2007-02-24

Live Surface

Christopher J. Armstrong
Brigham Young University - Provo

LIVE SURFACE

by

Christopher J. Armstrong

A thesis submitted to the faculty of

Brigham Young University

in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computer Science

Brigham Young University

April 2007

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a thesis submitted by

Christopher J. Armstrong

This thesis has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory.

Date	William A. Barrett, Chair
Date	Bryan S. Morse
Date	Dennis Ng

BRIGHAM YOUNG UNIVERSITY

As chair of the candidate's graduate committee, I have read the thesis of Christopher J. Armstrong in its final form and have found that (1) its format, citations, and bibliographical style are consistent and acceptable and fulfill university and department style requirements; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the graduate committee and is ready for submission to the university library.

Date	William A. Barrett Chair, Graduate Committee

Accepted for the Department

Parris Egbert
Graduate Coordinator

Accepted for the College

Earl M. Woolley
Dean, College of Physical and Mathematical Sciences

ABSTRACT

LIVE SURFACE

Christopher J. Armstrong

Department of Computer Science

Master of Science

Live Surface allows users to segment and render complex surfaces from 3D image volumes at interactive (sub-second) rates using a novel, Cascading Graph Cut (CGC). Live Surface consists of two phases. (1) Preprocessing for generation of a complete 3D watershed hierarchy followed by tracking of all catchment basin surfaces. (2) User interaction in which, with each mouse movement, the 3D object is selected and rendered in real time. Real-time segmentation is accomplished by cascading through the 3D watershed hierarchy from the top, applying graph cut successively at each level only to catchment basins bordering the segmented surface from the previous level. CGC allows the entire image volume to be segmented an order of magnitude faster than existing techniques that make use of graph cut. OpenGL rendering provides for display and update of the segmented surface at interactive rates. The user selects objects by tagging voxels with either (object) foreground or background seeds. Seeds can be placed on image cross-sections or directly on the 3D rendered surface. Interaction with the rendered surface improves the user's ability to steer the segmentation, augmenting or subtracting from the current selection.

Segmentation and rendering, combined, is accomplished in about 0.5 seconds, allowing 3D surfaces to be displayed and updated dynamically as each additional seed is deposited. The immediate feedback of Live Surface allows for the segmentation of 3D image volumes with an interaction paradigm similar to the Live Wire (Intelligent Scissors) tool used in 2D images.

ACKNOWLEDGMENTS

By authoring this thesis I by no means claim full responsibility for the innovation presented herein. The individuals and organizations listed here were invaluable in inspiring, exploring, discovering, motivating, and executing the development of Live Surface.

As a masters thesis presented to the faculty of Brigham Young University, many individuals affiliated with BYU should be acknowledged. Those most directly related to this work that I would like to thank are the following:

Dr. William Barrett for providing me with the opportunity to be involved with such exciting research; for providing insight, direction, motivation, and encouragement; and for ensuring that I felt like I could do anything.

Brian Price for laying some of the graph-cut ground work; for collaborating, sharing code, co-authoring, and treading the path of an MS student making it easier for me to see the way.

My graduate committee for taking the time to review my work and provide helpful suggestions.

The Computer Science Department including the department administration, secretaries, system administrators, and all others that I have taken for granted in their indispensable supporting roles.

The Center for Instructional Design for their employment of a poor graduate student.

Dr. David Belnap for providing the polio dataset and the expertise to understand it; and for providing support to my family as a dedicated home teacher.

Contributions have come from other organizations not directly affiliated with the university that have also had a hand in furthering this research. Those include:

Adobe Systems Inc. for funding the research and feeding my family.

Chief Architect Inc. for their support of my completion of this thesis.

Volume Graphics, **Computers and Graphics**, **SIGGRAPH**, and **Various news organizations** for publishing this research, helping me learn to write, and/or making this research available to the world.

More importantly still I acknowledge the support, patience, and love provided by the following individuals:

My grandmother, the late Virginia Armstrong, for taking us in and providing company for my wife on those late nights in the lab.

My parents for providing the stepping stones necessary to become a successful student.

My wife Jill for her patience, understanding, encouragement, support and unconditional love, without which this would not have been possible.

The final individuals that I would like to thank are:

The Holy Spirit whose still small voice whispers truth of all kinds.

My Savior, Jesus Christ who suffered through all of the anxiety, inadequacy, disappointment, and frustration of this work as well as sharing in the excitement, joy, and pleasure it has brought.

My father in Heaven who provided me with the talents necessary to pursue this research and see it through.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Intelligent Scissors for Volumes

The Live Wire interface of Intelligent Scissors [1] is an effective means for performing 2D segmentation by providing immediate feedback for boundary selection as the mouse moves. As shown in Figure 1.1, The user simply selects a pixel as a seed pixel by pressing the mouse button. As the user drags the mouse with the button pressed the optimal boundary containing the seed pixel, and the new mouse cursor position is selected. The selected boundary is simply the least-cost path from the cursor to the seed pixel. The least-cost path formulation makes for an "optimal" boundary. As the user continues to drag the mouse a new boundary is selected and displayed with each mouse movement. The boundary is called a "live wire" because as the user moves the mouse the boundary is continuously changing and snapping to the natural edges in the image. This gives the user constant awareness of what belongs to the current selection.

Unfortunately there is no straightforward extension of Intelligent Scissors to 3D image volumes. The crux of the problem is that a Live Wire is a one-dimensional boundary, and a segmentation in 3D requires a 2D "Live Surface". While there are many algorithms for finding a least cost path, none of them extend directly to the concept of a least cost surface. A series of least cost paths can be generated forming a mesh as in [2], but this does not provide a per voxel (3D pixel) classification of foreground and background.

For 3D volume segmentation, the analogous feedback model to the 2D Live

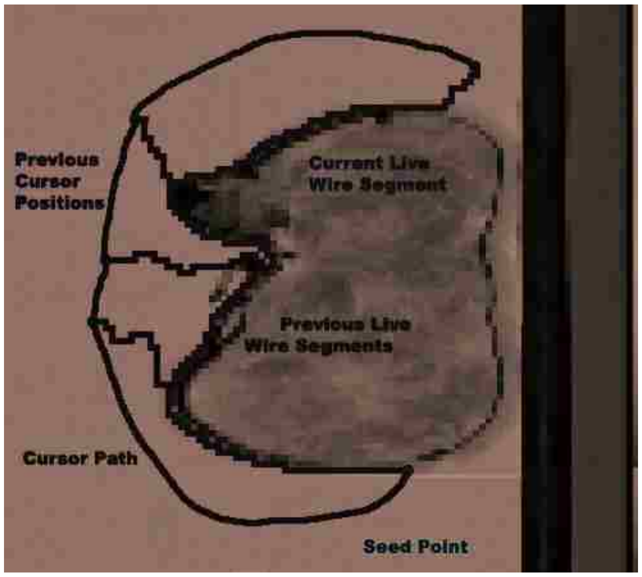

Previous Cursor Positions
Current Live Wire Segment
Previous Live Wire Segments
Cursor Path
Seed Point

Figure 1.1: The Intelligent Scissors/Live Wire interface. Here Live Wire is applied to a cross section of a CT scan. The seed pixel is show in red. Four cursor positions are shown in green. The optimal boundary containing the final cursor position and the seed pixel is shown in yellow. The path followed by the cursor is shown in pink. The boundaries based on the previous cursor positions are shown in light green.

Wire interface would be a "Live Surface." This close coupling of user interaction and optimal segmentation would also be useful for segmenting 3D image volumes. The ability to update surfaces incrementally and interactively, within an optimal framework, would provide a user with the same freedom of interaction in extracting surfaces from image volumes as Intelligent Scissors provides in the extraction of boundaries from 2D images. 3D image volumes are encountered frequently in the medical profession in the form of CT scans, MRI, etc. Other forms of tomography and microscopy also generate image volumes for various purposes. Even video can be considered an image volume. The ability to interactively detect surfaces of objects in image volumes would facilitate the analysis and exploitation of the volume.

Intelligent Scissors was not the first technique that used the idea of a least cost path for finding optimal boundaries. However, its key contribution was to bring optimal boundary detection into an interactive framework in order to allow for better cooperation between human and machine in the critical process of 2D image segmentation. Thus, the introduction of Intelligent Scissors fundamentally changed the paradigm for interactive boundary detection.

We present Live Surface, a new system for interactively selecting optimal surfaces in 3D image volumes. Live Surface gives users the ability to update surfaces incrementally and interactively, within an optimal framework, providing a user with the same freedom of interaction in extracting surfaces from image volumes as Intelligent Scissors provides in the extraction of object boundaries in 2D images. Live Surface does this by leveraging the existing optimal graph-cut algorithm rather than Dykstra's algorithm used by Intelligent Scissors. Live Surface accomplishes this with a simple, intuitive user interface consisting of only mouse clicks and strokes not only on 2D cross-sections, but also on 3D rendered surfaces. Interaction with 3D surfaces provides the user with the ability to inspect the results and provide further input without switching contexts or searching through cross sections.

1.2 Motivation

CT and MRI are two common means for noninvasive diagnosis of many medical conditions. In each case, a 3D image volume is generated and analyzed. The accurate analysis of medical image volumes can allow medical professionals to diagnose illness and save lives. Along with diagnosis, interactive optimal segmentation of CT and MRI can be used for measurement, planning surgeries, and prosthesis fitting.

A common task in the analysis of medical image volumes is the measurement of anatomy. Measurements of diameter and circumference are typically taken from a 2D interaction with a single slice of a volume. For example, the measurement of the circumference of an unborn child's head is taken by drawing an ellipse on a 2D cross section acquired through ultrasound. The ultrasound technician attempts to maneuver the sensor to a position that will acquire a cross section containing the widest point on the baby's head. Finding that cross section is not always straightforward.

Using 3D ultrasound, an image volume can be acquired. An accurate segmentation and visualization of the volume greatly simplifies the process of obtaining a measurement of the widest circumference of the baby's head as well as visualizing the overall anatomy.

Another medical task in which 3D images volumes are quite useful is in surgical planning. Before performing a surgery it would be helpful for the surgeon to explore the particulars of the patient's anatomy. An interactive tool for selecting the surfaces of objects within the volume would facilitate a contextual exploration and allow for better preparation for surgery. Along with the performance of the surgery itself, prosthesis, or artificial replacement anatomy could be custom made based on the geometry obtained through an interactive segmentation.

3D image volume are made up of a series of parallel 2D images distributed along a z axis. Video is a series of 2D images distributed along a third access. In this sense video can be treated as an image volume as well. In this case the third axis is the temporal (t) dimension. Interactive segmentation of video would greatly facilitate many visual effect processes used in multimedia production. Furthermore the exploration of a video surface would prevent the need to search through each frame to check for discrepancies in the segmentation since they will be visible in one place on the video surface.

Image volumes are acquired in many other applications as well. Among these are geological, biochemical, industrial, and meteorological image volumes. In every case segmentation, visualization, and measurement come into play. An interactive tool for segmenting image volumes could have wide application in each of these areas.

1.3 Challenges

The main difficulty in providing a system that interactively extracts surfaces from 3D image volumes is in combining the principle of optimality with interactive speeds. Optimal here means that, based on the user input and some function measuring the cost of the segmentation, the result is the best segmentation in the sense of minimizing the cost. Finding the best segmentation based on the user input requires searching among the possible segmentations. Due to the large number of voxels

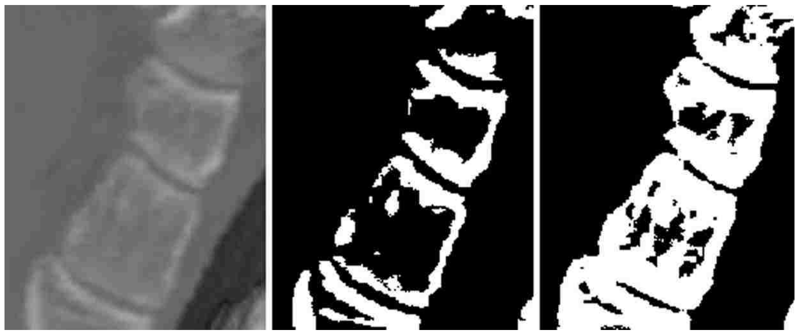


Figure 1.2: Drawbacks to thresholding. (a) Foot bones from a CT volume. Thresholding techniques often either (b) underestimate the surface, eroding away the bones or (c) overestimate the surface, merging the bones.

in each volume and the enormous number of possible segmentations, previous optimal segmentation techniques are simply unable to segment the volume at interactive speeds.

3D image volume segmentation techniques that are able to run at interactive rates do so because they have a much smaller search space from which to choose a segmentation. In other words they are not optimal. Such techniques as simple thresholding have a tendency to overestimate or underestimate the desired surface, as illustrated by the example in Figure 1.2. For objects whose surfaces vary in intensity or color it can be impossible to obtain an accurate segmentation using thresholding. Furthermore, when the choice of the threshold or other parameters is left to the user, the user must have a deeper contextual knowledge of the domain from which the image was obtained, and then still may have to make assumptions. A second problem with these simplistic techniques is the inability to distinguish between neighboring objects with similar properties, fusing two or more into a single object. This is evident in Figure 1.2c. The surfaces between such neighboring objects are then impossible to explore. For these reasons it is necessary to find a means for combining the interactive speeds with optimal 3D object segmentation.

1.4 Overview of Live Surface

Live Surface is a tool that combines interactive speeds with optimal segmentation of 3D image volumes by using Cascading Graph Cut on a complete watershed hierarchy, the details of which are discussed in Chapter 3. Cascading Graph Cut allows the user to segment the volume as the mouse moves. Updates are computed in less than a second. Just as Intelligent Scissors brought optimal boundary selection in 2D images into an interactive framework. Live Surface brings optimal graph cut for 3D image segmentation into an interactive framework with constant feedback, allowing better cooperation between the user and the machine.

The remainder of this thesis is organized as follows: Chapter 2 discusses the relevant previous work. It begins with a discussion of systems for working with volume graphics, and segmentation techniques. It also contains a discussion of graph cut, a key algorithm in the development of Live Surface. It moves from general segmentation techniques to recently developed systems for segmenting images, volumes, and video. It also details two key algorithms upon which Live Surface is built. Chapter 3 details the components that make up Live Surface, including the complete watershed hierarchy and Cascading Graph Cut. Chapter 4 shows examples of segmentations generated with Live Surface and analyzes it's performance. Chapter 5 draws some conclusions and indicates some possible future directions for Live Surface.

Chapter 2

Related Work

At the heart of Live Surface is the segmentation of 3D image volumes. For this reason this chapter begins with some general information of 3D image systems. From there some general techniques for image segmentation are discussed. Finally greater detail is presented on the main algorithm on which Live Surface is based: graph cut.

2.1 Volume Graphic Systems

Some of the systems representative of the excellent work done in extracting surfaces from 3D image volumes are discussed in this section. The Mayo Clinic's Analyze System [3] consists of a comprehensive suite of tools for segmentation, analysis and visualization of image volumes. 3D Viewnix from the University of Pennsylvania [4] contains advanced segmentation tools such as Fuzzy Connectedness [5, 6, 7]. Voxel-man [8], well known for its exquisite daVinci-like renderings, represents the state of the art in anatomical visualization. In addition, many commercial systems now provide high quality volume rendering of medical image volumes (e.g. GE [9]). However, these and other systems could greatly benefit from techniques that combine optimal, interactive segmentation with sub-second rendered feedback.

Recently volume rendering has become the tool of choice for visualizing 3D image volumes. Volume rendering has been around for quite some time [10, 11], but only recently has the hardware made it viable for quickly providing high quality results. Volume rendering generates 2D images of 3D image volumes by projecting image data along a ray through the image volume to the viewer. The data that is projected is fed through a transfer function, which allows for coloring, transparency, and translucency in order to make the objects of interest conspicuous. Transfer functions have been studied to the point that physicians using volume rendering software have a library of transfer functions to choose from in order to view many kinds of anatomy in CT volumes.

The main drawback of volume rendering and transfer functions is that they are unable to provide actual segmentation of the anatomy, resulting in only false coloring from the transfer functions. As a result 3D measurements of the visualized surface are also not possible. Segmentation is needed to measure objects from image volumes. Volume rendering is targeted at visualization and thus side-steps the difficulties associated with segmentation. Another drawback of volume rendering is the complexity of choosing a transfer function. Since predefined transfer functions are available for most human anatomy in CT volumes, volume rendering has found success with this kind of data. However, the average physician does not have the necessary background to define their own transfer functions for special purposes or for different imaging modalities. According to Ron Kikinis of Brigham and Womens Hospital, if a physician is given more than one "slider" he/she won't use the tool [12]. Ideally, what is needed is an approach that provides measurable segmentation and has a simple interface without any parameters to tune or set.

2.2 Segmentation Techniques

There are many techniques for performing image segmentation. Most of them fall into three main categories: thresholding, edge-based segmentation, and region-based segmentation. Each of these is discussed here.

Thresholding is possibly the simplest approach to image segmentation [13]. Thresholding algorithms classify image elements (pixels or voxels) as foreground or background based on whether or not they are less than or greater than a given threshold. Thresholding is very fast making it efficient for segmenting large volumes. The major concern with thresholding is the sensitivity to overestimation or underestimation of the threshold as shown in Figure 1.2. In most cases, due to variability in the surface or boundary, a global threshold is simply unsatisfactory. For this reason techniques have been developed to adaptively threshold images locally, or refine the segmentation after the threshold has been set.

Edge-based segmentation techniques [13] look for natural boundaries in the image. All edge-based segmentation techniques begin with some form of edge detection, frequently applying a gradient or Laplacian operator to the image in order to generate an edge image. Such operators detect areas of high changes in intensity (gradients) in the image. The areas of high gradient are likely the edges of objects in the image. A set of pixels exhibiting high gradient may constitute the boundary of an object in 2D. In 3D an entire surface of voxels in needed. There are various ways to find such boundaries. Some of these include edge relaxation, or graph search techniques.

Intelligent Scissors introduced an efficient means for interactive 2D segmentation through the use of a Live Wire, which adheres to natural edges within the image. Intelligent Scissors [1] brought 2D segmentation based on optimal graph search techniques into an interactive framework and is the basis for Adobe Photoshop's magnetic lasso. Intelligent Scissors provided much of the inspiration for Live Surface, however Intelligent Scissors does not readily extend to Surfaces.

Region-based segmentation [13] tends to perform better than edge-based segmentation in the presence of noise. Region-based methods, rather than looking for areas of high gradient, look for areas of homogeneity. The general idea behind region-based segmentation is usually some form of merging of pixels (voxels) and groups of pixels. Splitting larger regions into smaller regions is also often used. Sometimes a combination of merging and splitting is used.

Watersheds [14, 15, 16] are considered a form of region-based segmentation. Since watersheds form a critical portion of Live Surface they will be discussed in more detail later. Intelligent Paint [17], a region-based segmentation tool that provides interactive segmentation through a flood fill metaphor, uses a watershed hierarchy [17]. This hierarchy has been incorporated in Live Surface.

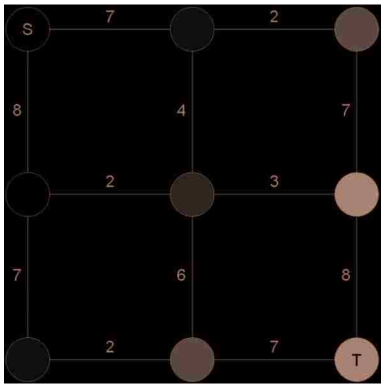
2.3 Graph Cut

A key algorithm for extraction of a Live Surface is graph cut. Live Surface uses the Boykov–Kolmogorov implementation [18] to perform the graph cut. Graph cut segments a graph into two mutually exclusive subgraphs (foreground and background) by cutting the edges connecting one subgraph to the other such that the sum of the weights of the cut edges is minimized. Seeds (placed by the user) assign nodes to belong to one of the two subgraphs. Our graph is formed as described in 3.3. In Boykov and Jolly's implementation [19] each voxel is a node in the graph, and edges indicate adjacency of voxels. Two nodes to take note of are the source and the sink (labeled S and T respectively in Figure 2.1). These terminal nodes are linked with infinite edge weights to the foreground and background seed voxels specified by the user.

