and also shading where the surface normals are based on the volume gradient. Iso-surface extraction algorithms can be based on ray-casting methods or polygonization like in [Wyvill et al. 1986] and the more well-known marching cubes (MC) algorithm [Lorensen and Cline 1987]. We have chosen the MC algorithm since it is straight-forward and fits well into the already existing visualization environment.

In our application, we want to render the segmented and labeled objects contained in **S**. This is done by using the label of each object as iso-value.

4.1 Surface detection

The first step in the MC algorithm is to identify the cells in the volume that are intersected by the iso-surface. A cell is a cube consisting of eight $(2 \times 2 \times 2)$ neighboring voxels.

In the original implementation, the whole volume is traversed and all cells are examined for surface intersection. This is very inefficient if the surface only intersects a small part of the cells in the volume, which usually is the case.

One way to speed up the surface extraction is to use alternative data representations of the volume instead of an ordinary 3D array, e.g., an octree [Wilhelms and van Gelder 1992; Bærentzen 1998]. Drawbacks of using octrees are that the tree needs to be re-generated when the image is manipulated, and it is not straight-forward to make use of shared vertices and normals during the triangle generation.

To facilitate image manipulation and sharing of vertices and normals, we decided to use an ordinary 3D array representation as in [Galyean and Hughes 1991]. To avoid the traversal of non-intersecting cells, surface tracking [Shekhar et al. 1996] is used. This method takes advantage of surface connectivity. Given a seed-cell, i.e., a cell in the volume which is intersected by the surface, the surface is visited one cell at a time by following the connectivity until all connected cells have been visited. The connectivity for a certain MC configuration can be pre-computed and stored in a lookup-table (LUT) for efficient tracking, see Section 4.3.2.

4.2 Triangle generation

Once the surface is detected, each cell intersected by the surface should be triangulated according to the MC configurations. There are several options to consider when creating the triangles. Each vertex position of a triangle can be interpolated to give a more accurate position of the surface, or it can simply be set to a midway position on each cell edge. When dealing with binary data as in our case, no interpolation is necessary since it will default to the midway position. Regarding the normals, they can be calculated by either using the geometric normal of the triangle, or by using the gradient in the volume image. If the gradient is used in conjunction with interpolation of vertices, the gradient needs to be interpolated too. Computation of gradients and interpolation of positions are time-consuming and should be avoided unless needed for accurate visualization purposes.

One of the major drawbacks with MC is the excessive output of triangles. Since each cell intersected by the surface can result in up to five triangles, even a small volume image can result in surfaces of a massive number of polygons. Different algorithms for triangle decimation have been proposed [Montani et al. 1994; Shekhar et al. 1996],

4.3 Implementational aspects

The following was taken into consideration when implementing the MC-based surface renderer:

- The volume **S** should be easy to access and manipulate for the surface renderer, the haptic renderer, and the image processor.
- The renderer should be optimized for extraction and rendering of segmented data, but interpolation of vertices and gradient based normal computations should be included as an option.
- When the volume is manipulated, re-rendering of the surface must be efficient.

More details can be found in [Agmund 2004].

4.3.1 Data structures

The following data structures are used by the surface renderer:

- The original volume **V** of size $W \times H \times D$.
- The segmented volume **S** of the same size as V.
- A cell index array **C** of size $(W-1) \times (H-1) \times (D-1)$ containing the MC configuration index for each cell. Cells that are intersected by the surface has an index between 1 and 254 and the non-intersected cells have index 0 or 255.
- Two 2D coverage arrays $\mathbf{X}^-$ and $\mathbf{X}^+$ of size $(H-1) \times (D-1)$ containing minimum and maximum x-coordinates for surface intersected cells in each line in **C**. A value of zero in $\mathbf{X}^+$ means that the surface does not intersect any cell on the current line.
- A vertex list $\mathbf{v}_l$ for storing vertex positions.
- A normal list $\mathbf{n}_l$ for storing vertex normals.
- An index list $\mathbf{i}_l$ for triangle generation from $\mathbf{v}_l$ and $\mathbf{n}_l$.
- Three index cache arrays $\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3$ used for caching already computed indices during the triangle generation.

