

**Examensarbete**  
LITH-ITN-MT-EX--06/030--SE

# **Local Level Set Segmentation with Topological Structures**

**Gunnar Johansson**

2006-05-30



**Linköpings universitet**  
**TEKNISKA HÖGSKOLAN**

LITH-ITN-MT-EX--06/030--SE

# **Local Level Set Segmentation with Topological Structures**

Examensarbete utfört i mediteknik  
vid Linköpings Tekniska Högskola, Campus  
Norrköping

**Gunnar Johansson**

Handledare Ken Museth

Examinator Ken Museth

Norrköping 2006-05-30

**Avdelning, Institution**

Division, Department

Institutionen för teknik och naturvetenskap

Department of Science and Technology

**Datum**

Date

**2006-05-30****Språk**

Language

- Svenska/Swedish  
 Engelska/English

 \_\_\_\_\_**Rapporttyp**

Report category

- Examensarbete  
 B-uppsats  
 C-uppsats  
 D-uppsats

 \_\_\_\_\_**ISBN****ISRN LITH-ITN-MT-EX--06/030--SE****Serietitel och serienummer**

Title of series, numbering

**ISSN****URL för elektronisk version****Titel**

Title

Local Level Set Segmentation with Topological Structures

**Författare**

Author

Gunnar Johansson

**Sammanfattning**

Abstract

Locating and segmenting objects such as bones or internal organs is a common problem in medical imaging. Existing segmentation methods are often cumbersome to use for medical staff, since they require a close initial guess and a range of different parameters to be set appropriately. For this work, we present a two-stage segmentation framework which relies on an initial isosurface interactively extracted by topological analysis. The initial isosurface seldom provides a correct segmentation, so we refine the surface using an iterative level set method to better match the desired object boundary. We present applications and improvements to both the flexible isosurface interface and level set segmentation without edges.

**Nyckelord**

Keyword

Level set methods, isosurface extraction, flexible isosurfaces, contour tree, topology, segmentation

## Upphovsrätt

Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under en längre tid från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida <http://www.ep.liu.se/>

## Copyright

The publishers will keep this document online on the Internet - or its possible replacement - for a considerable time from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: <http://www.ep.liu.se/>

# Local Level Set Segmentation with Topological Structures

by

**Gunnar Johansson**



A Thesis submitted in partial fulfilment  
of the requirements for the degree of

**Master of Media Technology**

**Linköping University**

**2006**

---

# Abstract

---

Locating and segmenting objects such as bones or internal organs is a common problem in medical imaging. Existing segmentation methods are often cumbersome to use for medical staff, since they require a close initial guess and a range of different parameters to be set appropriately. For this work, we present a two-stage segmentation framework which relies on an initial isosurface interactively extracted by topological analysis. The initial isosurface seldom provides a correct segmentation, so we refine the surface using an iterative level set method to better match the desired object boundary. We present applications and improvements to both the flexible isosurface interface and level set segmentation without edges.

---

# Acknowledgements

---

I would like to thank my supervisor Ken Museth and our collaborator Hamish Carr for the many great ideas and valuable feedback for putting this thesis together. Also, thanks to Michael Bang Nielsen for providing the DT-Grid implementation.

Finally, much love to Sara for supporting me, emotionally and financially.

---

# Contents

---

<b>Abstract</b>	<b>i</b>
<b>Acknowledgements</b>	<b>ii</b>
<b>Contents</b>	<b>iii</b>
<b>List of Figures</b>	<b>v</b>
<b>List of Algorithms</b>	<b>vii</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Background</b>	<b>2</b>
2.1 Preliminaries . . . . .	2
2.2 Medical Imaging and Segmentation . . . . .	2
2.3 The Contour Tree and Flexible Isosurfaces . . . . .	4
2.4 Level Set Methods . . . . .	7
<b>3 Previous Work</b>	<b>8</b>
3.1 Isosurface Extraction . . . . .	8
3.1.1 Marching Cubes . . . . .	10
3.1.2 Continuation . . . . .	10
3.2 The Contour Tree . . . . .	11
3.2.1 Computation . . . . .	11
3.2.2 Path Seeds and Piecewise Continuation . . . . .	12
3.2.3 Local Geometric Measures and Simplification . . . . .	15
3.2.4 Extending to any Mesh and Interpolant . . . . .	15
3.3 Level Set Methods . . . . .	16
3.3.1 Temporal Discretization . . . . .	17



---

3.3.2	Spatial Discretization . . . . .	18
3.3.3	The Signed Distance Property and Reinitialization . . . . .	20
3.3.4	Narrow Band Schemes . . . . .	21
3.4	Level Set Segmentation with Edges . . . . .	23
3.5	Level Set Segmentation without Edges . . . . .	24
<b>4</b>	<b>Combining Topological Structures with Local Segmentation</b>	<b>27</b>
4.1	Motivation . . . . .	27
4.2	Topological Computation on Discrete Data . . . . .	27
4.3	From Contours to Initial Level Set . . . . .	28
4.4	Localizing Segmentation without Edges . . . . .	31
<b>5</b>	<b>Implementation and Results</b>	<b>36</b>
5.1	Implementation . . . . .	36
5.2	Results . . . . .	36
<b>6</b>	<b>Conclusions and Future Work</b>	<b>40</b>
6.1	Conclusions . . . . .	40
6.2	Future Work . . . . .	40
	<b>Bibliography</b>	<b>42</b>