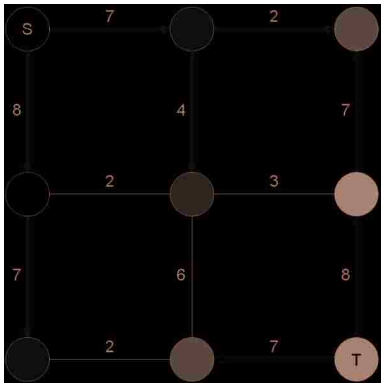
The Boykov–Kolmogorov version of graph cut begins with a bidirectional search for a path from the source to the sink as illustrated by the arrows in Figure 2.1b. Once a path is found, all edge weights on the path are decremented by the minimum edge on the path (Figure 2.1c) and the minimum edge is cut (Figure 2.1d). The algorithm continues by picking up the search for a path where it left off (Figure 2.1e), decrementing the edge weights and cutting an edge (Figure 2.1f). The process repeats until there is no longer a path from source to sink (Figure 2.1g–i). Those nodes still connected to the source are the foreground. The rest are background.

The graph-cut algorithm popularized by Boykov and Jolly [19] has found widespread use in advancing the work of optimal 2D and 3D segmentation. Interactive segmentaion of 2D images using graph cut has had great success in systems like Lazy Snapping [20]. In fact, Live Surface uses graphs built using a portion of the edge-weighting scheme described in Lazy Snapping.

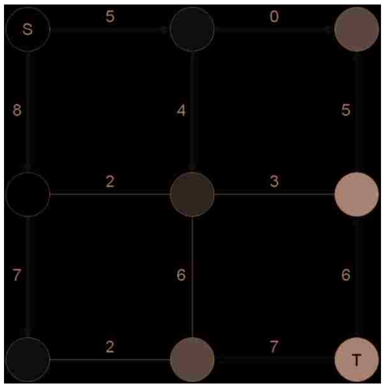
The use of graph cut for segmentation of 3D surfaces has been extensively validated for medical image volumes [21]; however, execution times can last up to tens of minutes to cut volumes of 2–8 megavoxels. In order to accelerate the process, a single layer of watershed regions [16] has been used in the place of voxels for medical volumes [22] and video [23]. Interactive Video Cutout [24] goes a step



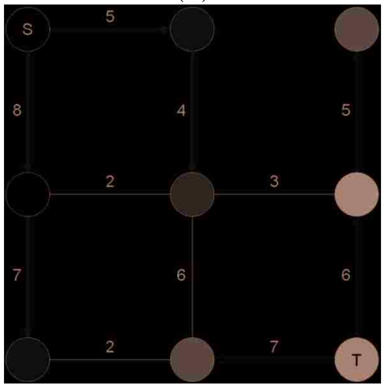
(a)



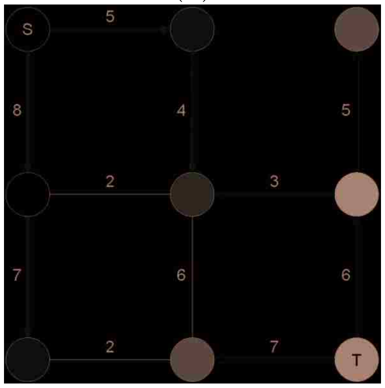
(b)



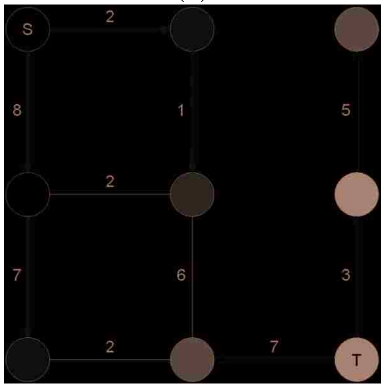
(c)



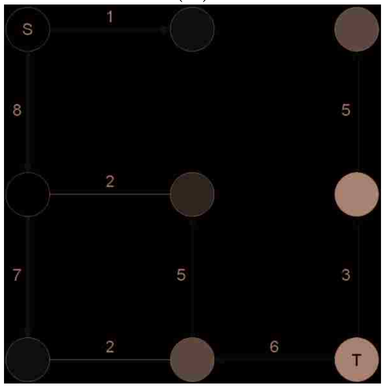
(d)



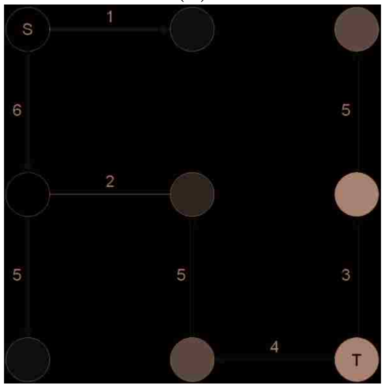
(e)



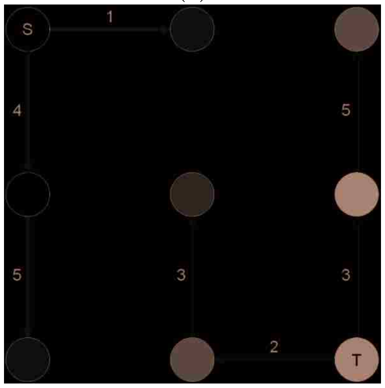
(f)



(g)



(h)



(i)

Figure 2.1: The Boykov–Kolmogorov graph-cut algorithm.

further, generating two layers of oversegmented image regions. Lombaert uses a resolution pyramid to perform coarse-to-fine refinement in Banded Graph Cuts [25]. Each of these accelerated approaches still typically requires from 5 to 10 seconds to one minute. However, even five to ten seconds is too slow to be truly interactive. The complete watershed hierarchy as described by Reese and Barrett [17] now provides a means for near real-time segmentation in Live Surface.

Chapter 3

Live Surface

This chapter provides conceptual and implementational details of Live Surface. To begin, Section 3.1 describes Live Surface from the user's perspective. Section 3.2 describes how all of the components of Live Surface are combined to work together. The next sections describe the main algorithmic components: the watershed hierarchy (Section 3.3), and Cascading Graph Cut (Section 3.4). This is followed by a description of how the resulting surface is rendered in Section 3.5 and a description of how user input is interpreted in Section 3.6.

3.1 A User Perspective

The interface to Live Surface is shown in Figure 3.1. The user is presented with two windows: The data view, which is displayed as slices or cross sections on the left, and the rendered surface of the selected 3D object.

The data view is a parallelepiped through which the user can advance a cross section parallel to the viewing plane. The parallelepiped can also be rotated so that arbitrary planar cross sections can be viewed. The user may also lock the cross section to a specific voxel. As the user rotates the volume, the cross section will advance and retreat so that the locked voxel remains in the current cross section. This allows the user to more easily find a cross section that will minimize interaction.

When the user finds a cross section on which to make a selection, the user selects *seed mode* and clicks on voxels to place foreground seeds (left mouse button) and background seeds (right mouse button) in the volume. The user should select foreground seeds that are representative of the object and are contained anywhere

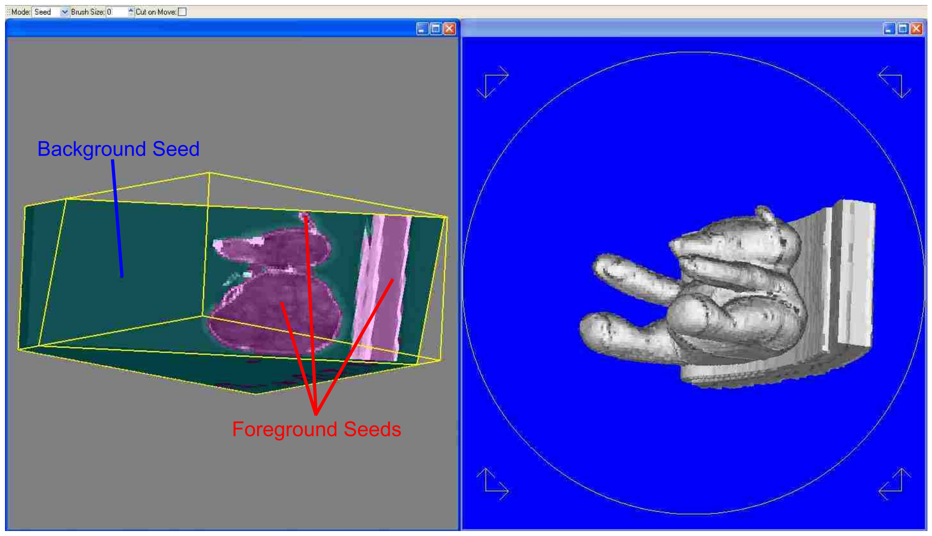

Mode: Seed Brush Size: 0 Cut on Move: ☐

Figure 3.1: Visual Interface for Live Surface. The user interacts with cross sections of the image volume (left). The Resulting surface is rendered (right). The user may then interact with the rendered surface.

within the object; likewise the background seeds should be representative of the background and placed anywhere outside of the object. Foreground and background seeds are shown as tinted voxels (red and blue respectively). Whenever the user releases the mouse button, the entire volume is tinted magenta and cyan, for foreground and background respectively, illustrating what the user has currently selected. The user may also choose the option of updating the selection with each mouse movement rather than when the mouse is released.

The view containing the selected surface is also updated whenever the data view is updated to indicate the selection. The surface view shows a smooth 3D rendering of the surface of the selected object. The user can also interact with this surface. The user can place both foreground and background seeds on the rendered surface. This will grow or cut away from the selected object in the areas where the foreground or background seeds are placed.

3.2 Components Overview

There are two main phases to Live Surface: preprocessing, and interaction. The goal of the preprocessing phase of Live Surface is to compute everything possible *a priori* so as not to slow the interaction. The two main steps of the preprocessing phase are (1) computing the tobogganed watershed hierarchy (Section 3.3) and (2) defining the surface of each catchment basin (Section 3.5). The principal algorithm of the interaction phase is Cascading Graph Cut (Section 3.4). During this phase Live Surface also renders the selected surface (also in Section 3.5).

Graph cut, as it is typically used, is unable to handle the massive number of voxels in image volumes at interactive speeds. The tobogganed watershed hierarchy simplifies the image volume and allows for faster segmentation. At the same time it preserves natural boundaries in the volume. Voxels are grouped into catchment basins using the tobogganing watershed algorithm [14]. Catchment basins are grouped into larger basins using a similar algorithm [26]. The process is repeated until all basins are grouped into a single basin, thus generating a complete hierarchy. Each level of the hierarchy forms an adjacency graph for the graph-cut algorithm. The surface of each catchment basin is also predefined in order to speed rendering during interaction.

Cascading Graph Cut is a novel extension of graph cut that takes advantage of the hierarchical graph structure of the watershed hierarchy. The general idea, illustrated in Figure 3.2, is to estimate the surface by making a graph cut at the top of the hierarchy (a) and then using graph cut to refine the selection at the next level of the hierarchy (d), only taking into consideration the basins close to the estimated surface to allow for a faster cut (b). Once CGC has cascaded to the bottom of the hierarchy the resulting surface *is* the selection. This resulting surface is rendered by looking up the surfaces of the contributing catchment basins and rendering them with OpenGL.

3.3 The Watershed Hierarchy

Voxels are grouped into catchment basins using the tobogganing watershed algorithm [14] (using the keep-sliding method [15] to handle plateaus). Catchment

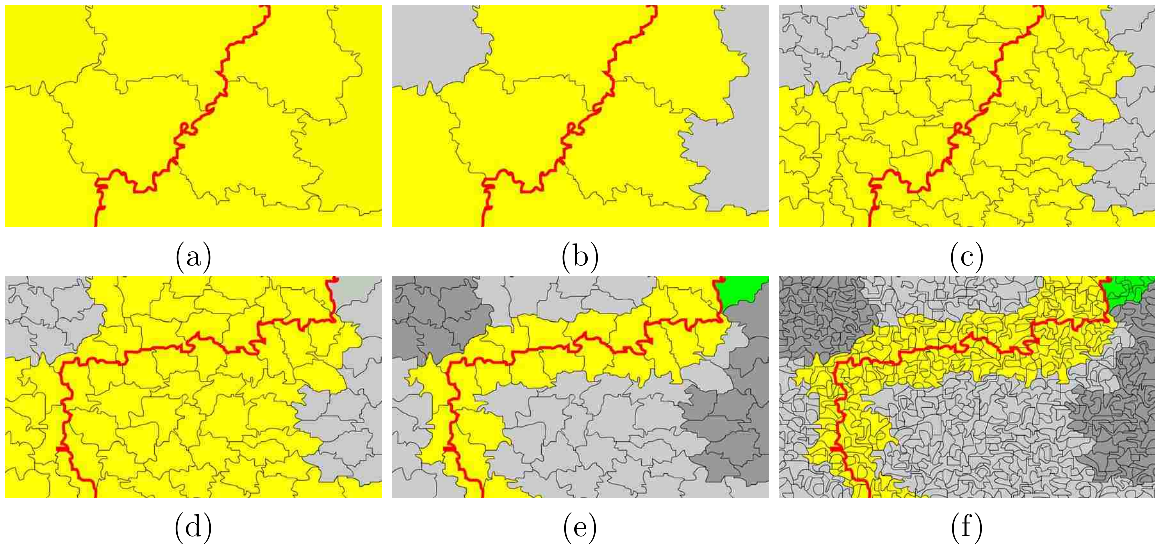


Figure 3.2: Simple Illustration of CGC. (a) A graph cut is computed to create an object surface (red). (b) The basins not bordering the segmentation are ignored (grey) leaving a smaller region of interest (yellow). (c) A graph composed of the children of the remaining CBs is prepared for a subsequent cut. (d) The result of the subsequent cut. (e) Repeat step b. Regions previously ignored (green) may be added to the region of interest if they now border the segmentation. (f) Continue to repeat the process.

basins are grouped into larger basins using a similar algorithm [26]. The process is repeated until all basins are grouped into a single basin, thus generating a complete hierarchy (Figure 3.3). Each level of the hierarchy forms an adjacency graph for the graph-cut algorithm.

3D tobogganed watersheds are a straightforward extension of their 2D counterpart (Figure 3.4). Figure 3.5 shows a 2D example of the generation of watersheds as implemented in Live Surface. First the gradient magnitude of each voxel is computed (red in Figure 3.5). Voxels are grouped into regions called catchment basins. In row-major order each voxel is recursively grouped with ("slides" to) its 6-connected neighbor whose gradient magnitude is the smallest (blue arrows in Figure 3.5), with the condition that the neighbor's gradient magnitude is less than or equal to that of the voxel itself. A catchment basin is thus composed of all voxels that slide to the same local minimum (blue squares in Figure 3.5).

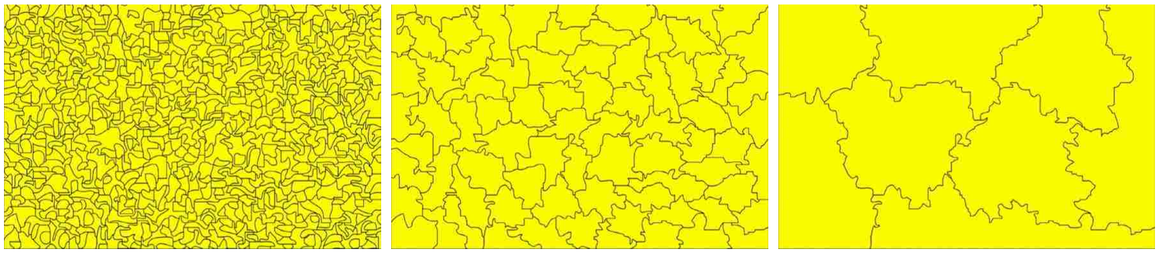


Figure 3.3: Simple illustration of grouping catchment basins. Catchment basins are grouped into larger basins in the watershed hierarchy. This is the hierarchy is used in Figure 3.2.

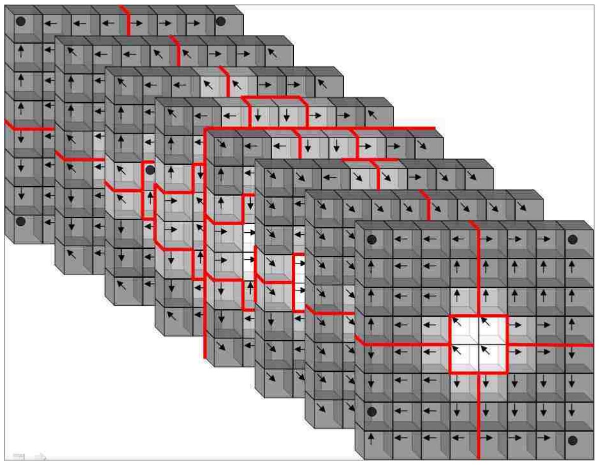


Figure 3.4: Simple illustration of tobogganing in 3D. Arrows indicate the slide direction ($\nwarrow$ and $\searrow$ indicate slides in the $\pm z$ direction). Dots are local minima. Red lines show the boundaries of catchment basins.

Once the catchment basins have been generated an adjacency graph keeps track of neighboring basins and edge weights (Figure 3.6). The graph represents the 3D adjacency of all basins at the same level of the hierarchy. Each node is a basin. Each edge represents the adjacency of a pair of basins. Live Surface currently uses the metric described in Lazy Snapping [20] to weight the edges in the graph. The weight is computed as

$$w(m,n) = \frac{1}{1 + \|\overline{C_m} - \overline{C_n}\|^2} \tag{3.1}$$

where $w(m,n)$ is the weight of the edge between m and n, and $\overline{C_m}$ and $\overline{C_n}$ are the

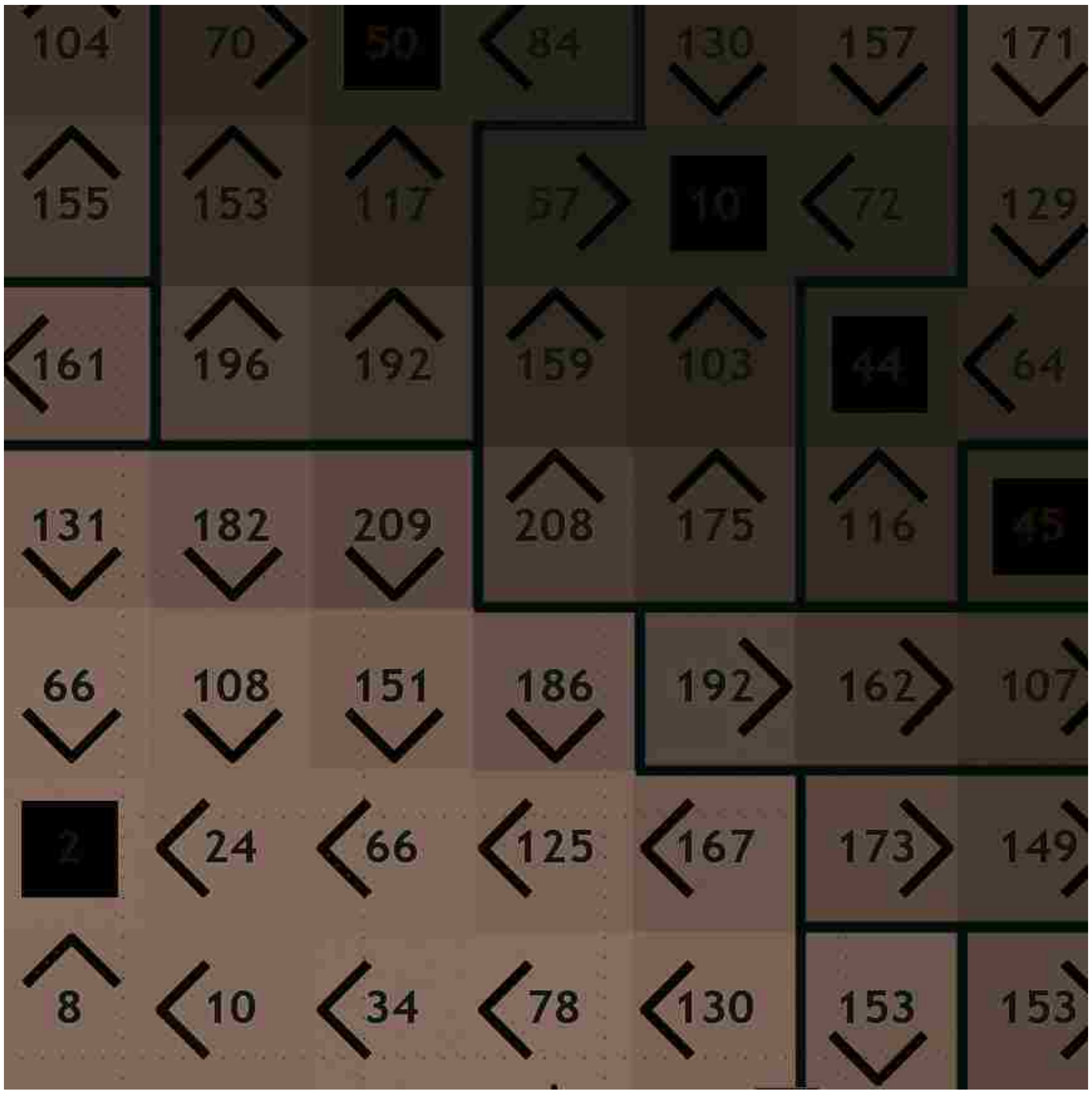


Figure 3.5: Tobogganing on voxels. The red numbers are the gradient magnitude of each pixel. The blue arrows indicate the slide direction of each pixel. The blue squares are the local minima. The green lines are the boundaries of the catchment basins.

average color vectors of basins m and n.