The main steps in the implementation of our surface renderer is shown in Figure 4.

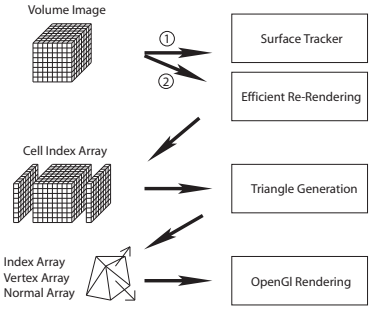


Figure 4: Overview of the surface renderer.

4.3.2 Surface tracking

The surface tracking can be started immediately if an intersected seed-cell is known and the surface to be extracted is connected. In cases where this is not true, the whole volume is scanned to find all existing surfaces. The basic algorithm is as follows: First, **C** is cleared and set to zero. $\mathbf{X}^-$ is set to W and $\mathbf{X}^+$ is set to 0. A linear search through **S** is performed until a given iso-value is found (a simple equality test). If an index is found and the cell is not previously visited (stored in **C**), surface tracking is started at the seed-cell. This procedure is repeated until the whole volume is traversed.

The surface tracking uses the values in **C** to keep track of already visited cells, and the pre-calculated connectivity LUT to find in which directions the surface is connected. See Figure 5. To be able

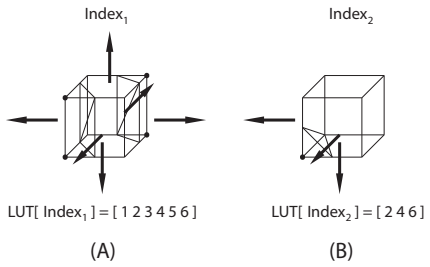


Figure 5: Example of information in the connectivity LUT for two MC configurations with connectivity in 6 directions (A) and in 3 directions (B).

to know in which order and which cells to visit a *deque* is used. A deque is a modified linked list, being efficient when elements are to be added and removed only to the end and beginning of the list. The algorithm is initialized by putting the seed-cell in the deque. The algorithm continues in the following way:

1. Pop the first cell in the deque.
2. Calculate the MC index of the current cell and insert the index into **C** for future use in the triangulation and to mark the cell as visited.
3. Compare the current x-position with the values in $\mathbf{X}^-$ and $\mathbf{X}^+$ and update if necessary.
4. Use the connectivity LUT to determine which directions the surface continues in and put these cells at the end of the deque, if they have not already been visited.
5. Repeat from step 1 until the deque is empty.

4.3.3 Vertex and normal computations

When all surfaces are found, the triangulation is performed. This is a separate process using the information stored in **C** and the coverage arrays $\mathbf{X}^-$ and $\mathbf{X}^+$. The original volume **V** is not used here, unless if interpolation is performed or if gradient-based normals are used.

The triangle generation is performed through an *xyz*-order traversal of **C**. During this process, the coverage arrays are consulted to skip the first and last non-intersected cells on each line, respectively. A LUT stores which edges for each MC configuration that will contribute with a triangle vertex. Due to the scan direction, there are only 3 of the 12 cell edges that can contribute with a new vertex. For vertex and normal sharing between triangles, the cache arrays store indices of already computed vertices and normals. This is illustrated in Figure 6. Vertex normals can be calculated in two

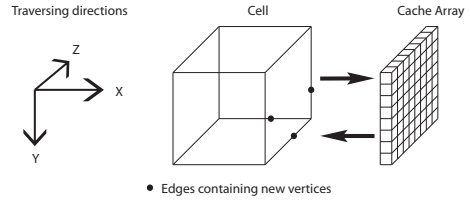


Figure 6: Illustration of the three edges that contribute with new vertices. Information about vertices on the other nine edges are already known and can be retrieved from the cache arrays.

different ways. The first, which is the most efficient, uses the average geometrical normal from each triangle that shares the vertex. The efficiency lies in that the geometrical normals for each triangle are pre-calculated and stored in a LUT for each MC configuration. This is possible since we work with binary data and only use the midway position on each edge. The second, more costly, method to calculate normals is to use the gradients from **V** at each vertex position.