---

# List of Figures

---

2.1	Cut planes, isosurface visualization and direct volume rendering . . . . .	3
2.2	Heightfield of Vancouver, Canada . . . . .	5
2.3	The landscape and topographic map of Vancouver . . . . .	5
2.4	Critical points and the corresponding contour tree . . . . .	6
2.5	The flexible isosurface interface [Car04] . . . . .	6
2.6	A simple level set function . . . . .	7
3.1	Barycentric interpolation function . . . . .	8
3.2	Possible cases for tetrahedra with barycentric interpolation . . . . .	9
3.3	Bilinear interpolation function with isolines . . . . .	9
3.4	The Marching Cubes Cases . . . . .	10
3.5	The Marching Squares Cases . . . . .	11
3.6	Computation of the Join Tree . . . . .	13
3.7	Join Tree and Split Trees . . . . .	13
3.8	Computation of the Contour Tree . . . . .	14
3.9	Contour Tree for 3D dataset [CS03] . . . . .	14
3.10	A noisy contour tree before and after simplification [Car04] . . . . .	16
3.11	Narrow band displaying the interface, the $\beta$ tube and the $\gamma$ tube by Peng <i>et al.</i> [PMO <sup>+</sup> 99] . . . . .	22
3.12	Segmentation with edges using Canny edge detector . . . . .	24
3.13	Motivation for the functional in [CV01] using a simple example . . . . .	25
3.14	Images showing the initial curve and the segmented result by “segmentation with- out edges” . . . . .	26
4.1	6–18 Connectivity . . . . .	28
4.2	Union operation for two contours intersecting the same cell . . . . .	29
4.3	The inside/outside ambiguity . . . . .	31

---

4.4	Different solutions depending on the embedding space . . . . .	33
4.5	Segmentation without edges using narrow band level set propagation . . . . .	34
4.6	Segmentation without edges using local computations . . . . .	35
5.1	Interactive Brain Segmentation . . . . .	38
5.2	Kidney Segmentation . . . . .	39

---

# List of Algorithms

---

1	INITIALIZELEVELSETBOUNDARY() . . . . .	30
2	FLOODFILL() . . . . .	32
3	CONVERTFROMCONTOURSTOLEVELSET() . . . . .	33

---

## Chapter 1

# Introduction

---

A common task in medical imaging is locating and segmenting objects of interest such as the brain or other internal organs. This is often complicated since medical datasets are noisy and hard to study using direct techniques such as isosurface visualization and direct volume rendering. Moreover, these techniques rely on a single isovalue to represent the object of interest which seldom gives a close match to the real boundary.

In contrast, level set segmentation techniques rely on other properties of the data such as gradients or image variations to represent a particular object. Level set methods describe an initial value problem which are evolved over time using a specific partial differential equation. Thus, they tend to give local solutions which are highly dependent upon the initial surface.

In this work, we combine isosurface techniques with level set segmentation in a pipelined manner to solve the problem of both approaches. Isosurfaces seldom represent the real object but they provide the user with an efficient way of selecting and generating a close initial surface to use as input to the level set method. In this thesis we have used a segmentation method based on global image variations as described by Chan and Vese [CV01].

Chapter 2 will start by introducing the basic concepts of medical imaging, flexible isosurfaces and level set methods as the core components of this thesis. Chapter 3 will expand on these by presenting the previous work in each area.

Thereafter, Chapter 4 will present the main contributions of this thesis. This will explain the topological issues of converting an isosurface to the implicit representation of a level set followed by a description of how we localize and restrict the global method of Chan and Vese.

Finally, Chapter 5 and Chapter 6 will present the implementation and results and describe future work in this field.

---

## Chapter 2

# Background

---

This chapter will introduce and provide the background for the core components of this work. We will start by presenting preliminaries important for the understanding of the thesis. Then we will describe the concepts of medical imaging and the key techniques used in such an application. This is followed by introducing the *contour tree* which is a datastructure representing the topology of a scalar dataset. The contour tree defines the base of a technique called *flexible isosurfaces* used for exploratory visualization. Last we briefly describe *level set methods* which is a robust and flexible way to represent and interact with surfaces in 3D.