The watershed hierarchy further simplifies the data by grouping basins together. A similar method to that used for grouping voxels is used to group catchment basins for the next level of the hierarchy, as illustrated by Figure 3.7. Each basin slides to the basin most similar to itself (blue arrows in Figure 3.7). Similarity is measured using the 2-sample t-score difference between adjacent regions as used in Intelligent Paint [26]; the t-score value t is computed as

$$t^2 = \frac{N_m N_n \|\overline{C_m} - \overline{C_n}\|^2}{N_m V_n + N_n V_m} \tag{3.2}$$

where N is the number of voxels in the basin (the subscripts m and n indicate which basin), $\overline{C}$ is the average color vector of the basin, and V is a measurement of the color variance of the basin. V is computed as

$$V = \frac{1}{N} \sum_{k \in n} \|\overline{C_n} - C_k\|^2 \tag{3.3}$$

where C_k is the color of some voxel k in the basin. The t-score models the distributions of colors in neighboring basins parametrically, allowing us to quickly compare neighboring basins. All basins that slide together form a larger basin. Our data structure keeps track of all parent and child basins beginning to generate a hierarchy (Figure 3.8).

As the process repeats the hierarchy continues to generate. An adjacency graph is created at each level of the hierarchy. Every time tobogganning is computed on the basins the number of basins is reduced and a smaller adjacency graph is generated. This can be seen in Figure 3.9. The process continues until a single basin remains. This is what is meant by a *complete* watershed hierarchy. Each basin is a node in the hierarchy, with its children nodes being the basin from which it is composed. Each level of the hierarchy represents a level of image granularity. Each level of the hierarchy forms a complete partition of the image volume. The basins at each level of the hierarchy are mutually exclusive and exhaust the space. The watershed hierarchy preserves the natural edge information in the image volume, thus preventing the loss of detail resulting from a traditional resolution pyramid.

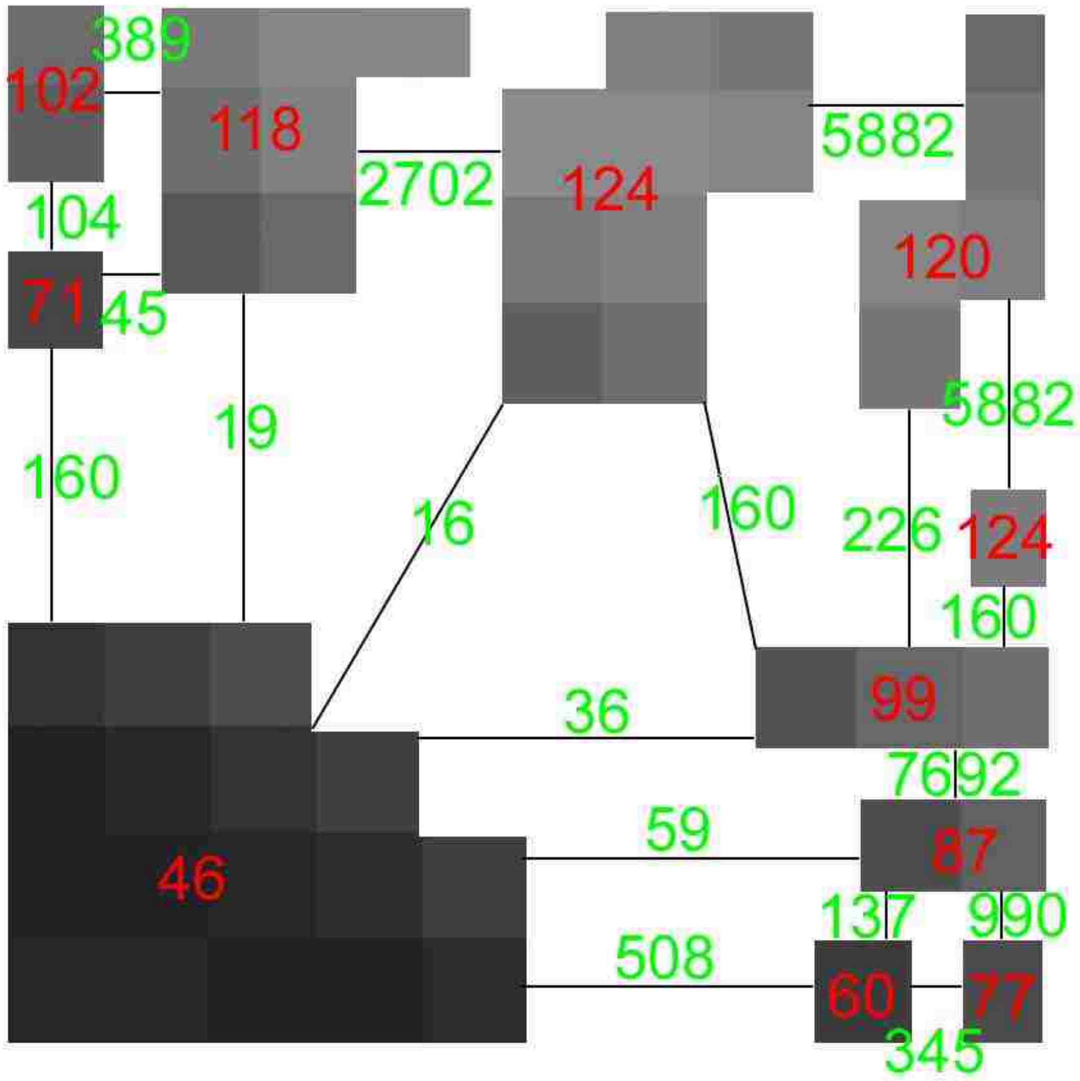


Figure 3.6: Adjacency graph of neighboring catchment basins. The red numbers are the average intensity of the regions. The green numbers are the weights of the edges computed with Equation 3.1.

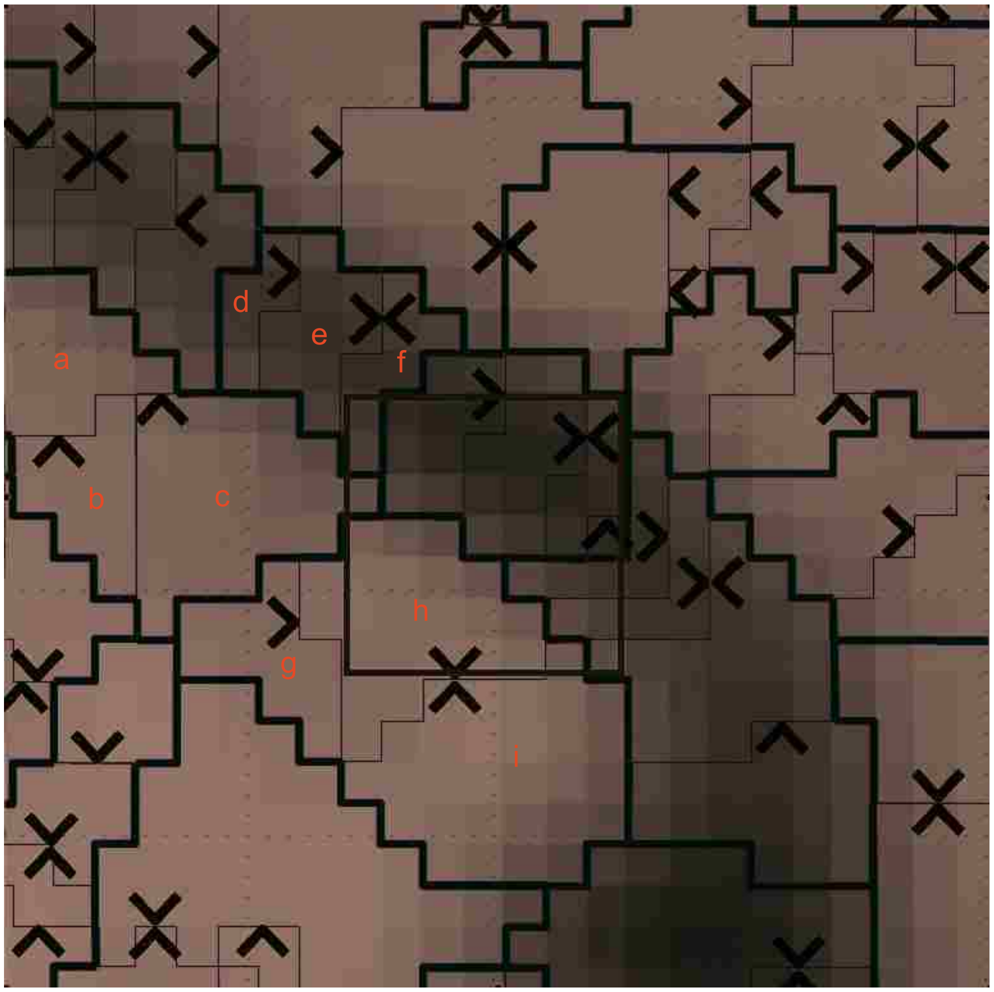


Figure 3.7: Tobogganing on basins. Figures 3.5 and 3.6 show the area in the red rectangle. The red letters correspond to the letters in Figure 3.8. The blue arrows indicate the slide direction. The light green lines are the previous basin boundaries. The heavier green lines are the resulting basin boundaries.

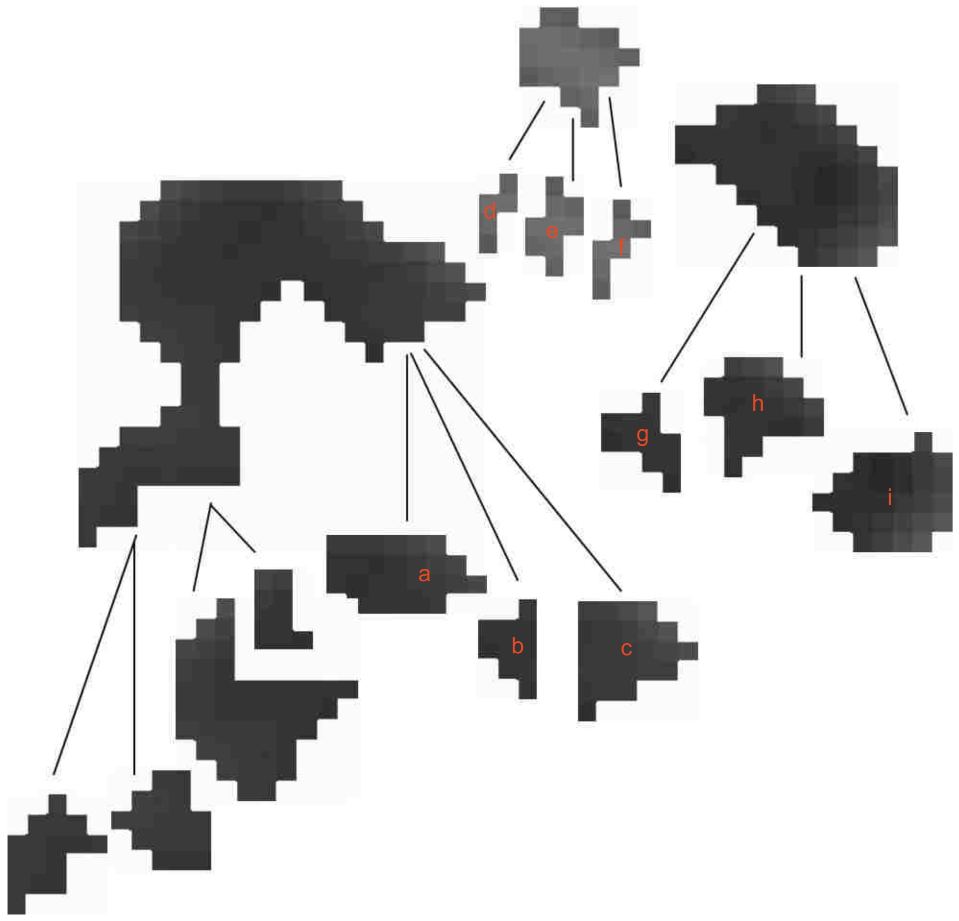


Figure 3.8: The beginnings of a watershed hierarchy. The red letters correspond to the letters in Figure 3.7.

3.4 Cascading Graph Cut

The goal of Live Surface is to extract and render a surface that evolves as the user moves the mouse. The mouse input specifies the foreground and background seeds which are applied to the watershed hierarchy. The volume is then segmented using Cascading Graph Cut. Finally, the result of the segmentation is rendered using the predefined surfaces.

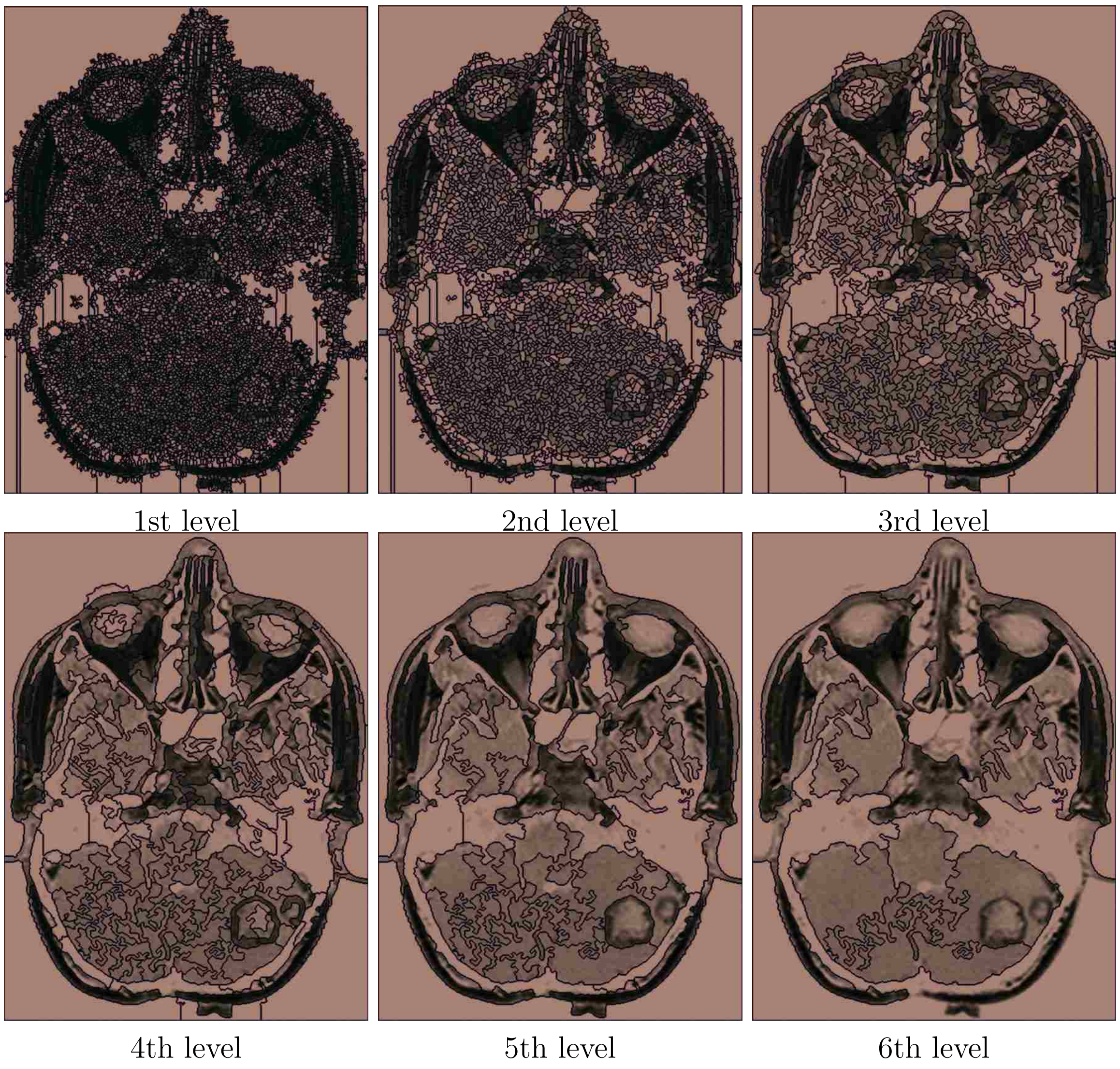


Figure 3.9: Four levels of a watershed hierarchy. A 2D example of the reduced number of basins as the hierarchy is generated showing four successive levels of the hierarchy (typically 9–12 levels in 3D). Notice the edges of basins adhere to natural edges in the image.

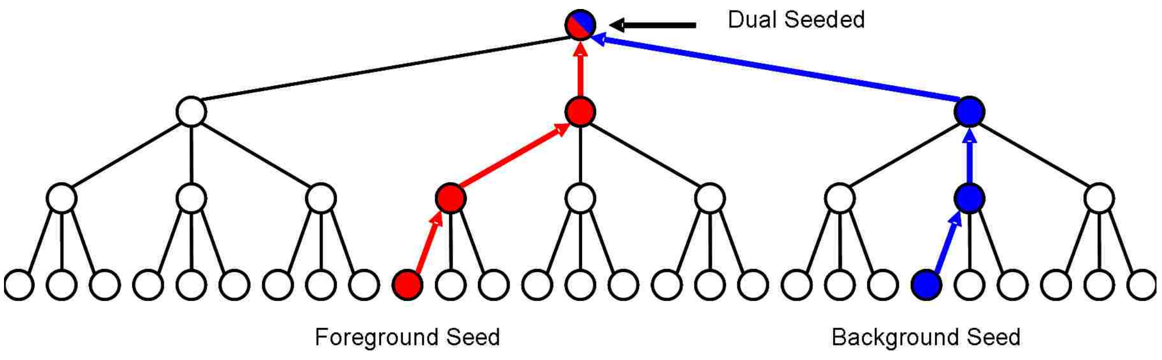


Figure 3.10: Propagation of seeds and dual-seeded basins. Seed propagate up the hierarchy. Dual seeded basins contain both foreground and background seeds.