4.3.4 Efficient re-rendering

In the application, there are two ways of manipulating the segmented volume **S**. The first is to apply a global method (e.g., thresholding of **V**) that requires a total update according to the surface tracking algorithm. The second is to apply local editing operations from the volume editor. Since these operations only affects a small part of the image it is enough to traverse a sub-volume which extent depends on the current editing tool, and to update the corresponding values in **C**, $\mathbf{X}^+$, and $\mathbf{X}^-$. In the current implementation, we must re-generate all triangles, but since all computations are reduced to table lookups this is not a problem. However, in the future we will try to improve the implementation to modify only parts of $\mathbf{v}_l$, $\mathbf{n}_l$, and $\mathbf{i}_l$.

5 Interactive editing with haptic feedback

One of the first attempts to use haptics for the display of volume data was made in [Avila and Sobierajski 1996]. In their work, the force feedback provided to the user is a direct mapping from the position of the haptic probe to a force computed from intensity values and local gradients at that position. A drawback with this type of method is instability. The rendering parameters can be hard to tune correctly in order to avoid force oscillations. In surface haptics, the stability problem was solved by introducing a virtual *proxy* that is connected to the haptic probe through a spring-damper [Ruspini et al. 1997].

5.1 Proxy-based volume haptics

The idea in proxy-based haptic rendering is to constrain the proxy to certain movements in a local reference frame (LRF) and to provide a resulting force vector proportional to the displacement of the haptic probe relative to the proxy. Proxy-based rendering of volumetric data was first proposed by [Lundin et al. 2002], where a LRF for scalar volumes is obtained through tri-linear interpolation of the volume gradient at the proxy position. The gradient is used as a surface normal that defines a surface to which the proxy is constrained. It is also shown how friction and viscosity can be rendered and how different material properties can be simulated by using haptic transfer functions. In [Ikits et al. 2003], a framework for more general LRFs and proxy *motion rules* was presented.

5.2 Haptic feedback when editing

We have based our haptic rendering on the two works mentioned in Section 5.1 combined with the idea of a tool with sample points on the surface [Petersik et al. 2003].

The basic steps in the haptic loop are as follows: let $\{\boldsymbol{e}_0, \boldsymbol{e}_1, \boldsymbol{e}_2\}$ denote the LRF, $\boldsymbol{p}^q$ the proxy position at time step q, $\boldsymbol{x}^q$ the probe position, and $\boldsymbol{d} = (\boldsymbol{x}^q - \boldsymbol{p}^{q-1})$ the displacement of the probe relative to the previous proxy position. In each iteration of the haptic loop the proxy is moved in small steps according to user input and rendering parameters such as stiffness and friction. Allowed proxy movements are defined by certain motion rules for each axis in the LRF. The proxy position at time step q is computed as

$$\boldsymbol{p}^q = \boldsymbol{p}^{q-1} + \sum_{i=0}^{2} \Delta p_i \boldsymbol{e}_i,$$

where Δp_i is a motion rule function of the displacement $d_i = \boldsymbol{d} \cdot \boldsymbol{e}_i$. The resulting force is computed as $\boldsymbol{f}^q = -k(\boldsymbol{x}^q - \boldsymbol{p}^q)$, where k is a stiffness constant.