### 2.1 Preliminaries

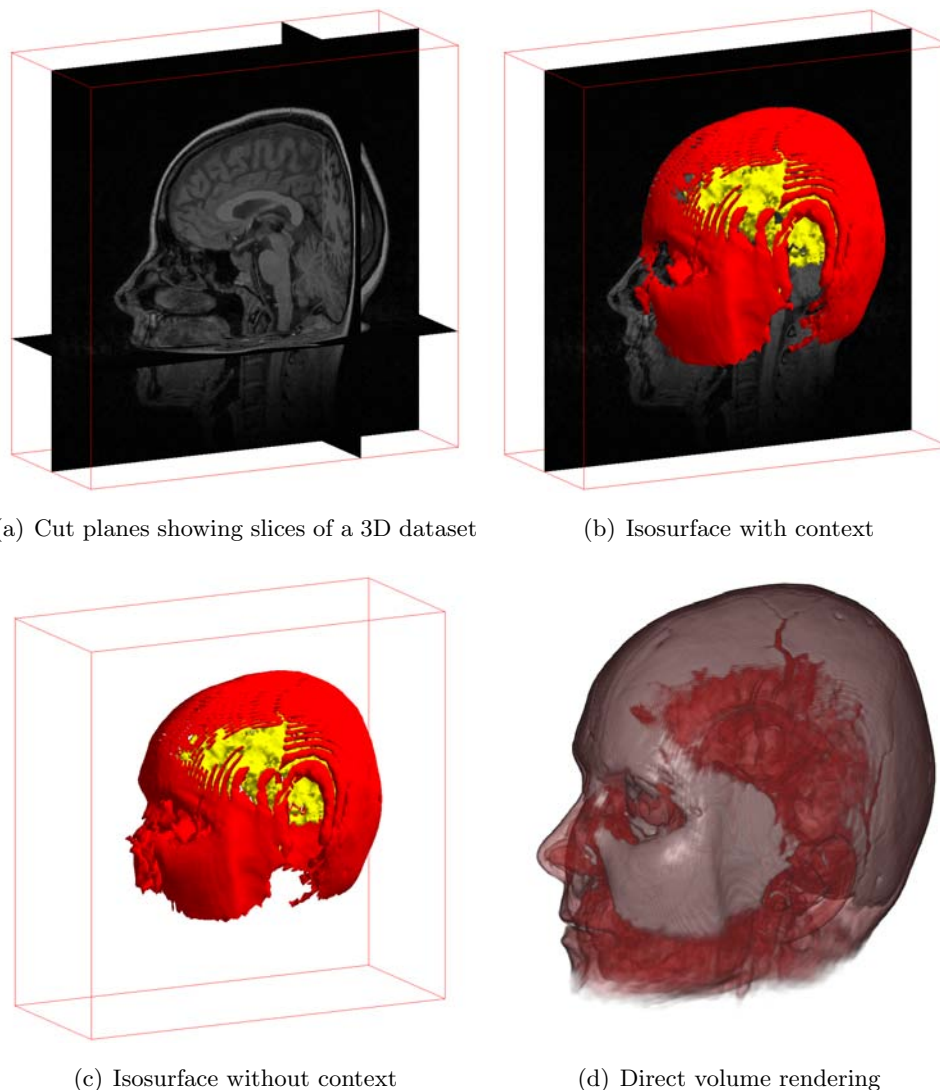
In this section we will describe some fundamental terms used in visualization. An important concept is the relation between *sampling* and *reconstruction*. Most visualization techniques assume that the property being visualized is a continuous function and defined everywhere in the domain. However, most of the time we are dealing with *sampled datasets*, where the property is defined only at *discrete sample points*. Thus, we need to *reconstruct* the property in-between sample points using an interpolating function.

How the sample points are organized defines our *grid*. Usually the points are regularly sampled with a uniform spacing giving us a *regular grid*. The function is then reconstructed using a *mesh* which is subdivided into *cells* defining how the points in the grid, or equivalently *vertices* in the mesh, are interconnected. Commonly used cell types include *simplicial* (triangular in 2D, tetrahedral in 3D) and *square* or *cubic*. An interpolating function uses the vertices of the cell to interpolate the value of any point inside the cell. Simplicial meshes often use *barycentric* interpolation while square and cubic meshes often use *bilinear* or *trilinear* interpolants.

If not stated otherwise, we assume that medical datasets are sampled in a regular grid. We will further discuss the implications of different cell types and interpolants in Chapter 3.

### 2.2 Medical Imaging and Segmentation

Medical imaging is a rapidly advancing field with the important purpose of assisting medical staff in everyday tasks such as diagnostics and surgery planning. Data from MRI and CT



(a) Cut planes showing slices of a 3D dataset

(b) Isosurface with context

(c) Isosurface without context

(d) Direct volume rendering

**Figure 2.1:** Cut planes, isosurface visualization and direct volume rendering. In 2.1(b) the isosurface is displayed together with a slice to give context. This isosurface consists of *two connected components*, the brain in yellow and parts of the skull in red.

scans are increasing in size and efficient methods are needed to present the information in a useful and correct way. The fundamental task of a medical imaging application, or any scientific visualization application, is to hide redundant data, only showing that of interest to the user. In other words, the visualization problem aims to *reduce the dimensionality* of the data such that the important information is easily perceivable.