3.4.1 Seeding

The user extracts a surface by specifying seed voxels within the volume. The placement of foreground and background seeds allows the user to guide the segmentation. Each seed the user specifies in the volume must be represented at each level of the hierarchy (Figure 3.10). When a foreground (red) or background (blue) seed is placed, it is propagated up the hierarchy from child to parent until it reaches the top. Some basins then contain both foreground and background seeds (shown as a mix of red and blue). Such "dual-seeded" basins indicate that the user disagrees with the hierarchy's aggregation of certain basins and thus signals Cascading Graph Cut that further refinement is needed in that area.

3.4.2 Cascading Graph Cut

Figure 3.11 shows an example of seeds placed by the user in this MRI scan of a knee. The surface shown on the right is rendered immediately ($<$ 1 second) after the user's interaction. There is a lot more going on behind the scenes to allow for the quick segmentation. The key to the fast segmentation and the main contribution of Live Surface is Cascading Graph Cut (CGC).

The intent of CGC is to provide hierarchical speedup by reducing the number of nodes in the graph to be cut. Figure 3.12 shows the intermediate results of each

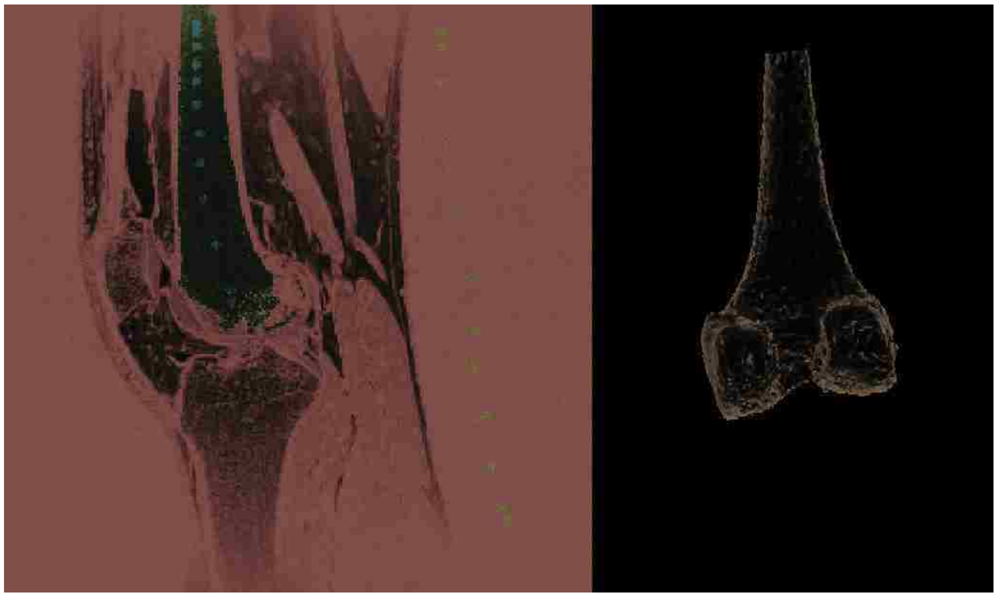
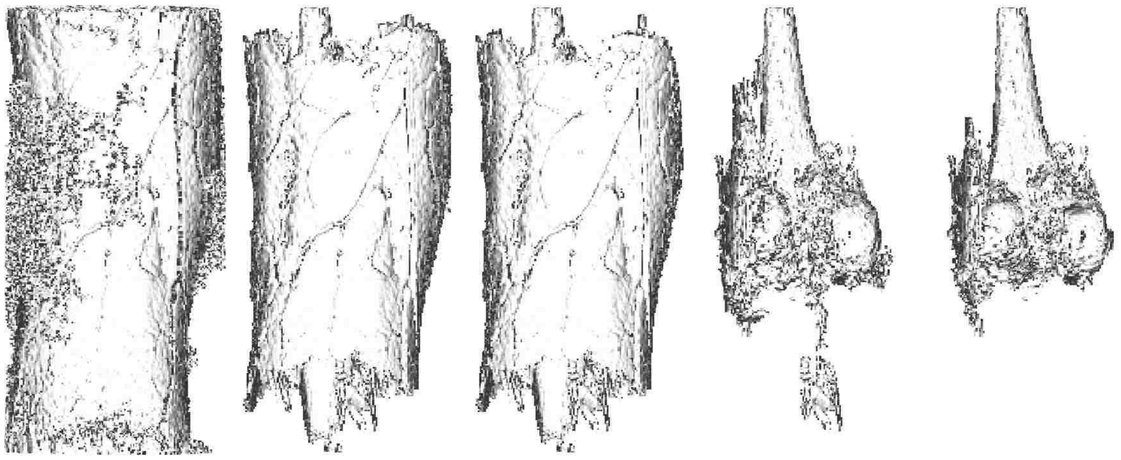

(a) (b)

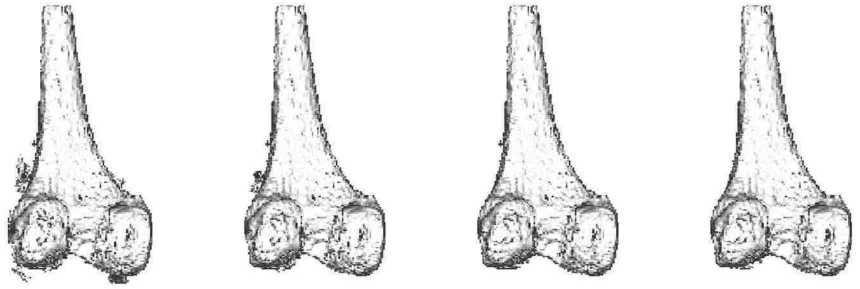
Figure 3.11: An example segmentation of the femur from an MRI of the knee. (a) A cross section on which the user has placed foreground seeds (red) and background seeds (blue). The selection is highlighted magenta and the background is highlighted cyan. (b) The surface of the 3D object selected.

recursive iteration of CGC. CGC begins by making a coarse cut at the top of the hierarchy (Figure 3.12a). The process then moves down to the next level of the hierarchy and locally refines the surface, taking into account only basins near the previous surface (Figure 3.12b). Near the top of the hierarchy refinements have the potential to drastically change the selected surface because basins are large (Figure 3.12c–e). As the process continues to lower levels of the hierarchy, refinements become more and more subtle (Figure 3.12f–h). This process is similar to the process followed by a sculptor; first large portions are cut away and then gradually smaller and smaller portions are refined. As the surface is refined at the last few levels, the selection more and more closely approaches the final result. Only the final result is actually rendered (Figure 3.11b). Figure 3.12 is given to illustrate what goes on behind the scenes.

The 2D example in Figure 3.13 illustrates CGC in further detail. Figure 3.13a shows a cross section of a sagital view of a tumor in the frontal lobe of a human brain.



(a) 9th level (b) 8th level (c) 7th level (d) 6th level (e) 5th level



(f) 4th level (g) 3rd level (h) 2nd level (i) bottom level

Figure 3.12: Step-by-step renderings of a single execution of CGC. Notice the coarse segmentation and the large changes near the top of the hierarchy and the finer segmentation and small changes near the bottom.

Foreground and background seeds have already been placed and can be seen in blue and red. The green lines in Figure 3.13b indicate the boundaries of catchment basins at some level in the hierarchy. The yellow line added in Figure 3.13c is the surface found by CGC at this level.

The recursive CGC does the following at each level of the hierarchy. CGC first ignores the blue background basins and the red foreground basins shown in Figure 3.13d because they do not border the surface and are not dual seeded. Second, gathering only the children of the gray remaining basins CGC forms an adjacency graph. Figure 3.13e shows the basins included in that adjacency graph. All of the ignored basins are consolidated into a single background and a single foreground seed

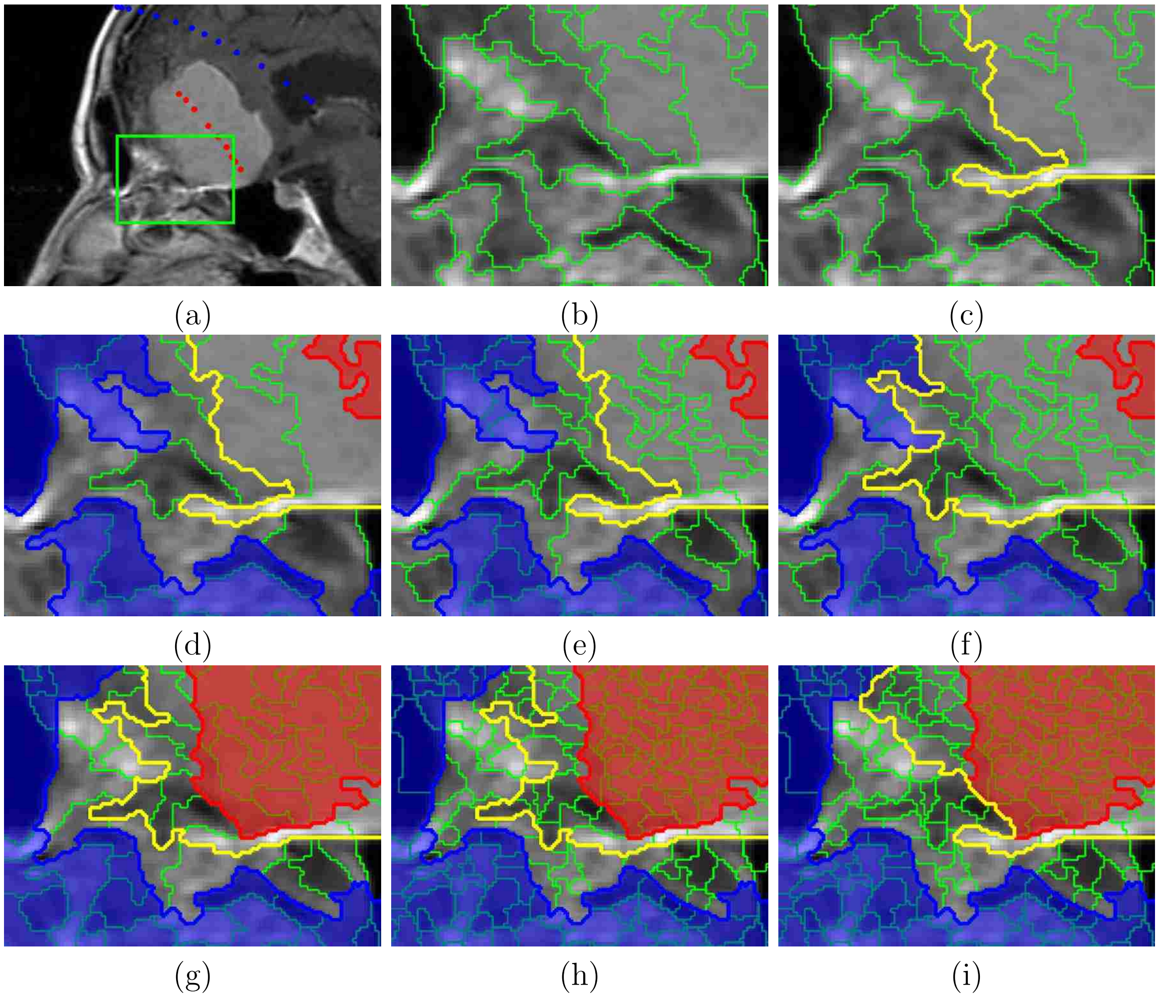
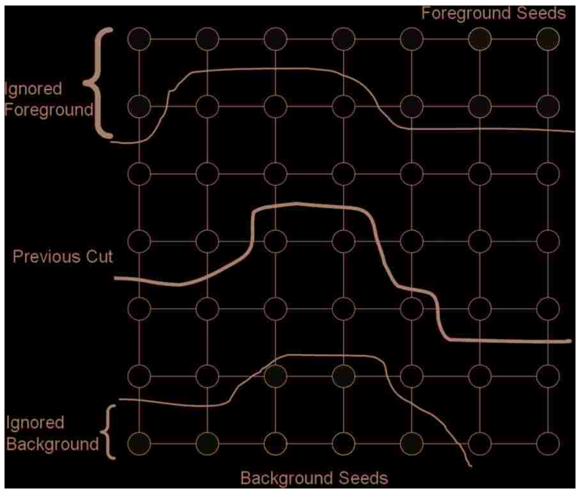
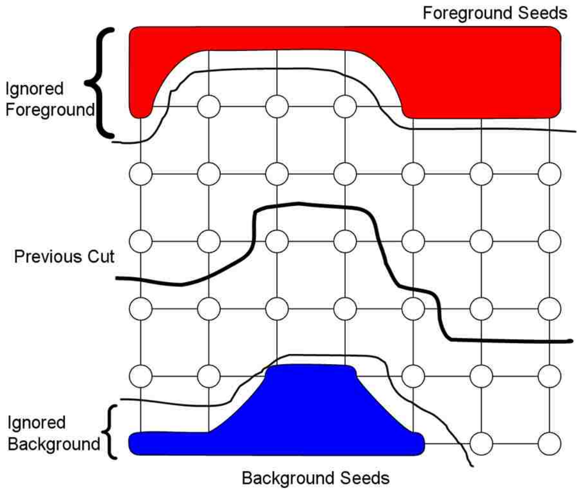


Figure 3.13: CGC illustrated in detail. (a) A cross section with seeds displayed in red and blue. (b) The basins making up the adjacency graph. (c) Segmentation at this level. (d) Red tinted foreground and blue tinted background not on the surface ignored. (e) Graph made from the children of the remaining basins. (f) New graph cut to refine the surface. (g) Basins not on the surface ignored. (h) Graph made from the children. (i) New graph cut to refine the surface.



(a)



(b)

Figure 3.14: Forming an adjacency graph as a recursive step of CGC. (a) The graph at some level n of the hierarchy showing the surface found at level $n + 1$, the current ignored nodes, and the foreground and background seeds. (b) The graph that will be cut at this level of the hierarchy showing the new foreground seed and background seed in the place of the ignored nodes.

basin. The reason for consolidating these regions into two seed basins is explained below. Third and finally, now that the new graph is defined, CGC computes a graph cut to refine the surface, resulting in Figure 3.13f.

Figure 3.14 shows a simpler example to illustrate why CGC consolidates the ignored basins into foreground and background seeds. Notice that all of the seeds (shown in bright red and blue in Figure 3.14a) are in the ignored foreground and background regions. For this reason CGC replaces all basins in the ignored regions with one foreground and one background seed as shown in Figure 3.14b. All edge weights remain the same.

CGC repeats the previous steps ignoring the basins not on the surface (Figure 3.13g). Notice that some basins that were previously ignored are now included because they lie on the surface of the cut. CGC again creates a graph from the children of the remaining basins (Figure 3.13h). Finally, CGC refines the surface with another graph cut (Figure 3.13i). CGC continues to repeat this process until it reaches the bottom of the hierarchy. As CGC cascades down the hierarchy the vast majority of basins are ignored. Notice that the gray region becomes narrower after each graph cut (Figure 3.13d, g). The segmentation speed no longer depends on the size of the volume but the surface area of the object selected. Since the watershed hierarchy is in fact a tree structure, determining which basins to include and which to ignore is done by simply traversing the tree. This allows for sub-second segmentation results. The resulting cut is the selected surface.

The recursive **CascadingGraphCut** algorithm in Appendix A.2 details this process further. It first computes a graph cut on the input graph G (line 1), classifying each basin as foreground or background. It then generates the next graph to be cut H (lines 3–14). The children of basins along the surface of the cut are included in H (line 5). Some of these basins may not be in G. The children of dual-seeded basins are also included in H (line 6). Each edge in H corresponds to an edge in the existing adjacency graph in the watershed hierarchy. Any edge connecting two basins in H is included (line 7). All edges from a basin in H to a basin not in H are represented in the cascaded graph by an edge from the basin in H to a generic seed basin (lines 10–12). The classification of the basin not in H determines which of the two generic seed basins to use (line 11). This has the effect of lumping each ignored basin into one of the generic seed basins.

3.4.3 Locally Adaptive Cascading

The following assumption allows for further acceleration of CGC. After the initial foreground and background seeds are placed, it can be assumed that further seeds will be placed with the intention of locally refining the surface. It is inefficient to compute a segmentation of the entire volume as the user refines the selection. In this case a local refinement method is preferable to a global segmentation. In order to accomplish this a variation of CGC is needed, locally adaptive cascading (LAC).

When the user places a seed in a basin that already contains the same kind of seed, the graph cuts performed on levels of the hierarchy at or above the previously seeded basin give no new information and so should not be performed. Instead, as

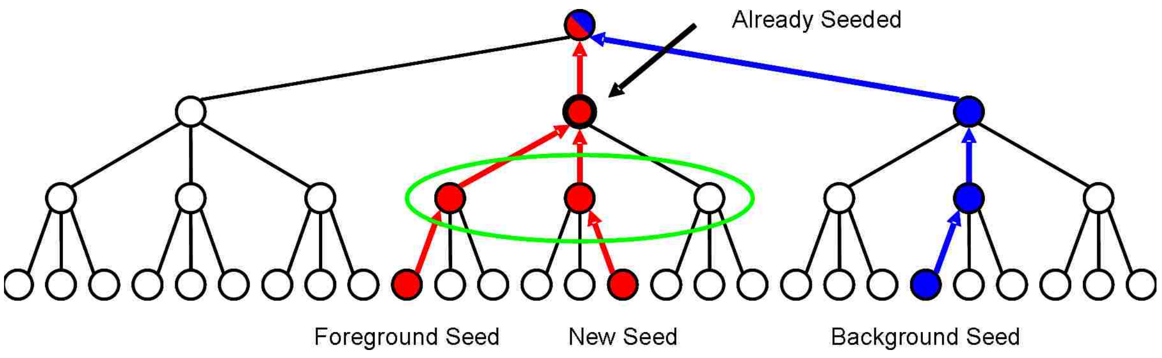


Figure 3.15: Seed propagation of LAC. The new red foreground seed propagates up the hierarchy until the basin already containing a foreground seed is reached. A CGC is then computed with only the children of that basin circled in green.

the seed is propagated up the hierarchy when it reaches the previously seeded basin the propagation stops. At this point LAC forms a graph from the children of the previously seeded basin and starts CGC (Figure 3.15). All of the ignored basins are consolidated into a foreground and background seed as usual with CGC. This localizes the bulk of CGC to the region near the new seed. However, as CGC cascades, basins that do not descend from the previously seeded basin may be included in CGC if they lie on the surface of one of the intermediate graph cuts. The effect is that cuts made by local refinements are integrated seamlessly into the globally optimal surface.

Appendix A.3 contains the detailed algorithm for **LocallyAdaptiveCascading**. When a user deposits a seed, it is propagated up the hierarchy until either the top of the hierarchy is reached or a basin is reached that already contains the same kind of seed (lines 4–8). If the top of the hierarchy is reached LAC begins a CGC with the graph at the top of the hierarchy as input (line 6). If LAC finds a basin that already contains the same kind of seed, it generates a graph with only the children of that basin, formed in the same manner as in CGC (lines 9–18).

The effect of this update appears as a growing surface like that shown in Figures 3.16 and 3.17. As the user adds new seeds, the selection is updated on average in less than a second.