We use a spherical tool with radius r that is centered at $\boldsymbol{p}$. In a precomputed array we store uniformly spaced sample points $\boldsymbol{t}_i$, $||\boldsymbol{t}_i|| = 1$, so that the points $\boldsymbol{T}_i = \boldsymbol{p} + r \cdot \boldsymbol{t}_i$ are located on the tool surface. The sample points that are in contact with the current object are used to define the normal component $\boldsymbol{e}_0$ in our LRF:

$$\boldsymbol{e}_0 = -\frac{\sum_{i \in I} \boldsymbol{t}_i}{||\sum_{i \in I} \boldsymbol{t}_i||},$$

where $I = \{i | \mathbf{S}(\boldsymbol{T}_i) > 0\}$. The tangential direction $\boldsymbol{e}_1$ is constructed by projecting $\boldsymbol{d}$ onto the plane defined by $\boldsymbol{e}_0$ [Lundin et al. 2002]:

$$\boldsymbol{e}_1 = \frac{\boldsymbol{d} - (\boldsymbol{d} \cdot \boldsymbol{e}_0)\boldsymbol{e}_0}{||\boldsymbol{d} - (\boldsymbol{d} \cdot \boldsymbol{e}_0)\boldsymbol{e}_0||}.$$

Since $\boldsymbol{e}_1$ is constructed in this way, the third component of the LRF is not needed, but it can easily be computed as $\boldsymbol{e}_2 = \boldsymbol{e}_0 \times \boldsymbol{e}_1$.

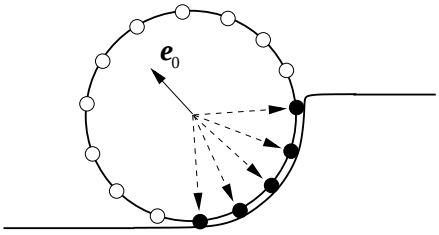
The motion rule for the normal direction $\boldsymbol{e}_0$ is

$$\Delta p_0 = \begin{cases} d_0 & \text{if} \quad d_0 > 0 \\ -\max(|d_0| - T_0/k, 0) & \text{if} \quad d_0 \leq 0 \end{cases},$$

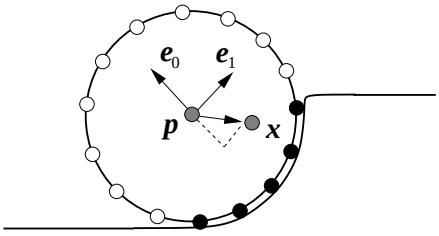
where the threshold T_0 is the force the user must apply to penetrate the surface with the tool. For the tangential direction $\boldsymbol{e}_1$, the motion rule is

$$\Delta p_1 = \max(d_1 - T_1/k, 0),$$

where $T_1 = \mu k |d_0|$, i.e., a friction force threshold with friction coefficient μ. This motion rule is used to avoid slippery surfaces. The parameters k, μ, and T_0 can be controlled through the user interface. Figure 7 illustrates the idea behind the haptic rendering.



(a) Computation of $\boldsymbol{e}_0$ by finding the points on the tool surface that are in contact with the object.



(b) $\boldsymbol{e}_1$ is constructed by projecting $\boldsymbol{d} = (\boldsymbol{x} - \boldsymbol{p})$ onto the plane defined by $\boldsymbol{e}_0$.

Figure 7: Idea behind the haptic rendering.

5.3 Editing operations

Editing of the volume **S** is performed with the spherical tool described in Section 5.2. The tool can be either active or inactive. When the tool is active, all object voxels in **S** located within the tool boundaries will be affected by the currently selected editing operation. So far, we have implemented four basic editing operations: draw, erase, erode, and dilate. Erosion and dilation are binary morphology operators [Gonzalez and Woods 2002, Chapter 9] that are used to peel off a voxel layer and to add a voxel layer, respectively. See Figure 8 for a simple erosion example. The editing operations that are provided with haptic feedback is erase, erode, and dilate. Haptic feedback for drawing can be turned on as an option and is based on the gradient of **V** for feeling object boundaries.

Drawing and erasing are simple operations that can work directly on **S**, while erosion and dilation need an input volume and an output

Figure 8: The smiley is constructed by erosion.

volume. Therefore, a temporary volume $\mathbf{S}'$ is used. $\mathbf{S}'$ is a copy of $\mathbf{S}$ that is not modified while the tool is active. When the tool is deactivated, $\mathbf{S}'$ is updated according to the current $\mathbf{S}$.