For this work we assume that medical datasets are described by a scalar function  $f : IR^3 \rightarrow IR$ . The task of the visualization is to reduce the information and provide a two dimensional view projected onto the user's screen. A simple approach is to use cut planes in the 3D dataset as shown in Figure 2.1(a). This may work for simple targets but can not be considered a volumetric technique since it does not provide a complete 3D view of the data. The two major techniques for volume visualization are *isosurface visualization* and *direct volume rendering*.

Isosurface visualization relies on a specific *isovalue* defining the objects of interest. For medical

datasets this isovalue can be equivalent to the density of the tissue or other physical properties. An isosurface for an isovalue  $h$  is the level set  $f^{-1}(h) = \{\mathbf{x} \in \mathbb{R}^3 : f(\mathbf{x}) = h\}$  of a scalar function  $f$ . We use the term *contour* to define a single connected component of an isosurface. Figure 2.1(c) shows an isosurface with two contours - the brain in yellow and parts of the skull in red.

Direct volume rendering is a technique which maps values in the data, or derivatives thereof, to a specific color and opacity by means of *transfer functions* (Figure 2.1(d)<sup>1</sup>). In contrast to isosurfaces, volume rendering tend to be better at displaying internal structures and provide a more “volumetric feel”.

Both isosurfaces and volume rendering rely on the assumption that any object of interest can be described by a specific value or a range of values. This assumption is rarely true however, due to noise or non uniform measuring of the data. For example, the attenuation of a ray in computed tomography (CT) varies. As a result, the measured density of a uniform-density object is seldom perfectly uniform, and the boundary can not be well represented by an isovalue. Therefore, segmentation methods are often used to provide a better match to the boundaries of an object. In this thesis we will only describe segmentation based on level set methods. These methods must be provided with an initial surface which is then attracted to the boundaries of interest. There are two principal methods with different definitions of a boundary - segmentation *with* edges and segmentation *without* edges.

Segmentation with edges assumes that a boundary can be defined by gradient, which is true for objects with well defined edges. Provided that this requirement is satisfied, these methods have been proven quite successful. However, since the gradient is a local feature these methods tend to converge to local solutions, requiring an initial surface very close to the actual boundary. Moreover, if the edges are not well defined the segmentation easily passes through the boundary. In contrast, segmentation without edges defines a boundary by solving a partition problem which aims to minimize the *variation* inside each partition. This approach has the benefit of finding objects with very soft edges and is not sensitive to the initial surface. We will describe these approaches further in Section 3.4 and Section 3.5.

## 2.3 The Contour Tree and Flexible Isosurfaces

The contour tree is a topological representation of the contours of a dataset. It gives an understanding of how different contours merge or split and can be used to gain a more clear view of the data. In this section I will illustrate this idea by providing an example in 2D. The computation of the tree and further applications will be discussed in Section 3.2.

The contour tree was first introduced in the context of topographic maps by Boyell and Ruston [BR63]. A landscape can be represented as a *heightfield* which maps a point in two dimensional space to a height value. Throughout this thesis we will use the heightfield of Vancouver, Canada, as shown in Figure 2.2. The landscape itself is shown in Figure 2.3(a).

<sup>1</sup>Direct volume rendering image produced by the courtesy of Ulf Lindgren and Olof Rehnström



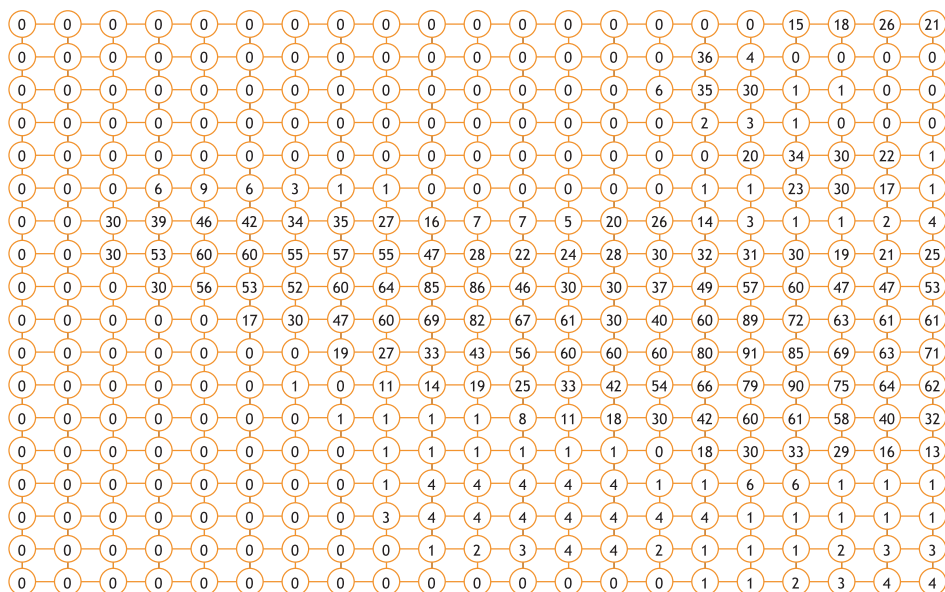
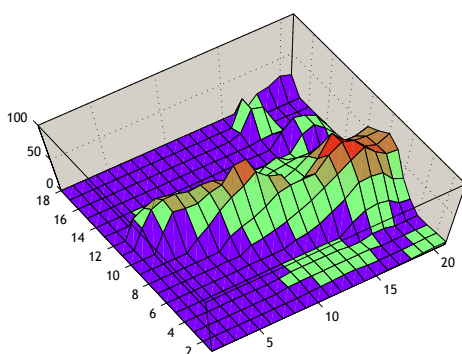
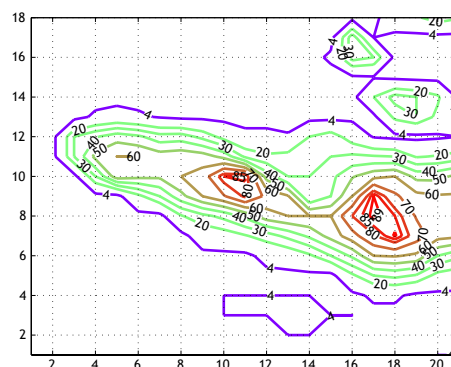