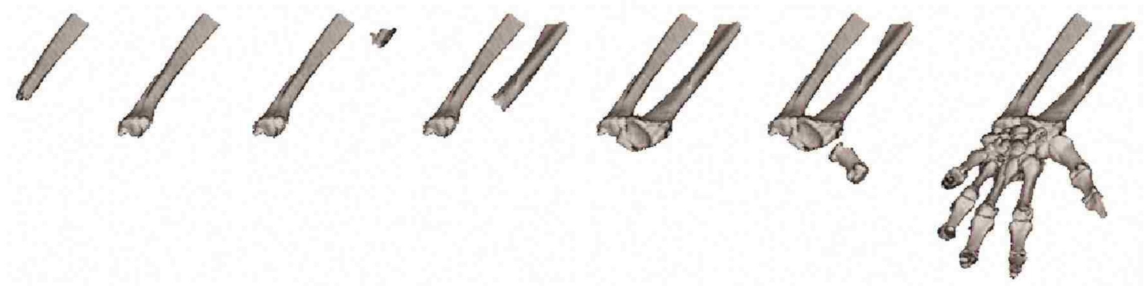


Figure 3.16: Evolving, growing surface. Selecting the bones in the arm and hand by dragging the mouse over them in the voxel display. The result is an evolving surface that appears to be growing.

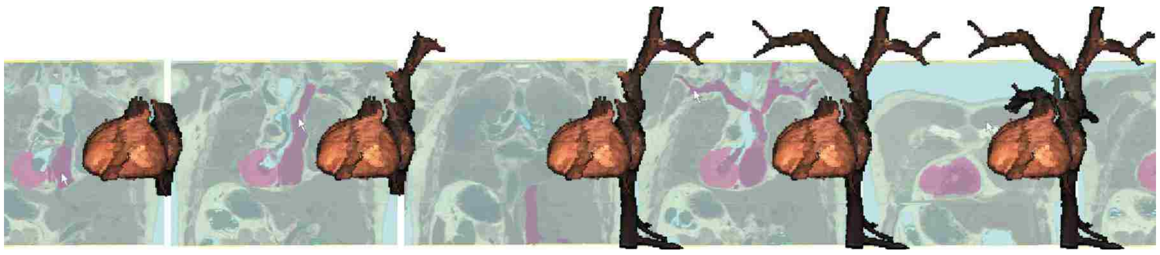


Figure 3.17: Growing the heart and arteries from the Visible Human. The rendered surface and the pseudocolored voxels are shown.

3.5 Rendering

A critical feature of Live Surface is the rendered surface of the selection. This allows for inspection of the results, which often leads to recognition of discrepancies. It also allows for improved interaction.

In order to speed rendering Live Surface predefines the visible surface of each basin at the lowest level of the hierarchy during the preprocessing phase of Live Surface. The surface of any selection at any level of the hierarchy is composed of these predefined surfaces. Live Surface stores a surface patch for each edge in the graph at the lowest level of the hierarchy in a lookup table. Each patch is the surface between two distinct neighboring basins. This gives us an entire honeycomb of possible surfaces to select from (Figure 3.18). Each patch can take on the color

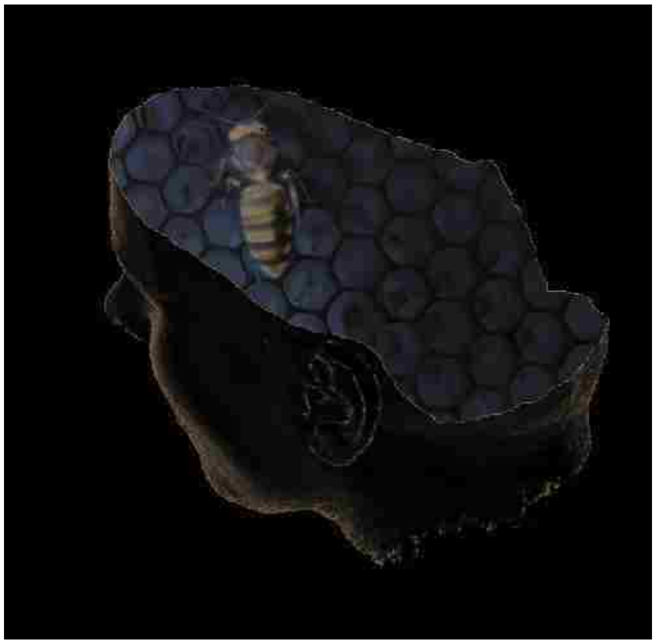


Figure 3.18: A honeycomb of surfaces. During preprocessing the surface of every catchment basin is generated and stored. This provides a honeycomb of surfaces to select from.

of either of the two basins which it separates depending on which is classified as foreground. Each patch is composed of actual voxel faces that are gradient shaded, the surface normal being the gradient direction of the voxel on the foreground side of that face. The surfaces are generated by sweeping through the volume and finding neighboring voxels belonging to different catchment basins.

After computing Cascading Graph Cut, once the bottom of the hierarchy is reached the resulting surface is rendered. The resulting surface is obtained by scanning through the edges of the graph at the lowest level of the hierarchy. Each edge that divides foreground from background is used to look up a surface patch in the table of predefined surfaces. The color and surface normals of the foreground basin are applied to the surface patch. OpenGL allows the evolving surface to be rendered at interactive rates.

3.6 Interaction Techniques

Interaction with Live Surface is generally accomplished using ray-casting techniques. This is true for interaction with the data and interaction with the rendered surface.

Placing seeds in the data view is done by backward mapping the pixels on the screen to voxels in the model. The tranfromation from model coordinates to the viewport and vice-versa are the following:

$$\begin{pmatrix} x_v \\ y_v \\ 1 \end{pmatrix} = V\ O\ R\ T \begin{pmatrix} x_m \\ y_m \\ z_m \\ 1 \end{pmatrix} \tag{3.4}$$

$$\begin{pmatrix} x_m \\ y_m \\ z_m \\ 1 \end{pmatrix} = T^{-1}\ R^{-1}\ O'\ V^{-1} \begin{pmatrix} x_v \\ y_v \end{pmatrix} \tag{3.5}$$

T translates the volume so that it is centered at the origin. R is the rotation matrix representing the current orientation of the volume due to user input. O is the Orthographic projection matrix which is defined as

$$O = \begin{pmatrix} \frac{1}{s} & 0 & 0 & 0 \\ 0 & \frac{1}{s} & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.6}$$

where s is half the length of the longest diagonal of the volume. Finally, V is the window to viewport tranformation and is defined as

$$V = \begin{pmatrix} \frac{w}{2} & 0 & \frac{w}{2} + x_0 \\ 0 & \frac{h}{2} & \frac{h}{2} + y_0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.7}$$

where x_0 and y_0 define the top left corner of the viewport and w and h are the width and height of the viewport. Since O^{-1} is undefined, O' is used in the backward mapping. O' is defined as

$$O' = \begin{pmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & c \\ 0 & 0 & 1 \end{pmatrix} \tag{3.8}$$

Equation 3.8 takes the ambiguous depth in eye coordinates and forces it to be at the current cross section depth c. The result indicates exactly which voxel was clicked by the user. In the case that the user clicks on the surface of the parallelepiped volume rather than within the cross section the voxel found will be outside of the bounds of the volume. In this case a ray is sent from this point in the viewing direction, $T^{-1}\ R^{-1}\ \hat{z}$, and a ray box intersection [27] is computed to find the point at which the ray enters the volume. If the ray intersects the volume the voxel at this point is where the seed is placed.

When interacting with the voxels in the cross-section view, the user spends a lot of time searching the volume by advancing the cross section and rotating the volume. Once the user has selected an object, it is natural and intuitive for the user to interact with the *rendered* surface of the object. The user can also more efficiently find discrepancies on the rendered model than in the cross-sectional data.

Interaction with the rendered Live Surface is accomplished using ray casting techniques, as illustrated in Figure 3.19. In order for the user to interact with the rendered surface, mouse clicks are interpreted as rays. The ray origin is found by backward mapping the mouse click from the viewport into model coordinates. In this case $-s$ replaces c in Equation 3.8. The ray direction is found in the same manner as mentioned previously. A simple ray-box intersection [27] is used to find the exact point and voxel where the ray enters the volume. The 3DDDA algorithm [28] computes the corresponding voxel on the currently rendered surface. This is a fast method for stepping from voxel to voxel along the ray until a voxel classified as foreground is found. If the user is placing background seeds, the first foreground voxel encountered becomes a seed. As a result the catchment basin containing that voxel and its similar neighbors (as determined by CGC) are subtracted from the object. This is illustrated

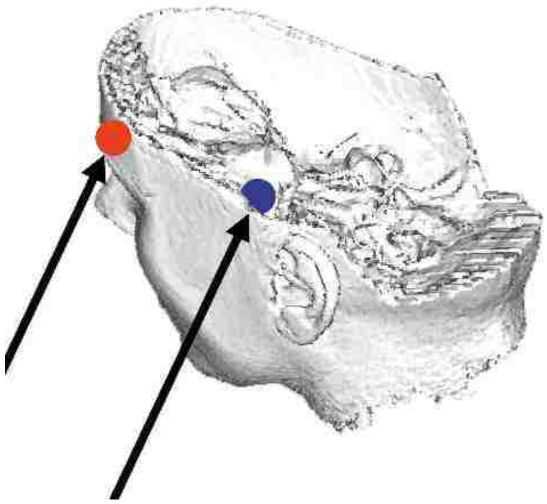


Figure 3.19: Placing seed on the rendered surface. Foreground seeds (red) and background seeds (blue) are placed just on the outside of and on the surface to augment or subtract from the object, respectively.

with the blue circle placed within the surface in Figure 3.19.

If the user is placing foreground seeds, the seed is deposited one voxel back along the ray on the last background voxel encountered before the first foreground voxel, as shown by the red circle just outside the surface in Figure 3.19. This allows the user to grow back areas where something is missing. The catchment basin containing the new foreground seed voxel is added to the object along with its similar neighbors. In both cases, augmenting or subtracting, the result is the same as if the user searched out the specific voxel on or under the surface in a cross section of the image volume and placed a seed in that manner.

Two examples of interaction with the surface of an object are shown in Figures 3.20 and 3.21. The first shows the use of surface interaction to remove the spout of the iconic Utah teapot. First the entire teapot is selected and a blue background seed is placed on its spout by clicking on the surface rendering (Figure 3.20a). A CGC is computed with the new seed, resulting in the surface shown in Figure 3.20b. Without the surface interaction the user would have to search for a cross section containing a portion of the spout. From start to finish the spoutless teapot can be generated in about 90 seconds interacting only with cross sections. When interaction with the surface is allowed, the entire process takes about 30 seconds. Objects can be much more difficult to identify in a cross section compared with the rendered surface. Furthermore, using the cross sections, it can be difficult and time-consuming to identify the voxels that correspond to a specific locale on the rendered surface.

The second example of interaction with the surface of an object is the completion of the femur bone in an MRI of a knee. Notice in Figure 3.21a that there is an obvious missing portion of the bone. A foreground seed has been placed just outside of this ragged surface. As a result the rest of the bone is grown back (Figure 3.21b). Thus, augmenting or subtracting from the surface is performed more naturally and efficiently by interacting directly with the rendered surface.

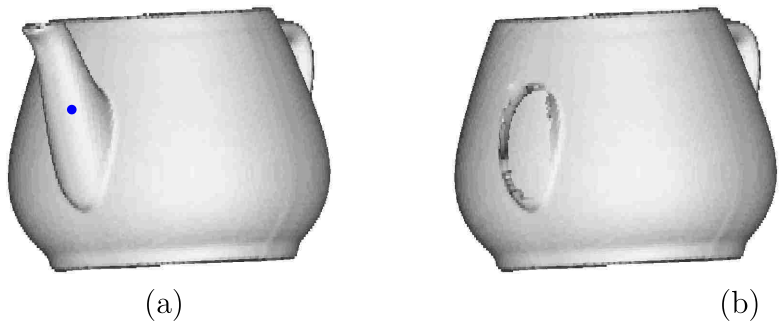


Figure 3.20: "Spouting off." Deleting the spout from the Utah Teapot by interacting directly with the rendered surface. Background seed is indicated by the blue circle.

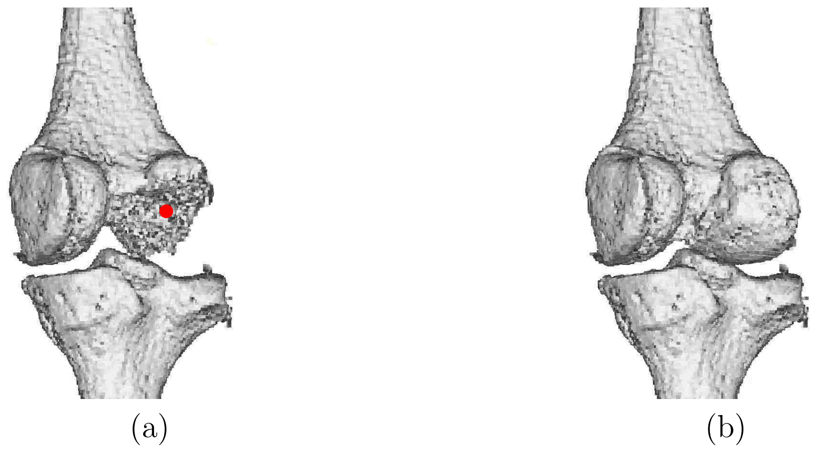


Figure 3.21: Surface interaction: growing. A portion of the femur is grown in the selection by interacting with the rendered surface. The red circle in (a) indicates a foreground seed.

Chapter 4

Results

This chapter contains an analysis of the performance of Live Surface on a variety of data. Both the run time (Section 4.1) and the accuracy (Section 4.2) are measured.

4.1 Run Time

Performance results for Live Surface using LAC applied to a variety of image volumes are summarized in Table 4.1. The data sets consist of CT, MRI, and the Visible Human Project. These results were obtained using a machine with a 3.6 Ghz Intel Xeon processor with 3 GB RAM. For image volumes ranging from 1 to 18 megavoxels, preprocessing required from 9.5 seconds to 50.6 seconds depending linearly on the size of the volume and the total number of regions at all levels of the hierarchy. This includes the time spent generating the 3D hierarchy and predefining surfaces. User interaction required at most 1.5 minutes. Interaction time is indicative of the number of seeds specified by the user, and includes time spent searching cross sections, examining the resulting 3D surface, and the actual cutting and rendering. The average time to compute LAC for all volumes was 0.203 seconds, with an average of 0.148 seconds to render the result. Larger volumes generally require longer to cut than smaller volumes. However, the complexity of the watershed hierarchy and the surface area of the selected object greatly affect the average time required to make a cut. For example, the largest volume in Table 4.1 is the hand from Figure 4.7, which has a shorter run time for the graph cut algorithm than some of the smaller volumes. The knee and brain take longer because they are noisier data sets. This

	Size	Pre-process	User Time	Average Cut	Cut and Render	Memory
teapot (Fig 3.20)	170x170x192	18.5s	30s	0.17s	0.3s	83M
teddy (Fig 4.3)	128x128x62	9.5s	10s	0.051s	0.084s	246M
knee (Fig 4.5a)	256x256x127	40.8s	30s	0.44s	0.66s	573M
spine (Fig 4.6)	219x146x164	15.5s	30s	0.06s	0.1s	199M
hand (Fig 4.7)	492x240x155	50.6s	70s	0.14s	0.21s	308M
brain (Fig 4.8)	256x256x109	30.4s	55s	0.5s	0.85s	488M
kidney (Fig 4.14)	225x120x135	24.5s	35s	0.06s	0.13s	220M
VHP (Fig 4.15)	182x106x207	15s	90s	0.2s	0.47s	204M
Minimum		9.5s	10s	0.051s	0.084s	83M
Maximum		50.6s	90s	0.5s	0.85s	573M
Average		25.6s	45s	0.203s	0.351s	290M

Table 4.1: Performance results for 8 different image volumes. User time includes all the time the user spends interacting with the volume. The average cut time is the amount of time spent performing each Cascading Graph Cut. The cut and render time is the time spent in Cascading Graph Cut and the time spent rendering the result to the screen. Cutting each volume took on average 0.203 seconds ranging from 0.051 seconds on the teddy bear to 0.5 seconds for the brain. The average time from each interaction to a rendered results was on average 0.351 seconds ranging from 0.84 seconds on the teddy bear to 0.85 seconds on the brain. The memory is the total amount of RAM used during processing.

is evidenced by the deeper hierarchies and larger number of regions in the first level of the watershed hierarchy shown in Table 4.2. The teapot takes longer because of a larger surface area. Also, in the case of the teapot, cutting off the spout is more difficult because of its similarity in intensity to the rest of the teapot.

The average amount of RAM used by the algorithm was 290M. Most of the memory is needed for storing the hierarchy and the predefined surfaces for rendering. The number of 3D tobogganed regions in the initial hierarchy was 199,239 on average. The complete hierarchy ranged from 9 to 10 levels (Table 4.2). The number of levels for each volume corresponds to the size of the volume and the complexity of its content.

	teapot	teddy	knee	spine	hand	brain	kidney	VHP peel
voxel	5.5M	1M	8.3M	5.2M	18.3M	7.1M	3.6M	4M
L 1	63k	18k	197k	44k	38k	164k	553k	44k
L 2	15k	4.7k	49k	11k	9.6k	41k	115k	11k
L 3	3.9k	1.1k	12k	3k	2.5k	10k	3.7k	3.1k
L 4	1k	295	3k	786	649	2.5k	985	815
L 5	266	81	765	208	158	628	274	208
L 6	79	23	199	55	44	166	72	49
L 7	27	5	56	17	13	50	14	12
L 8	3	2	15	4	3	17	4	3
L 9	1	1	3	1	1	5	2	2
L 10			1			1	1	1

Table 4.2: Size of the watershed hierarchy. Numbers of regions at each level of the complete tobogganed region hierarchy for each of the 6 image volumes.

4.2 Accuracy

Accuracy is demonstrated primarily in a qualitative manner; however, a brief quantatative analysis is also included.

4.2.1 Quantatative Analysis

Graph cut has been shown to produce accurate results in medical imaging [21] and desirable results for video [23, 24]. Lombaert [25] shows that a coarse-to-fine approach results in a good approximation, lending great confidence to the quality of our results. However, an important part of future work will be to quantitatively compare surfaces obtained from our Cascading Graph Cut (CGC) with volumes containing objects of known geometry.