6 Application example

In a recently started project in co-operation with the Dept. of Radiology at Uppsala University Hospital, we develop interactive segmentation methods as a part of liver surgery planning. As an initial part of the project, we have developed a method for segmentation of the liver from MR images. The images are of size $256 \times 256 \times 100$ voxels.

First, we apply pre-processing filters to the original data set, i.e., edge-preserving smoothing followed by gradient magnitude extraction. The gradient magnitude is used to construct a speed function for input to a fast marching segmentation algorithm [Sethian 1999]. As the next step, we use our drawing tool to create an initial seed-region inside the liver. The fast marching algorithm then propagates this region towards the liver boundaries. The propagation is fast where the gradient magnitude is low and vice versa. When the algorithm has converged, we examine the result and, if necessary, perform manual editing.

A screenshot from the application is shown in Figure 1. The initial segmentation result contains several artifacts due to "leaking", i.e., the contrast between the liver and the surrounding tissues is low. After manual editing, most of the artifacts are removed.

7 Results

To test the surface renderer, we generated a test image by sampling a 3D Gaussian function on a $128 \times 128 \times 128$ grid. We loaded the image into our enviroment and thresholded it at different levels to produce triangle meshes consisting of 20,000–100,000 triangles. We used the erosion operation to edit the mesh with tool radii $r = 5$ and $r = 10$ voxels. The resulting average update rates are given in Table 1, where it can be seen that the time for triangle generation increases linearly with the number of triangles. As a consequence, the effect of different tool radii decreases as the objects become larger. In the application example (Section 6), the marching cubes surface of the segmented liver consisted of 95000 triangles and was edited at frame rates between 10 and 12 frames per second. Tool radii between 1 and 10 voxels was used, and the number of tool sample points was 340. The haptic update rate was kept constant at 1 kHz which is the rule of thumb for perceptually convincing haptic feedback.

Table 1: Average update rates when editing triangle meshes with an erosion tool having a radius of r voxels.

	Update rate (frames/s)	
#Triangles	$r = 5$	$r = 10$
20,000	52	40
30,000	39	25
40,000	30	22
50,000	25	19
60,000	21	17
70,000	20	18
80,000	17	15
90,000	11	10
100,000	9	8

8 Conclusions and future work

The surface renderer that we have developed can be used for interactive editing of segmented objects. The efficiency lies mainly in the surface tracking and the index caching strategies. However, we note that when complex objects are triangulated, the huge number of triangles considerably slows down the rendering. To overcome this problem we will investigate how a triangle decimation algorithm could be incorporated and how the re-rendering can be improved to update only parts of the triangle mesh.

Regarding the haptic editing tools we are encouraged by these initial results, so we will extend the volume editor with several editing operations, arbitrarily shaped editing tools, and more sophisticated haptic rendering. Further more, we will investigate how to facilitate the creation of seed-regions by using haptic feedback based on the original volume $\mathbf{V}$. Ideally, the haptic feedback would force the user to draw inside the object, but using only gradient information for this purpose is not enough since it is easy to lose track of object boundaries when the contrast is low.

The segmentation method has shown promising results and we will continue development of the method. Evaluation of the segmentation method and the usefulness of the haptic editing tools will be conducted in coming work.

Acknowledgments

We would like to thank Doc. Ingela Nyström and Prof. Ewert Bengtsson at the Centre for Image Analysis for proofreading and useful comments. Prof. Håkan Ahlström and Dr Hans Frimmel at the Dept. of Radiology at Uppsala University Hospital are acknowledged for providing the MRI data. This work was funded by the Swedish Research Council, no. 2002-5645.

References

AGMUND, J. 2004. *Real-time surface rendering for interactive volume image segmentation in a haptic environment*. Master's

thesis, Uppsala University, Centre for Image Analysis. UPTEC F04 071.