Figure 2.2: Heightfield of Vancouver, Canada



(a) The landscape of Vancouver

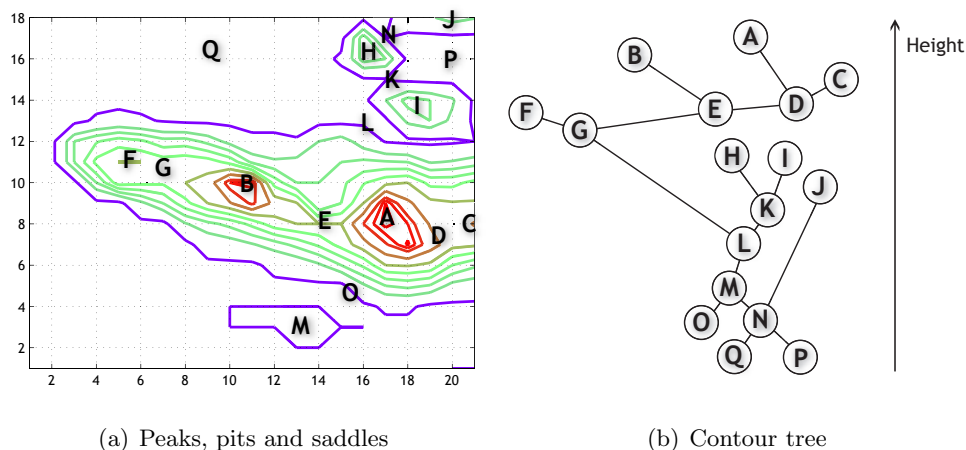


(b) Topographic map of Vancouver

Figure 2.3: Heightfield

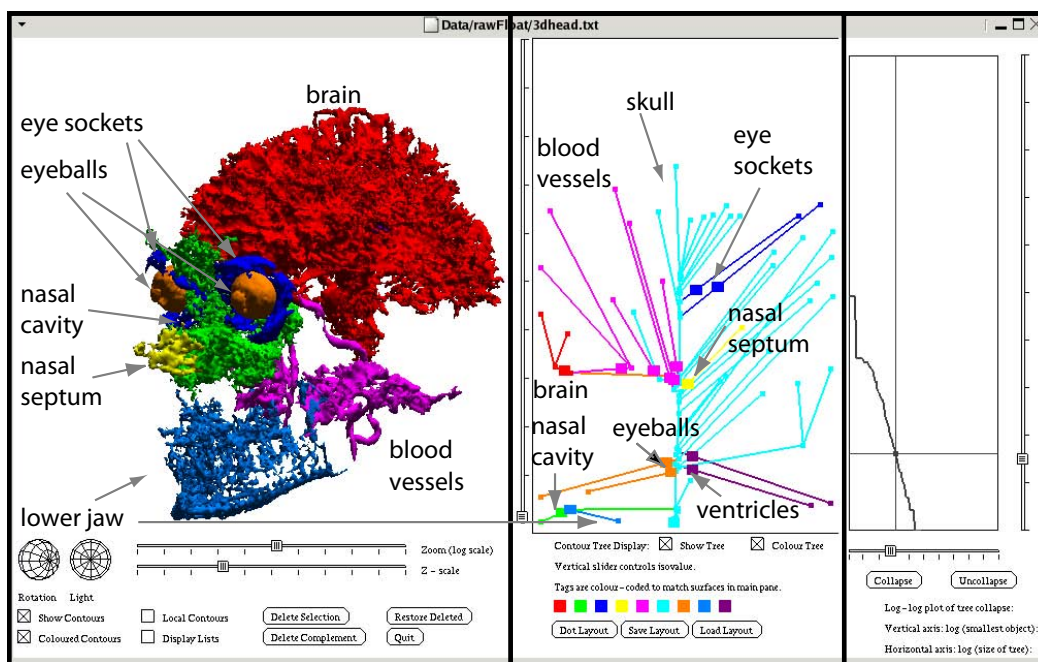
Topographic maps use *isolines* to display height differences in a landscape. Isolines is the 2D counterpart of isosurfaces, where the isovalue corresponds to height in our case. As for isosurfaces, we use *contour* to define a single connected component of an isoline. Thus, the term contour can be used in any dimension - whether a contour is a line or a surface is derived from the context. Figure 2.3(b) shows the contours of the heightfield as usually displayed in a topographic map. The height, or isovalue, is noted along each contour.