In this section correctness is measured by comparing the results to those of a voxel-level graph cut. For comparisons, the voxel-level graph cut used is formulated in the same manner as the graph cut on catchment basins. Each voxel is a node in the graph with edges connecting adjacent (6-connected) voxels. Edge weights are computed using Equation 3.1, replacing the average color of the basin with the actual color of the voxels. Error is measured by finding the Manhattan distance between each foreground surface voxel in the Live Surface result and the nearest foreground

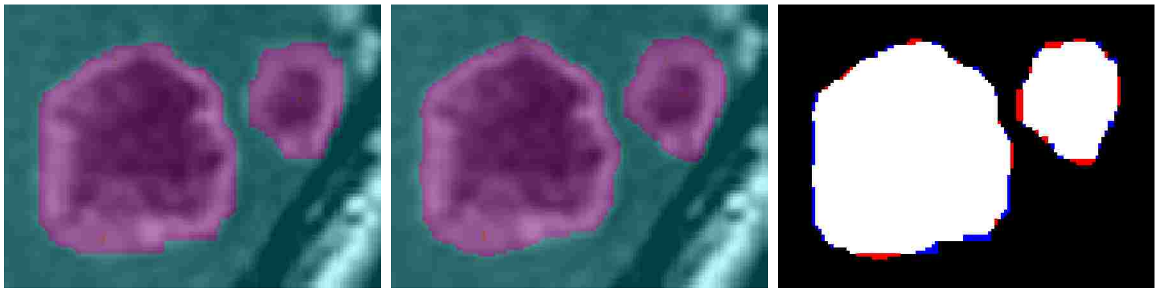

(a) Voxel-level result (b) CGC result (c) Difference

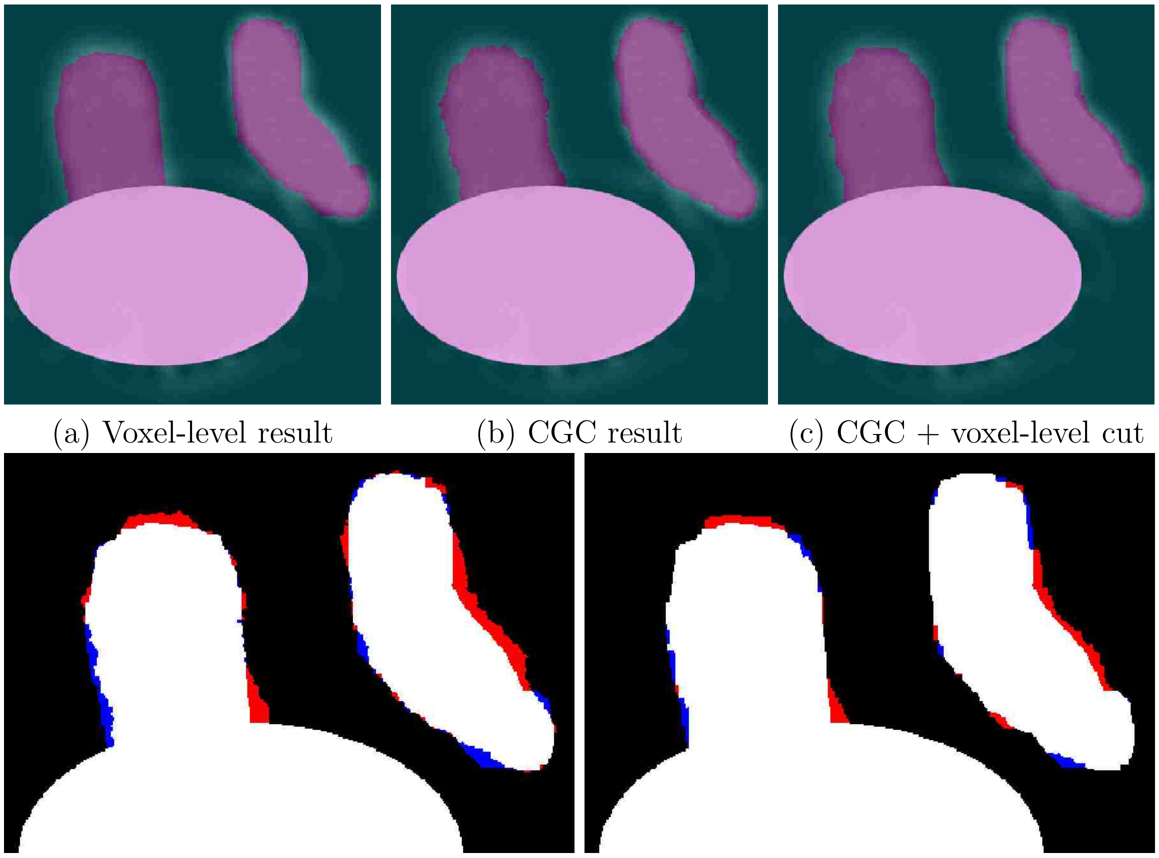
Figure 4.1: Comparing voxel-level graph cut to CGC. (a) the voxel-level result. (b) the result of CGC using the same seeds. (c) the differences in the result: white for true positives, black for true negatives, red for false positives, and blue for false negatives. The result of an extra cut on the voxels in the regions on the surface of the CGC result is the same as (a).

surface voxel in the voxel-level result. The difference between the voxel-level graph cut and CGC or the "error" is the average of this distance for each foreground surface voxel in the Live Surface result.

This is illustrated by Figure 4.1, which shows the results on a 2D cross section of a CT scan containing a brain tumor. Using the exact same set of seeds, a voxel-level graph cut and a CGC are computed. Assuming that the voxel-level cut is accurate, Figure 4.1c shows the false positives (red) and the false negatives (blue) of the CGC. Note that the exact boundaries computed are different (avarage Manhattan distance of 0.625899 pixels). However, the selection is generally the same.

In an attempt to improve the accuracy of Live Surface on this example, an extra graph cut has been performed. The graph for this extra cut contained only the voxels in the catchment basins at the lowest level of the hierarchy on the surface computed by CGC. In this case this extra cut exactly matches the voxel-level result in Figure 4.1a. This extra cut has been purposely left out of CGC in order to provide greater speed. It can be included as a postprocessing step if necessary.

The extra cut does not always perfectly match the voxel-level graph cut. This is illustrated by the synthetic image in Figure 4.2. Once again the exact same set of seeds is used to generate the three segmentations shown. While the Live Surface


(a) Voxel-level result (b) CGC result (c) CGC + voxel-level cut

(d) Voxel-level vs. CGC (e) Voxel-level vs. CGC + voxel-level cut

Figure 4.2: Further comparisons of CGC to voxel-level graph cut. This is a synthetic example illustrating that Live Surface does not always provide a result that exactly matches the result of graph cut applied directly to the voxels, even when refined all the way to the voxels themselves. (a) voxel-level graph cut. (b) CGC (average surface error of 1.944444). (c) CGC with an extra voxel-level refinement (average surface error of 1.206995). (d) difference between voxel-level graph cut and CGC. (e) difference between voxel-level graph cut and CGC with the extra voxel-level refinement. Notice in (e) that the result is closer but still not the same.

result roughly matches the voxel-level result, even after the postprocessing step the result still differs from the voxel-level result in the exact location of the boundary. This suggests that further research should be applied to discover why this is the case.

4.2.2 Qualitative Analysis

This measurement of accuracy in the previous section makes the assumption that the voxel-level graph cut is 100% accurate. Figures 4.1 and 4.2, when examined qualitatively, shed doubt on that assumption. The voxel-level graph cut used here has a bias toward straight, axis-aligned boundaries. This is because the straight axis aligned boundaries require fewer edges in the graph to be cut. Live Surface, using tobbogganed regions as the basis of a graph cut avoid this tendency, resulting in a more natural boundary and preserving more low-level detail.

In this section the quality of the result is based on visual inspection. The teddy bear data set is an excellent example of how Live Surface is able to extract well defined surfaces with a single foreground and background seed. Figure 4.3 shows an image volume containing CT scans of the teddy bear. On the far left is the volumetric data. With just two clicks (one foreground and one background) the surface of the teddy bear is selected. Notice the ear is missing in the initial segmentation. The interactivity of Live Surface allows for a quick remedy to this where the surface is updated with a single foreground seed placed in the missing ear. Updates for the teddy bear are computed and rendered in 0.084 seconds, on average.

Application of CGC to skeletal anatomy is shown in the CT of a foot (Figure 4.4). Segmentation of the bones in the foot is challenging for several reasons. Because of the articulation of the joints, the bones lie in close proximity to one another and therefore, due to aliasing, the intensity of the voxels between the bones is increased making it difficult to separate them using standard thresholding or region-growing segmentation techniques. Furthermore, it is clear that the density of the bones is certainly not uniform (Figure 4.4a). This causes many simpler segmentation techniques to either lose the interior of the bones or, in order to select the interior, select the space between bones as well, fusing the bones together (Figures 4.4b–c). CGC

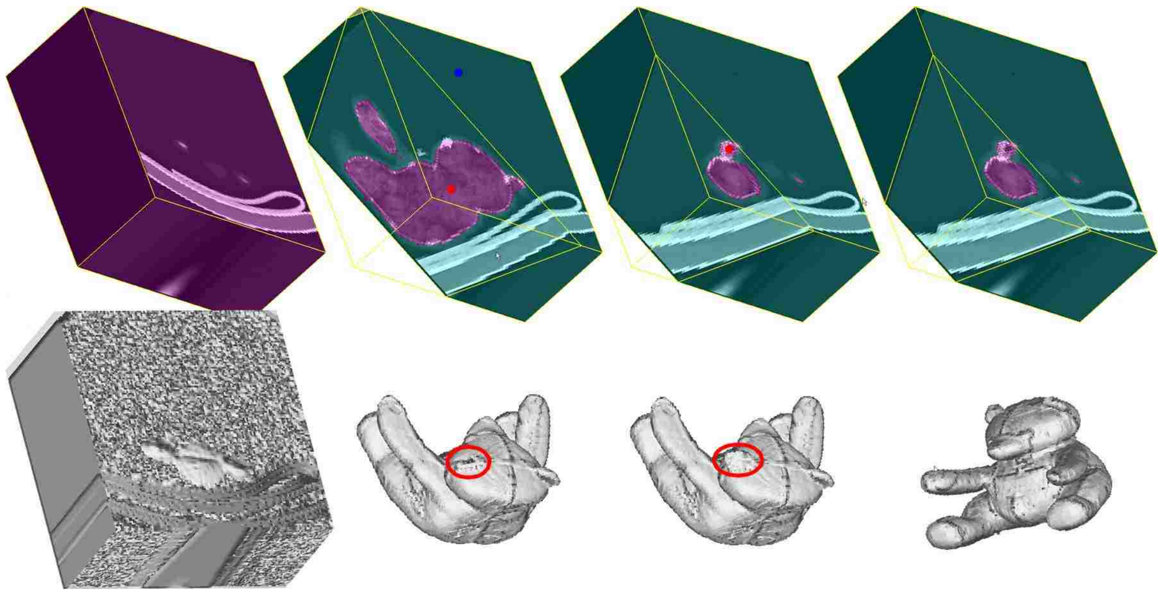


Figure 4.3: One-click segmentation. With one foreground seed and one background seed, the user interactively selects the teddy bear. Notice an ear is missing from "Teddy van Gogh". The user then finds the ear in the volume and places a foreground seed, and the final surface is updated instantly (0.084 seconds).

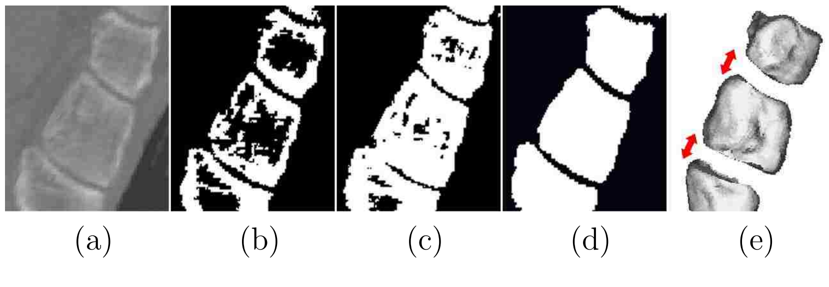


Figure 4.4: Live Surface vs. Thresholding. (a) Foot bones from a CT volume. Thresholding techniques often either (b) erode away the bones or (c) merge the bones. (d) Live Surface distinguishes between neighboring bones while including the less dense interior of bones. (e) The rendered surface shows that the bones remain separate throughout the volume. Note that the bones have been separated artificially to emphasize separability in the segmentation process.

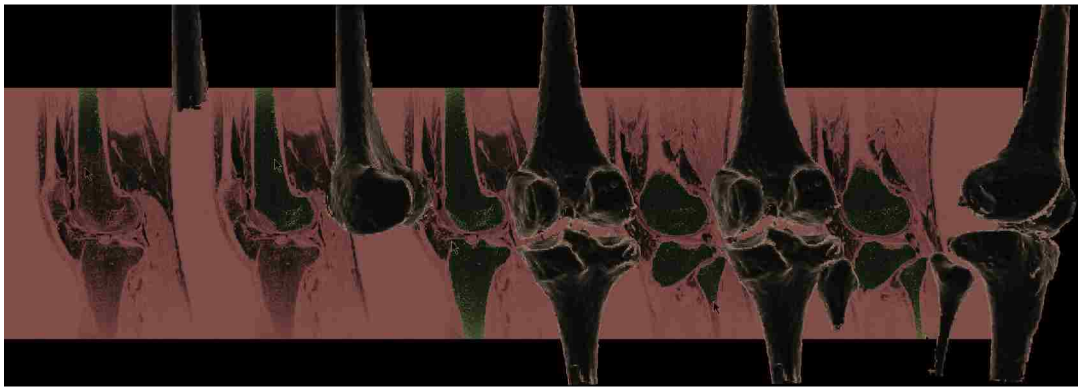
allows individual bones to be segmented and separated quite easily (Figure 4.4d–e).

MRI images are inherently more noisy than CT images, but this is another situation in which the optimal properties of graph cut excel. Figure 4.5 illustrates segmentation of the femur, tibia, fibula, and muscles of the knee from an MRI volume. In this example, CGC is still extremely efficient, completing each cut in an average of 0.44 seconds. The muscles are an example of the ability of CGC to segment not only the high contrast skeletal anatomy but also the low contrast, soft tissue.

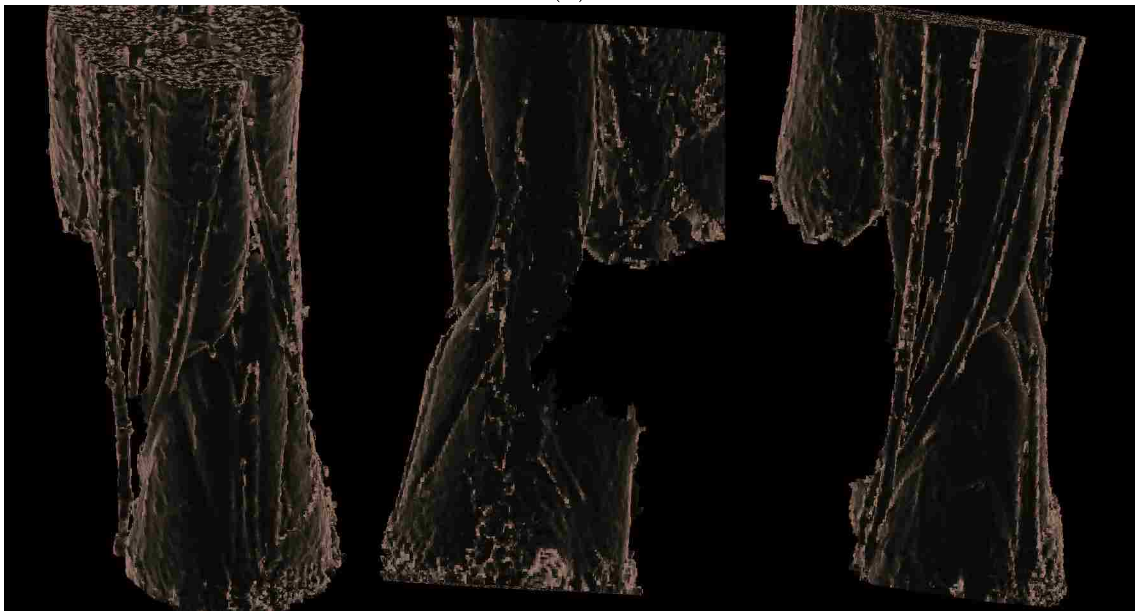
In Figure 4.6 the user selects the kidneys and pelvis from a CT scan. Selecting bones can be simpler because bone densities are considerably greater than surrounding tissues. Soft tissues create more difficulty because densities of neighboring organs or soft tissues are quite similar. This is another example in which segmentation based on graph cut is able to excel. Furthermore, the ability to combine both bone and soft tissue without making two contextually separate segmentations is another advantage of Live Surface.

The ability to simultaneously select different kinds of tissue allows Live Surface to generate some interesting cut-away views as illustrated by the hand in Figure 4.7. The radius and ulna are shown here in context with the hand. This allows the anatomy to be explored in context, and because of the segmentation and rendering speed, anatomical layers can be added or deleted interactively. This is further illustrated by the sequence of images in Figure 4.8. As the user places foreground seeds on the spinal cord, the face is mistakenly selected. With one more brush stroke of background seeds the user quickly removes the unwanted portion.

Live Surface can be applied to datasets generated from other imaging modalities as well. Figure 4.9 shows various surfaces from a poliovirus complexed with its cellular receptor. This dataset was reconstructed from images captured using cryogenic electron microscopy [29]. The voxel size of this data set is 0.363 (+/- 0.001) nm. From the data set shown in Figure 4.9a, the user can select the RNA genome (Figure 4.9b), the outer protein shell of the virus (Figure 4.9c), and the virus complete with the poliovirus receptors (Figure 4.9d).



(a)



(b)

Figure 4.5: Live Surface applied to MRI. (a) Knee bones and (b) leg muscles from an MRI volume.

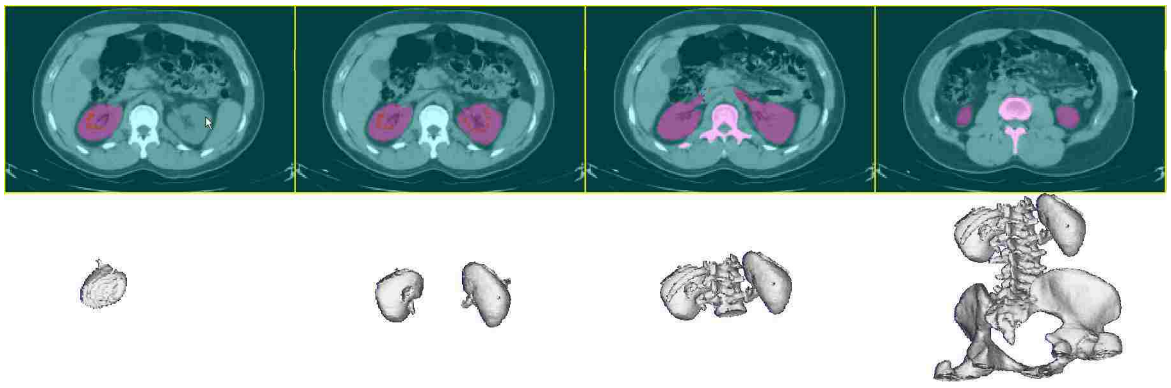


Figure 4.6: The spine, pelvis and kidneys selected from a CT volume.

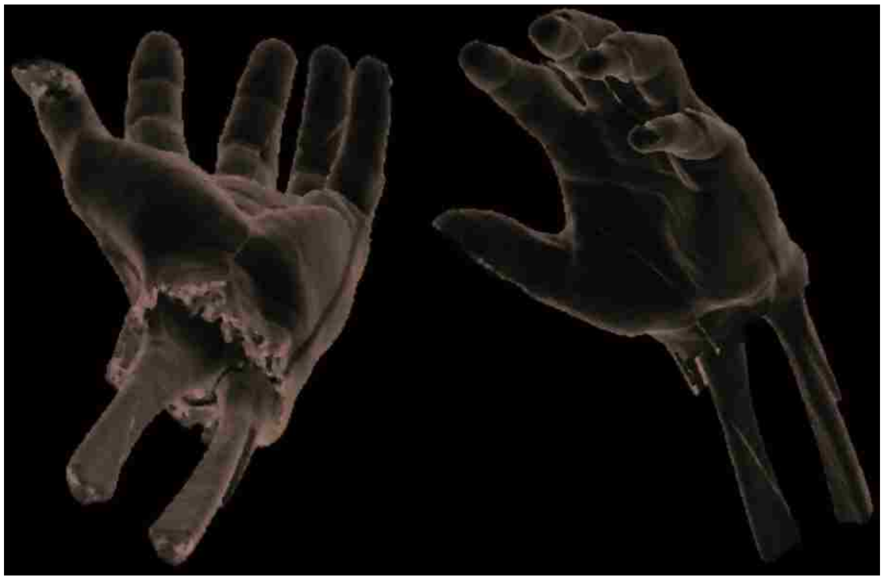


Figure 4.7: Combining multiple tissues. Skeletal and soft-tissue anatomy are segmented adaptively and simultaneously, allowing the user to explore both in context.