AVILA, R. S., AND SOBIERAJSKI, L. M. 1996. A haptic interaction method for volume visualization. In *Proceedings of IEEE Visualization'96*, 197–204.

BÆRENTZEN, J. A. 1998. Octree-based volume sculpting. In *Proceedings of Late Breaking Hot Topics. IEEE Visualization'98*, 9–12.

GALYEAN, T. A., AND HUGHES, J. F. 1991. Sculpting: An interactive volumetric modeling technique. In *Proceedings of ACM SIGGRAPH'91*, 267–274.

GONZALEZ, R. C., AND WOODS, R. E. 2002. *Digital image processing*, second ed. Prentice Hall, Inc.

HARDERS, M., AND SZÉKELY, G. 2002. Improving medical segmentation with haptic interaction. In *Proceedings of IEEE Virtual Reality (VR'02)*, IEEE CS Press, IEEE Computer Society, 243–250.

IKITS, M., BREDERSON, J. D., HANSEN, C. D., AND JOHNSON, C. R. 2003. A constraint-based technique for haptic volume exploration. In *Proceedings of IEEE Visualization'03*, 263–269.

KANG, Y., ENGELKE, K., AND KALENDER, W. A. 2004. Interactive 3D editing tools for image segmentation. *Medical Image Analysis 8*, 1, 35–46.

KIM, L., SUKHATME, G. S., AND DESBRUN, M. 2004. A haptic-rendering technique based on hybrid surface representation. *IEEE Computer Graphics and Applications 24*, 2, 66–75.

LORENSEN, W. E., AND CLINE, H. E. 1987. Marching cubes: A high resolution 3D surface reconstruction algorithm. *Computer Graphics 21*, 4 (July), 163–169.

LUNDIN, K., YNNERMAN, A., AND GUDMUNDSSON, B. 2002. Proxy-based haptic feedback from volumetric density data. In *Proceedings of Eurohaptics 2002*, 104–109.

MONTANI, C., SCATENI, R., AND SCOPIGNO, R. 1994. Discretized marching cubes. In *Proceedings of IEEE Visualization'94*, IEEE Computer Society Press, Washington D.C., USA, R. D. Bergeron and A. E. Kaufman, Eds., IEEE Computer Society, 281–287.

PETERSIK, A., PFLESSER, B., TIEDE, U., HOEHNE, K. H., AND LEUWER, R. 2003. Realistic haptic interaction in volume sculpting for surgery simulation. In *Proceedings of IS4TM'03*, Springer-Verlag Berlin Heidelberg, N. Ayache and H. Delingette, Eds., no. 2673 in LNCS, 194–202.

RUSPINI, D. C., KOLAROV, K., AND KHATIB, O. 1997. The haptic display of complex graphical environments. In *Proceedings of ACM SIGGRAPH'97*, ACM SIGGRAPH, 345–352.

SETHIAN, J. A. 1999. *Level set methods and fast marching methods*. Cambridge University Press.

SHEKHAR, R., FAYYAD, E., YAGEL, R., AND CORNHILL, J. F. 1996. Octree-based decimation of marching cubes surfaces. In *Proceedings of IEEE Visualization'96*, 335–ff.

THURFJELL, L., MCLAUGHLIN, J., MATTSSON, J., AND LAMMERTSE, P. 2002. Haptic interaction with virtual objects: The technology and some applications. *Industrial Robot 29*, 3, 210–215.

VIDHOLM, E., TIZON, X., NYSTRÖM, I., AND BENGTSSON, E. 2004. Haptic guided seeding of MRA images for semi-automatic segmentation. In *Proceedings of IEEE International Symposium on Biomedical Imaging (ISBI'04)*, 278–281.

WILHELMS, J., AND VAN GELDER, A. 1992. Octrees for faster isosurface generation. *ACM Transactions on Graphics 11*, 3, 201–227.

WYVILL, B., MCPHEETERS, C., AND WYVILL, G. 1986. Data structure for soft objects. *The Visual Computer 2*, 4, 227–234.