The contour tree gives us information about the peaks, pits and saddles (*critical points*) in a landscape, and tells us how they are interconnected. For example, the points *A* and *B* of Figure 2.4(a) are the highest peaks of Vancouver. They are represented as the top nodes in the contour tree as shown in Figure 2.4(b). The third highest peak is *C*, at the right boundary of the map. As two persons walk downhill from *A* and *C*, aiming for the first saddle, they would meet at *D* where the *contours of A and C merge*. The two lowest points in the landscape are *P* and *Q* which naturally define the sea level.



**Figure 2.4:** Critical points and the corresponding contour tree

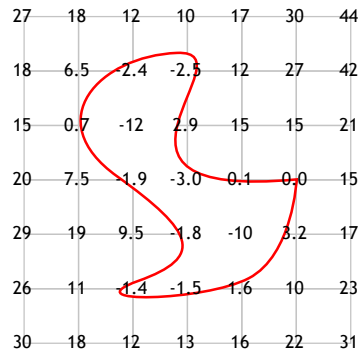
By definition, it is impossible for contours to intersect. They are very much like the layers of an onion, nested in a hierarchical manner. That is why the contour tree is indeed a *tree* instead of a more general *graph*. Visualizing and exploring contours in 2D is in general simple, but it is non trivial in 3D. The nested contours have a tendency to occlude smaller features which is a big problem in isosurface visualization. Consider Figure 2.1(c) where the skull occludes most of the brain. The contour tree provides a tool for solving this problem as demonstrated by Carr and Snoeyink [CS03]. Using *flexible isosurfaces*, the user can select and evolve individual contours in the contour tree, isolating features which are very hard to find using other techniques. An example is shown in Figure 2.5.



**Figure 2.5:** The flexible isosurface interface [Car04]

## 2.4 Level Set Methods

The level set method was originally introduced by Sethian and Osher [OS88] in the context of modeling propagating fronts described by a number of physical phenomenon. A front, or interface, is implicitly represented by embedding it in a space one dimension higher than that of the boundary. We say that the interface has *codimension one*. For example, a one dimensional curve is embedded in a two dimensional space, a two dimensional surface is embedded in a three dimensional space, and so on. In general, an  $n - 1$  dimensional interface is embedded in an  $n$  dimensional space. This space defines the *level set function* where every point evaluates to the *closest distance to the boundary*. Moreover, the sign of each point determines whether the point is *inside* or *outside* the boundary. For this thesis we use a *negative inside, positive outside* sign convention. The boundary itself is defined as the *zero level set* of the function, equivalent to the isoline (in 2D) or isosurface (in 3D) with isovalue zero. A simple example is shown in Figure 2.6.



**Figure 2.6:** A simple level set function. A one dimensional curve embedded in a two dimensional space.

In order to modify the boundary, we introduce an artificial time dependence to the level set function. We can then define a set of partial differential equations to evolve the function, and thus the boundary, over time. This approach has the great benefit of allowing any *topological changes* - the boundary can split, disappear and merge in any way. This is complicated using other representations such as meshes and parameterized surfaces, and the main reason for its success. The level set method will be further described in Section 3.3.

This chapter has introduced the core concepts of this thesis. We presented sampling and reconstruction which strongly direct the numerical methods we apply on the data. Then we described medical imaging and the basic techniques for visualizing volumetric datasets. We noted the problem of representing and segmenting an object using a particular isovalue and presented segmentation which relies on other properties of the data rather than isovalue alone. However, segmentation methods often need close initial surfaces to match the object to be segmented. The flexible isosurfaces based on the contour tree provides the user with an interface to generate such an initial surface or boundary. Last we introduced the level set method which provide a powerful representation of the boundaries we will use for the segmentation. We will study each of these parts in more detail in the next chapter.

---

## Chapter 3

# Previous Work

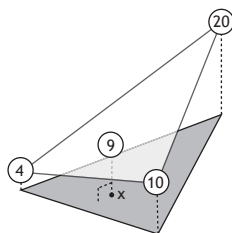
---

This chapter will provide a more detailed description of the concepts introduced in Chapter 2. We will start by describing algorithms for extracting isosurfaces, with *marching cubes* in particular. We will then study the computation of the contour tree, providing more insight in the topological analysis. Finally we will describe the mathematical foundation of level set methods and segmentation methods derived from this methodology.