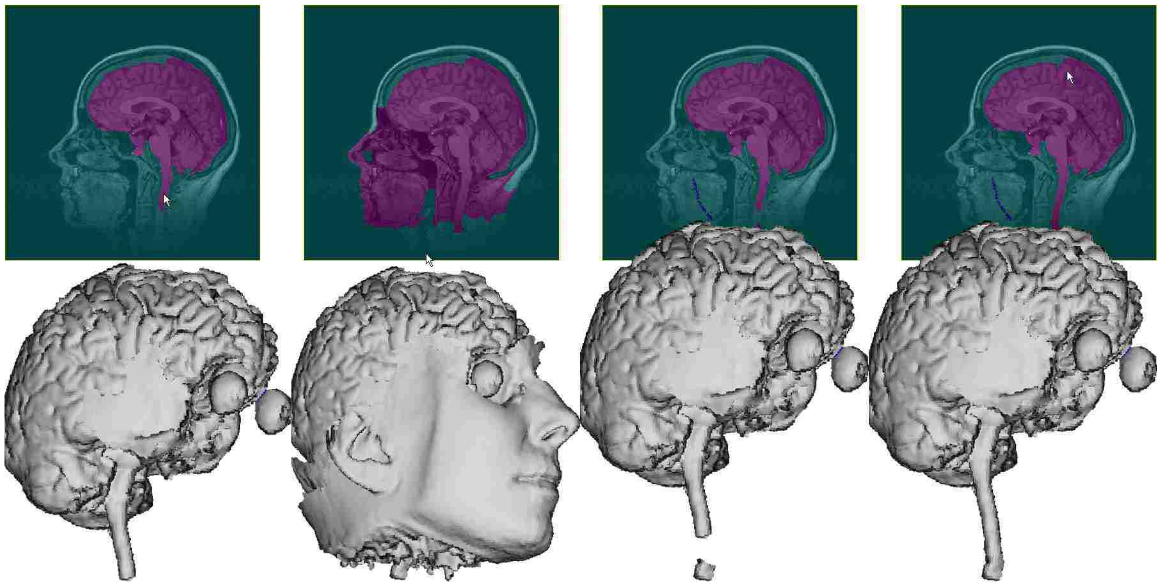


Figure 4.8: Step-by-step selection and deletion of portions of a head from an MRI volume.

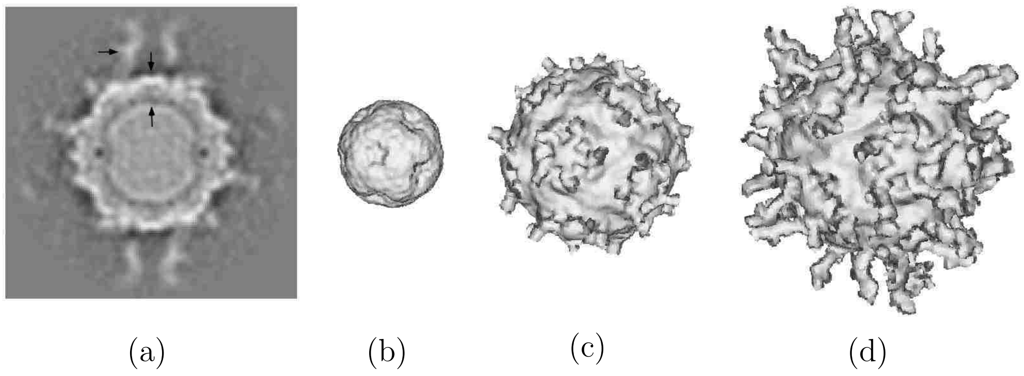


Figure 4.9: Poliovirus. Views of poliovirus complexed with its cellular receptor, as determined by cryogenic electron microscopy [29] and viewed with our algorithm. The max. diameter of poliovirus is 33 nm. (a) Slice through the volume. Whiter intensities represent higher particle densities. At increasing radius from the center of the particle, arrows correspond respectively to the outer diameter of RNA genome (inner diameter of the protein shell or capsid), the outer diameter of the capsid, and the cellular receptor. Surface rendering of (b) RNA genome, (c) protein shell (including stubs of the receptors), (d) poliovirus and receptor.

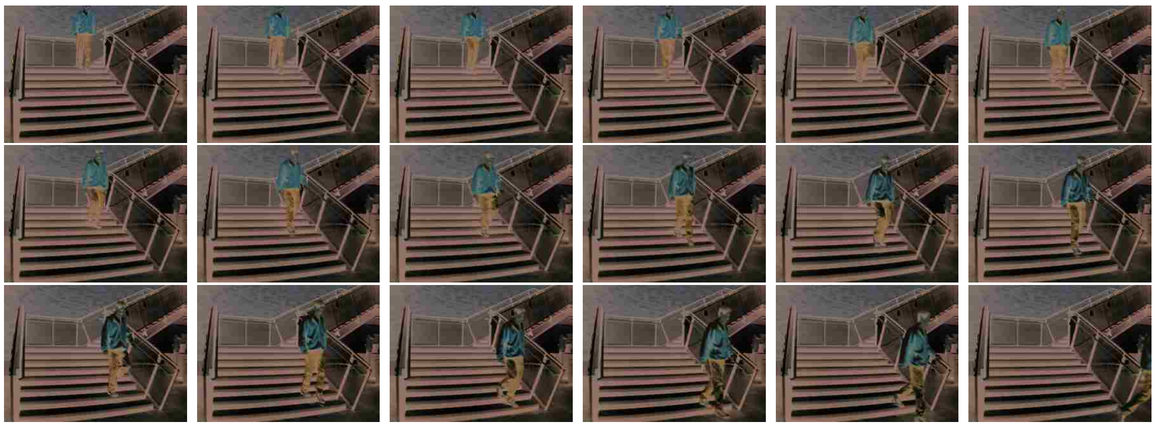
4.3 Application to Video

Live Surface was designed for volumes such as CT and MRI. However, video can also be segmented with Live Surface. Video demonstrates more significant changes in the "t" dimension than volumes in the z dimension, so video requires different treatment. One change to allow for this difference is to group regions within frames before grouping them between frames. For video Live Surface computes the first level of tobogganed regions within frames before grouping tobogganed regions between frames. A cine-loop window has been added, for verifying the results.

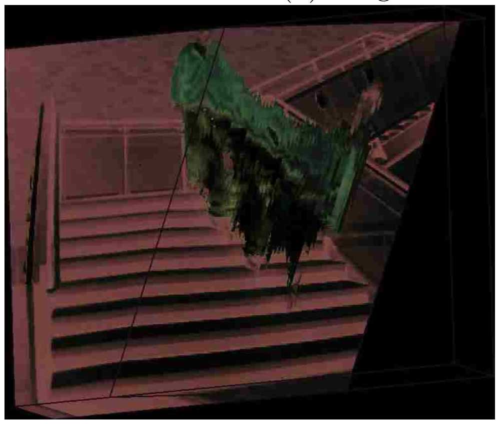
The volume can still be rotated, allowing for cross sections that are not orthogonal to the viewing plane. The result, as with Video Cutout [24], is the ability to minimize interaction by selecting seeds from multiple frames simultaneously. The example in Figure 4.10b shows a cross section in which the user is able to place foreground seeds on the individual's pants in multiple frames with a single stroke. This is particularly helpful for thin, moving objects.

While it may seem that generating a surface by treating the video as a volume would yield unintuitive results, we have found that the video surface is quite helpful for feedback on the current selection. Just as before, the surface not only helps in recognizing discrepancies in the selection (like the attached railing that is evident in Figure 4.10c), but the surface also provides the user with the ability to quickly correct the mistake by interacting with the surface. Here the user quickly clicks on the railing in the video surface to remove it.

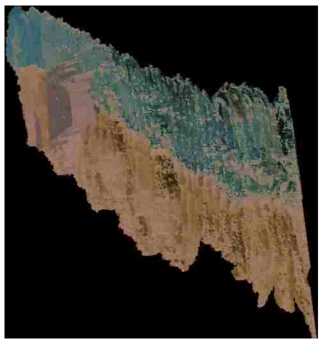
Figure 4.10 shows the extraction of a video surface. The actual frames from the extracted video surface are shown in Figure 4.10a. Figure 4.10b shows the data view and interaction with a nonorthogonal cross section in order to select the individual's pants. Figure 4.10c shows the video surface where the user is able to recognize an unwanted object in the current selection, namely the railing. Figure 4.10d shows the resulting segmentation as presented in the cine-loop window. Although Live Surface currently requires a little more interaction to segment video than is required for conventional image volumes, the sub-second update time still holds, making the process considerably less tedious than other systems.



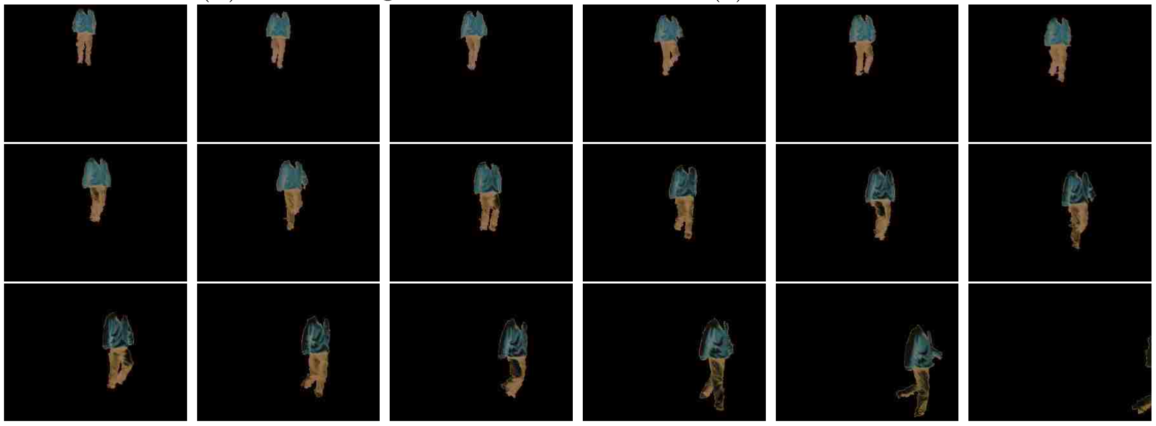
(a) Original frames



(b) Nonorthogonal cross-section



(c) Video Surface



(d) Resulting segmentation

Figure 4.10: Live Surface applied to video. (a) Frames from a video clip of a man walking down stairs. (b) A nonorthogonal cross section for interaction. (c) Interaction with a video surface and placement of seeds. (d) The resulting segmentation.

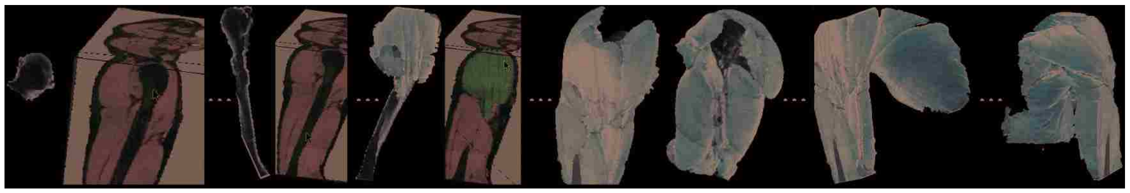


Figure 4.11: Live Surface applied to the Visible Human dataset. The bone's surface is extracted, then the muscle layers are added.

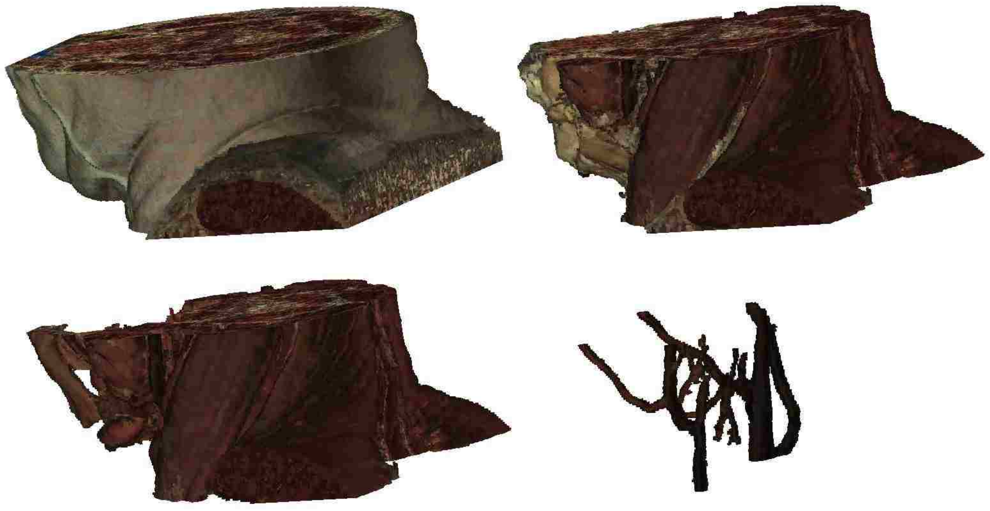


Figure 4.12: Neck from Visible Human project. (top left) Neck with skin. (top right) Neck without skin. (bottom left) Muscle layer. (bottom right) Major blood vessels.

4.4 Application to the Visible Human

Segmentation and rendering results for Live Surface applied to the Visible Human Project are shown in Figures 3.17 and 4.11–4.15. The coloration in the rendering comes from sampling the color in the data itself. For example, the color chosen for the heart (Figure 3.17) is the average color of the regions forming its surface. CGC performs very well, allowing the user to extract surfaces interactively and incrementally, with anatomically relevant surfaces (heart chambers, ascending or descending aorta, carotid arteries, ribs, etc.) added to the cumulative surface with each user interaction.

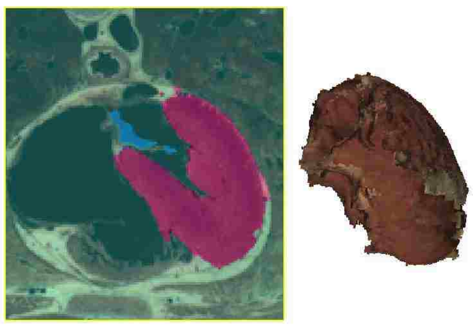


Figure 4.13: Left ventricle from Visible Human Project. (left) Slice from selected volume. (right) Surface of ventricle.

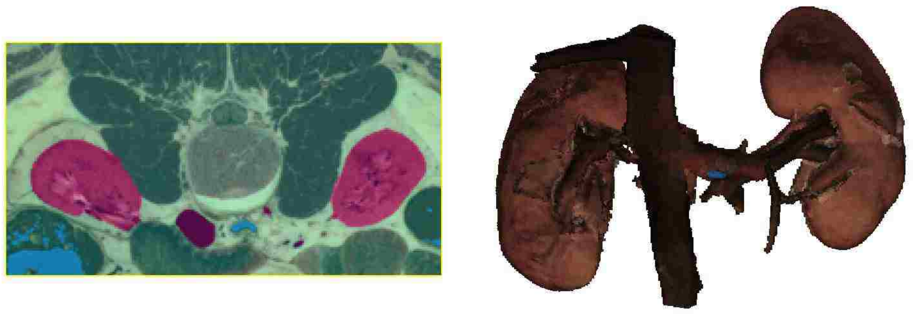


Figure 4.14: Kidneys from the Visible Human Project. (left) Slice from selected volume. (right) Surface of kidneys and major vessels.

The Visible Human data provides a means for exploring human anatomy in full color. Live Surface provides a rich exploratory experience by allowing the user to interactively adjust the selection, adding and removing individual anatomical entities. Figures 4.12-4.14 display various selections from the Visible Human obtained with Live Surface.

Figure 4.15 shows the user's ability to peel back various layers including skin, muscle, and bone to reveal other anatomical structures beneath. In this manner the seeds placed directly on the rendered surface act as an intelligent scalpel peeling away the voxels corresponding to the tissue layer sampled. The entire sequence was produced using a total of 12 user brush strokes, 6 of which are shown. The surface in Figure 4.15a was created with three brush strokes—two for seeding the foreground (skin and muscles of the back and other organs) and one for the background. Each image in the figure shows the seeds placed in a single brush stroke. Figure 4.15b results from the brush stroke in Figure 4.15a, Figure 4.15c results from Figure 4.15b, etc. Three more strokes on the object surface were used between (d) and (e), one for each of the arms and one for the rest of the torso. The entire sequence of images can be generated in about a minute, allowing an interactive exploration of human anatomy.

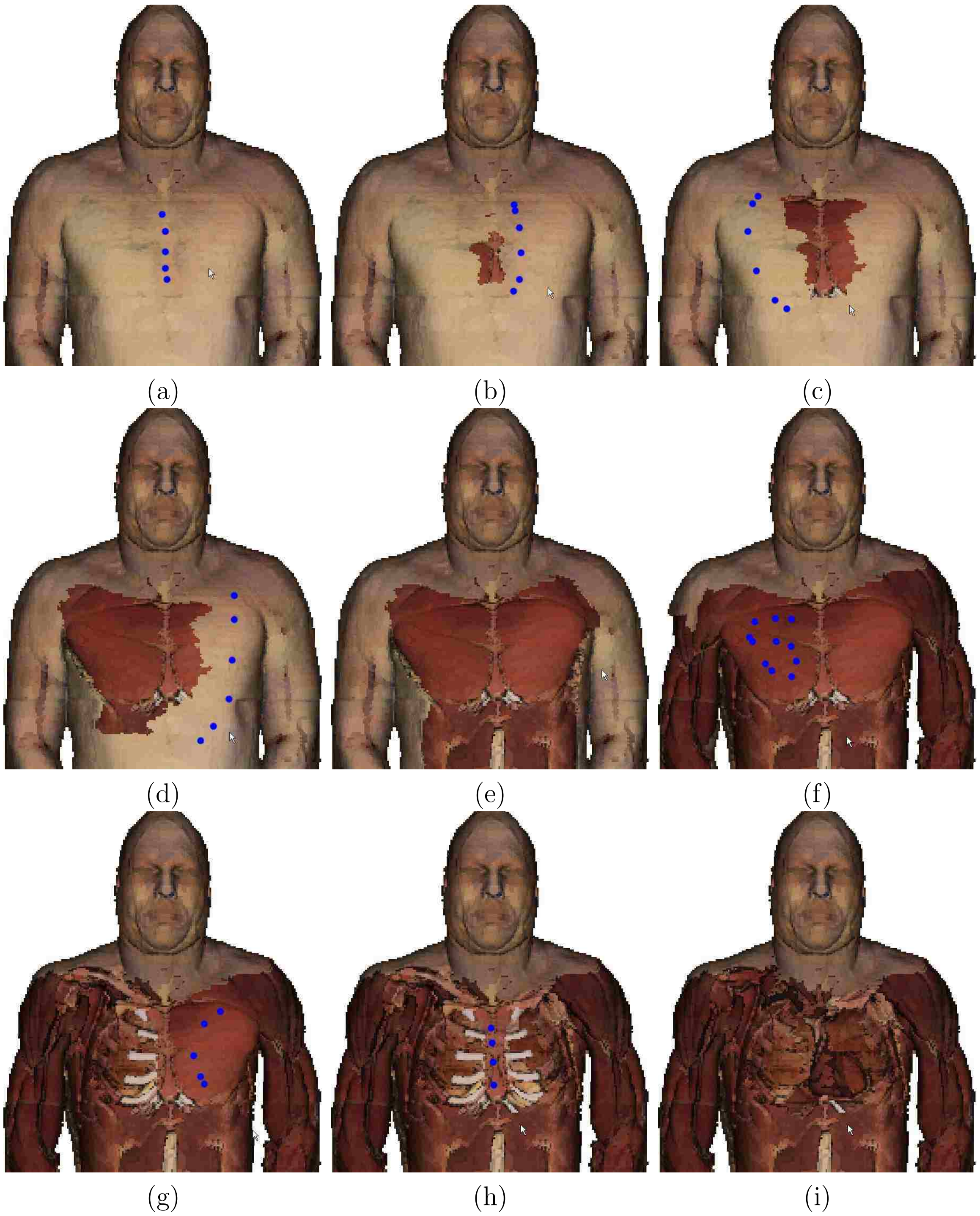


Figure 4.15: Onion skin peeling the Visible Human.