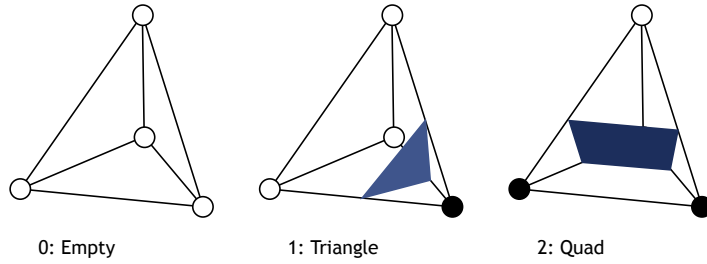
### 3.1 Isosurface Extraction

We gave a brief introduction to isosurface visualization in Section 2.2. This section will further describe the extraction of isosurfaces with specific applications to the contour tree. An isosurface for an isovalue  $h$  is the set  $f^{-1}(h) = \{\mathbf{x} \in \mathbb{R}^3 : f(\mathbf{x}) = h\}$ . Thus, we require the function  $f(\mathbf{x})$  to be a continuous function. However, as noted in Section 2.1, we usually deal with sampled functions which are defined only at discrete sample points. To reconstruct the function value in-between the sample points, we choose a cell type and an interpolant.

An example of such is the simplicial cell combined with barycentric interpolation. The simplicial cell is a triangle in 2D and a tetrahedra in 3D. These cells are mathematically convenient since barycentric interpolation is *monotonic*, so that any critical point is required to be at the vertices. This means that the function is bounded by the values at the vertices. For example, consider a triangular cell with vertex values 4, 10 and 20 as depicted in Figure 3.1. The monotonic function defines a plane which gives the value at any point  $x$  by the height of the plane. An isoline is the cross-section of the plane at a given height giving us a straight line passing through the cell. This extends also to the tetrahedral cell in 3D where any isosurface is a cross-section of a hyper-plane (volume) giving us a plane passing through the cell.



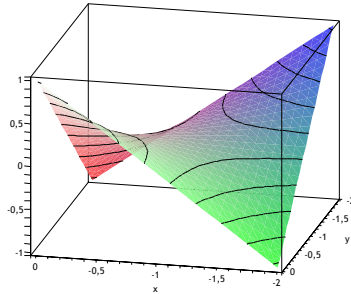
**Figure 3.1:** Barycentric interpolation function



**Figure 3.2:** Possible cases for tetrahedra with barycentric interpolation

To analyze the different ways in which a cell can be intersected by an isosurface we classify each vertex as either *above* (black) or *below* (white) a certain isovalue  $h$ . Reducing the equivalent cases by rotation and value symmetry, we get three possible isosurface cases for the tetrahedral cell as shown in Figure 3.2. In practice, extracting the correct isosurface for a tetrahedra is a two step process. First we classify the vertices as above or below to determine the isosurface case. Then we simply interpolate the corners of the isosurface along *each intersected edge*.

However, there are reasons for not choosing this simple cell type. Most scientific datasets are organized in a cubical grid, which can indeed be subdivided to a simplicial mesh. However, as shown in [CMS06], this introduces visible artifacts and may increase storage and computation since some subdivision schemes require the insertion of additional vertices. To avoid these problems, a more complex cell type and interpolant must be used.

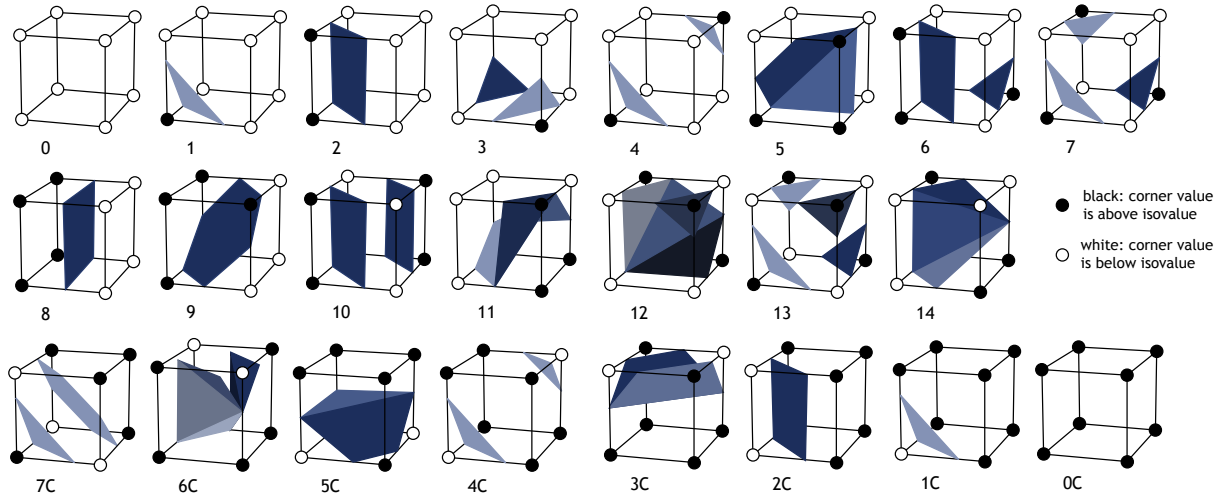


**Figure 3.3:** Bilinear interpolation function with isolines

In 2D, a natural choice is the square cell with bilinear interpolation. The bilinear interpolation function can be written on the form  $F(x, y) = axy + bx + cy + d$ , where  $(x, y)$  is a point in the cell and  $a, b, c$  and  $d$  depends on the values at the vertices. The gradient of  $F(x, y)$  is:

$$\nabla F = \left( \frac{\partial F}{\partial x}, \frac{\partial F}{\partial y} \right) = (ay + b, ax + c) \quad (3.1)$$