Chapter 5

Conclusions

5.1 Concluding Observations

Live Surface does for 3D volumes what Intelligent Scissors did for 2D images. Intelligent Scissors was not the first technique to formulate optimal boundary detection as a minimum cost path search in a graph, but it brought it into an interactive framework. In the same way, Live Surface provides the user with immediate (~ 0.5 second) feedback about the selected surface. This immediate feedback helps the user decide what to sample next. This is a significant improvement over existing graph-cut techniques, which typically require 5–10 seconds (often much more) to make the cut. (Imagine trying to steer a car remotely with a snapshot of the car's position on the road updated only every 5–10 seconds!) Live Surface allows the user to segment volumes continuously with immediate visual feedback in the refinement of the selected surface. Hence, Live Surface fundamentally changes the paradigm for interactive 3D segmentation and surface extraction.

Live Surface allows objects in 3D image volumes to be extracted and visualized interactively using a cascaded formulation of optimal graph cut. By identifying object surfaces in a preprocessing stage, data structures can be generated that allow for sub-second object segmentation. Optimal graph cut coupled with immediate user feedback makes Live Surface an efficient tool for segmentation, visualization, and exploration of object surfaces in 3D image volumes.

Live Surface performs optimal segmentation at interactive, sub-second speeds. Immediate feedback allows the user to continuously give input until the desired surface is extracted. Segmentation times are an order of magnitude faster than other graph-cut formulations. Surface interaction allows for an improved contextual interface.

5.2 Future Work

While the results generated by Live Surface are quite compelling there are many areas in which further improvements can be made. There are various aspects of the results which can be improved. This section discusses possible strategies for improving accuracy, increasing robustness, shortening the run time, improving results when applied to video, improving usability, and some possible new features.

5.2.1 Improving Accuracy

Accurate segmentation is of extreme importance, especially in medical imaging where it can affect a physician's diagnosis. To date our validation consists mainly of visual inspection and claims based on previous work. One question about Cascading Graph Cut is whether or not it achieve the same results as a graph cut on the actual voxels. Section 4.2.1 has shown that, given seeds for a voxel level graph cut, CGC will yield a result similar but not identical to the voxel level cut. However, a voxel level graph cut requires considerably more seeds than CGC to obtain a comparable segmentation. This is based on some preliminary experiments which have shown that CGC requires far fewer seeds. More extensive comparisons are warranted.

Ultimately, validation of segmentation results must be compared with volumes of known geometry. In medical images such volumes are known as phantoms. In order to perform such validation, measurement tools would be needed so that the segmentation results can be compared to the known measurements from the phantoms.

5.2.2 Increasing Robustness

Although the results shown here are quite compelling, there are many means of improving the robustness of Live Surface. Improving the robustness could eliminate much of the user interaction time, and could make Live Surface applicable to more datasets.

One method for improving the robustness of Live Surface would be to implement some form of dynamic training. If Live Surface can learn what the user is trying to select, it could accelerate the process by automatically selecting similar items. There are many ways in which this could be accomplished.

Most formulations of graph cut make use of a second term or second kind of edge other than that described by Equation 3.1. The second term is a measurement of how each node conforms to some foreground or background model. Dynamic generation of the foreground and background model as done by Boykov [19] and Lazy Snapping [20] may provide greater adaptivity to the current Live Surface formulation.

Another means of training to explore is to propagate seeds outside of the cross section with which the user is interacting. Seeds could be placed in neighboring cross sections based on spatial locality and color similarity.

Many medical scans are not taken with isotropic voxels, meaning that slices are further apart than neighboring voxels within a slice. This being the case it is unreasonable to weight links between neighboring voxels within each slice the same as voxels in neighboring slices. Weights between neighboring basins would need to be adjusted as well, possibly using the distance between centers of mass for each basin. Along with adjusting weights, the rendering should be scaled in accordance with the aspect ratio.

5.2.3 Shortening Run Time

Along with improving robustness, there may be ways to further increase the speed of Live Surface. This would allow for an improved user experience and give the ability to work with larger volumes. One idea for speeding up the algorithm is "cooling" parts of the volume. If a portion of the volume has maintained the same classification for long enough the basins in that vicinity can be eliminated from future cuts. Eliminating basins from the computation shortens the computation time. If the user were to interact near the cooled area it could be reactivated or reheated for new cuts.

Each time CGC cascades down to the next level of the hierarchy it refines the surface. This indicates that CGC and the watershed hierarchy have a different idea of how things should be grouped. As CGC cascades down the hierarchy the parenthood of each basin could be dynamically rearranged so that the same cut made at the top of the hierarchy is made at the bottom. If this is the case, it may not be necessary to descend the entire hierarchy. This would provide for faster results.

The branching factor of the watershed hierarchy is about five. One valid question is whether or not a shallower hierarchy with a higher branching factor would perform faster. Eliminating intermediate levels of the hierarchy should be attempted to determine if CGC would run faster.

5.2.4 Improving Video Results

Since video is significantly different than purely spatial volumes it deserves some special attention. In order to deal with two of the main difficulties in video, large movements between frames and occlusions, adjustments should be considered to Live Surface in both preprocessing and interaction.

One preprocessing step that could improve the interframe connectivity of the graphs used is optical flow. If pixels are linked not with the pixel in the same location in the next frame, but with the pixel that they flow to in the next frame, an object would be more likely to be linked to itself in the next frame. The watershed hierarchy could then take into account this linkage instead of the 6-connected linkage of voxels.

Furthermore, interframe links could be weighted differently than within-frame links. Weights could fall off sharper between frames, or they could be discounted by a constant value. This is the same idea that would be used for medical volumes that are not isotropic.

During interaction there are a few things that could reduce the *amount* of interaction that is required. One drawback to graph cut segmentation is its tendency toward a global solution. That is, once a user has achieved a good segmentation in one frame, their interaction on another frame may throw off the first frame. The ability to lock the segmentation on a specific frame would reduce the amount of interaction required, eliminating the need to return to previously segmented frames to fix them.

A second idea for improving interaction in segmenting video is to add seeds on subsequent and previous frames to the one being edited by the user. Those seeds can be placed using optical flow information, or some other metric based on locality and color similarity. If such automatically placed seeds are placed intelligently, it would reduce the number of seeds that the user would need to place and it would provide better connectivity between frames.

5.2.5 Improving Usability

A third area for improving Live Surface is in the user interface. Since the user interface is critical to an interactive system, user-interface design is crucial. Some of our considered user interface changes deal with new ways to interact with the volume and others simply with the user experience. An aspect of the user interface to improve is the sub-second lag between each interaction. One approach to eliminate this lag is to incorporate multiple threads. One thread would be dedicated to the user interaction, one to CGC, and one to rendering. Furthermore, if the CGC thread was taking too long, the segmentation could be stopped before it reaches the bottom of the hierarchy and the intermediate result could be rendered.

Currently, Live Surface has two main contextual windows. Switching contexts between them slows the interaction. Putting everything in one window by placing the current cross section around the current selected object could eliminate some context switching. Deletion from the current surface tends to be more intuitive interacting with the rendered surface. Addition to the current surface tends to be more intuitive in cross sections of the volume. Thus, combining the two into one window should considerably accelerate interaction by eliminating the context switch.

Another means of interaction would be the idea of intelligent brushes for placing seeds. In the case of a severely fractured skull, for example, it would be quite difficult to pick up every single fragment. Thus if the user could place a single seed and have that seed reflected in every voxel of the same value within a certain radius, with one click the user could select a group of small similar objects. Similarly, seed voxels need not have the same value. Voxels in the vicinity can be seeded with weights according to how similar they are to the selected voxel and how far they are from the selected voxel.

In interacting with the rendered surface there are many tools that would be useful. One example would be a cutting plane. A cutting plane would make every voxel on one side of it a background seed, thus effectively deleting a large portion of the volume and finding a surface on the other side. Such cutting tools could be various geometric shapes, and they could be used to add many foreground or background seeds simultaneously. A sphere or cube of foreground or background seeds could be added all at once. An ellipse could specify that everything outside of its extrusion in the viewing direction is background or everything inside is foreground. A virtual scalpel could intelligently cut through a single layer rather than a fixed depth to simulate an incision used to find the surface underneath.

5.2.6 New Features

Along with the means for improving Live Surface there are a number of useful features to add. One feature would be the inclusion of measurement tools. Once the user has a selection, it is useful to measure the selection. Measurement tools that would be useful are volume measurement, linear measurement as a caliper, measurement of circumference, and more.

Another feature to add is the ability to select any number of distinct objects. There are various ways that this could be accomplished. The binary nature of graph cut could be used to parse a binary tree of component objects. Each time an object is subdivided there would be less data to include in subsequent CGC computations.

As multiple selection are generated, they would be pseudocolored in order to distinguish between them. The color could be user selected, or it could come from successful volume-rendering transfer functions.

The other means for making multiple selections would be to generalize the graph-cut implementation to allow for $n > 2$ classes. This would allow multiple objects to compete for voxels in a complete partition of the volume.

There are other types of data in which to extend Live Surface. None of the core components of Live Surface limit it to 3D data. Thus Live Surface could be applied to n-dimensional data sets as well [30, 31]. Such applications may exceed the sub-second update speeds; however, the speedup should still scale and demonstrate significant improvement over non-hierarchical approaches.

Point-based graphics are another domain that Live Surface may have some application to. Generating a hierarchical graph and cutting it with CGC should be effective in this domain as well.

Appendix A

Cascading Graph Cut Algorithms

Pseudo-code for Cascading Graph Cut and Locally Adaptive Cascading is given here. For detailed descriptions of these algorithms, see Sections 3.4.2 and 3.4.3.

A.1 Global Variables and Data Structures

Globals:

```
    BASIN $fSeed$, $bSeed$                          {generic seed BASINs}
    GRAPH $top$                                     {the top GRAPH in the hierarchy}
```

Data Structures:

```
    GRAPH   {BASIN[] $B$;   EDGE[] $E$;   **Int** $level$;}
    BASIN {
      **Bool** $isFSeed$, $isBSeed$, $isF$, $isB$;
      EDGE[] $E$;   BASIN $parent$;   Basin[] $children$;
      }
    EDGE   {BASIN $from$, $to$;   **Float** $weight$;}
```

A.2 Cascading Graph Cut Algorithm

CascadingGraphCut

Input:

```
    GRAPH $G$
```

Algorithm:

```
1.  **GraphCut**($G$)
2.  **if** $G.level = 0$ **return**
3.  Initialize GRAPH $H$ to empty
4.  $H.level \leftarrow G.level$-1
5.  $H.B \leftarrow$ the *children* of all BASINs in $G$ that border the cut
6.  $H.B \leftarrow$ the *children* of all BASINs in $G$ that are dual seeded
7.  $H.E \leftarrow$ all existing EDGEs *to* and *from* a BASIN in $H$
8.  **for each** EDGE $E$ *from* each BASIN in $H$ **begin**
9.    **if** $E.to$ is **not** in $H$ **begin**
10.     Initialize EDGE $F$ to be a copy of $E$
11.     $F.to \leftarrow$ **ForeOrBack**($E.to$)
12.     $H.E \leftarrow F$
13.   **end**                {this loop generates edges from BASINs
14. **end**                        in $H$ to $fSeed$ and $bSeed$}
15. **CascadingGraphCut**($H$)
```

GraphCut

Input: GRAPH G

Algorithm: run graph cut, set *isF* & *isB* for BASINS in G

ForeOrBack

Input: BASIN B

Output: $fSeed$ **or** $bSeed$

Algorithm:

```
16. **if** $B.isF$ **return** $fSeed$
17. **else if** $B.isB$ **return** $bSeed$
18. **else return ForeOrBack**($B.parent$)
```

A.3 Locally Adaptive Cascading Algorithm

LocallyAdaptiveCascading

Input:

BASIN B

Bool $isFore$

Algorithm:

```
1.  if hasSeed($B$, $isFore$) then return
2.  initialize Int $i \leftarrow 0$
3.  initialize BASIN $C \leftarrow B$
4.  while not hasSeed($C$, $isFore$) begin
5.     setSeed($C$, $isFore$)
6.     if $i = top.level$ then return CascadingGraphCut($top$)
7.     $C \leftarrow C.parent$;    $i \leftarrow i + 1$
8.  end
9.  initialize GRAPH $H$ to empty
10. $H.B \leftarrow C.children$
11. $H.E \leftarrow$ existing EDGEs $from$ and $to$ a BASIN in $G$
12. for each EDGE $E$ $from$ each BASIN in $H$ begin
13.    if $E.to$ is not in $H$ begin
14.       Initialize EDGE $F$ to be a copy of $E$
15.       $F.to \leftarrow$ ForeOrBack($E.to$)
16.       $H.E \leftarrow F$
17.    end                                  {this loop generates edges from BASINs
18. end                                              in $H$ to $fSeed$ and $bSeed$}
19. CascadingGraphCut($H$)
```

hasSeed

Input: BASIN B, **Bool** $isFore$

Output: Bool

Algorithm:

```
20. if $isFore$ then begin
21.     if $B.isFSeed$ then return true;    else return false
22. else
23.     if $B.isBSeed$ then return true;    else return false
```

setSeed

Input: BASIN B, **Bool** $isFore$

Algorithm:

```
24. if $isFore$ then $B.isFSeed \leftarrow$ true
25. else $B.isBSeed \leftarrow$ true
```

Bibliography

[1] E. N. Mortensen and W. A. Barrett, "Intelligent scissors for image composition," *ACM Trans. Graph.*, pp. 191–198, 1995.

[2] J. Edwards, "Live Mesh: an interactive 3D image segmentation tool." Master's thesis, Brigham Young University, 2004.

[3] R. Robb, D. Hanson, R. Karwoski, A. Larson, E. Workman, and M. Stacy, "Analyze: a comprehensive, operator-interactive software package for multidimensional medical image display and analysis." *Comput Med Imaging Graph*, vol. 13, no. 6, pp. 433–54, 1989.

[4] J. Udupa, D. Odhner, S. Samarasekera, R. Goncalves, K. Iyer, K. Venugopal, and S. Furuie, "3DVIEWNIX: An Open, Transportable, Multidimensional, Multimodality, Multiparametric Imaging System," *Proc. Int'l Soc. for Optical Eng.(SPIE) Conf*, vol. 2164, pp. 58–73, 1994.

[5] J. K. Udupa and S. Samarasekera, "Fuzzy connectedness and object definition: Theory, algorithms, and applications in image segmentation." *CVGIP: Graphical Model and Image Processing*, vol. 58, no. 3, pp. 246–261, 1996.

[6] J. K. Udupa, P. K. Saha, and R. de Alencar Lotufo, "Relative fuzzy connectedness and object definition: Theory, algorithms, and applications in image segmentation." *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 11, pp. 1485–1500, 2002.

[7] J. K. Udupa and P. K. Saha, "Fuzzy connectedness and image segmentation." *Proceedings of the IEEE*, vol. 91, no. 10, pp. 1649–1669, 2003.

[8] K. Höhne, "VOXEL-MAN, Part 1: Brain and Skull," pp. 3–540.

[9] "GE healthcare - image gallery - volume rendering," http://www.gehealthcare.com/usen/ct/clin_app/products/volrendering.html, October 2006.

[10] R. A. Drebin, L. Carpenter, and P. Hanrahan, "Volume rendering," in *SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques.* New York, NY, USA: ACM Press, 1988, pp. 65–74.

[11] M. Levoy, "Efficient ray tracing of volume data," *ACM Transactions on Graphics (TOG)*, vol. 9, no. 3, pp. 245–261, 1990.

[12] R. Kikinis, "Personal Communication," 2006.

[13] C. Sonka, M. Hlavac and R. Boyle, *Image Processing, Analysis, and Machine Vision*, 2nd ed. Brooks/Cole Publishing Company, 1999.

[14] J. Fairfield, "Toboggan contrast enhancement for contrast segmentation," in *Proceedings of the 10th International Conference on Pattern Recognition (ICPR '90)*, June 1990.

[15] X. Yao and Y. P. Hung, "Fast image segmentation by sliding in the derivative terrain," *Proceedings of SPIE Intelligent Robots and Computer Vision 1991*, vol. 1607, 1991.

[16] L. Vincent and P. Soille, "Watersheds in digital spaces: An efficient algorithm based on immersion simulations," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 13, no. 6, pp. 583–598, 1991.

[17] L. J. Reese and W. A. Barrett, "Image editing with intelligent paint," in *Proceedings of Eurographics 2002*, vol. 21, 2002, pp. 714–724.

[18] Y. Boykov and V. Kolmogorov, "An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision," in *Energy Minimization Methods in Computer Vision and Pattern Recognition*, 2001, pp. 359–374.

[19] Y. Boykov and M.-P. Jolly, "Interactive graph cuts for optimal boundary and region segmentation of objects in n-d images." in *ICCV*, 2001, pp. 105–112.

[20] Y. Li, J. Sun, C.-K. Tang, and H.-Y. Shum, "Lazy snapping," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 303–308, 2004.

[21] K. Li, X. Wu, D. Z. Chen, and M. Sonka, "Optimal surface segmentation in volumetric images-a graph-theoretic approach," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 1, pp. 119–134, 2006.

[22] X. Yuan, N. Zhang, M. X. Nguyen, and B. Chen, "Volume cutout," *The Visual Computer (Special Issue of Pacific Graphics 2005)*, vol. 21, no. 8–10, pp. 745–754, 2005.

[23] Y. Li, J. Sun, and H.-Y. Shum, "Video object cut and paste," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 595–600, 2005.

[24] J. Wang, P. Bhat, R. A. Colburn, M. Agrawala, and M. F. Cohen, "Interactive video cutout," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 585–594, 2005.

[25] H. Lombaert, Y. Sun, L. Grady, and C. Xu, "A multilevel banded graph cuts method for fast image segmentation," in *ICCV '05: Proceedings of the Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1.* Washington, DC, USA: IEEE Computer Society, 2005, pp. 259–265.

[26] L. Reese, "Intelligent paint: Region-based interactive image segmentation," Master's thesis, Brigham Yound University, 1999.

[27] T. Kay and J. Kajiya, "Ray tracing complex scenes," *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pp. 269–278, 1986.

[28] J. Amanatides and A. Woo, "A fast voxel traversal algorithm for ray tracing," *Eurographics*, vol. 87, no. 3, p. 9, 1987.

[29] D. Belnap, B. McDermott, D. Filman, N. Cheng, B. Trus, H. Zuccola, V. Racaniello, J. Hogle, and A. Steven, "Three-dimensional structure of poliovirus receptor bound to poliovirus," *Proc. Natl. Acad. Sci. USA*, vol. 97, no. 3, pp. 73–78, 2000.

[30] W. A. Barrett, "Dynamic three-dimensional shaded surface display of the rotating, beating left ventricle, atrium and aorta from cine CT," in *IEEE Proceedings of Computers in Cardiology.* Washington, DC, USA: IEEE Computer Society, 1986, pp. 491–494.

[31] S. Siegler, J. Udupa, S. Ringleb, C. Imhauser, B. Hirsch, D. Odhner, P. Saha, E. Okereke, and N. Roach, "Mechanics of the ankle and subtalar joints revealed through a 3D quasi-static stress MRI technique." *J Biomech*, vol. 38, no. 3, pp. 567–78, 2005.