Since the partial derivatives of  $F$  are linear, we cannot have any local maximum or minimum in the cell. However, we have a saddle point where  $\nabla F = 0$ , which evaluates to  $x = -c/a$  and  $y = -b/a$ . For an intuitive understanding, a bilinear function with  $a = b = c = d = 1$  is depicted in Figure 3.3. This figure also displays isolines at regular intervals. Thus, in order to correctly identify all critical points in such a mesh, we need to consider possible saddle points inside each cell. Moreover, the isolines are not straight lines as compared to the isolines for a simplicial mesh. This inconvenience is even more evident when studying the cubical cell with a



**Figure 3.4:** The Marching Cubes Cases

trilinear interpolation function, given by  $F(x, y, z) = axyz + bxy + cxz + dyz + ex + gy + hz + k$ . Following the analysis in [PCM03], we can show that this cell has at most two interior saddle points, further complicating the extraction of correct isosurfaces.

### 3.1.1 Marching Cubes

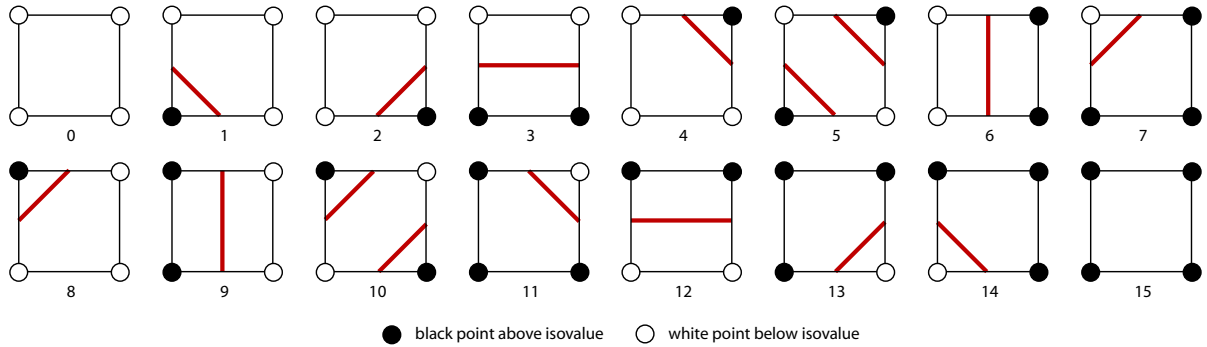
As a way of simplifying the extraction of isosurfaces using cubical cells, the *marching cubes algorithm* was proposed by Lorensen and Cline [LC87]. Instead of using the full trilinear interpolant, they interpolate *linearly along each edge of the cube*. As for the simplicial cell, we classify each vertex as being either above or below a specific isovalue. The binary classification of the eight vertices gives us  $2^8 = 256$  possible ways in which an isosurface can intersect a cube. Due to symmetry, these cases were reduced to 15 for the original algorithm. However, this algorithm was later found to generate holes in the surface under certain conditions [Dur88]. A number of solutions were presented to this problem, the simplest of which, proposed by Montani *et al.* [MSS94], added complementary cases to the original 15 basic cases (Figure 3.4) to resolve the problem.

This idea can also be applied in 2D for the square cell, resulting in *marching squares*. For later discussion in this thesis, the complete set of marching squares cases are shown in Figure 3.5.

### 3.1.2 Continuation

The marching cubes algorithm proceeds by iteratively visiting every cell in the mesh, classifying the vertices and interpolating the isosurface for the corresponding case. There are two problems with this basic approach. The first problem is efficiency - an isosurface is expected to intersect much fewer cells than the total number of cells in the mesh, so the algorithm spends much time in empty cells. This problem can be minimized by applying accelerating datastructures such as the kd-tree [Ben75, LSJ96] or the interval tree [Ede80, CMM<sup>+</sup>97]. The second problem lies





**Figure 3.5:** The Marching Squares Cases

in the *order* in which the cells are processed. By traversing the cells in an arbitrary order, the notion of a *connected* surface is lost.

Building on the same idea as marching cubes, the *continuation method* [WMW86] by Wyvill *et al.* solves both problems stated above. Instead of visiting the cells in any order, it starts at a cell which is known to be intersected by the surface and follows the surface to the neighbouring cells. Since only cells intersected by the surface are visited, the algorithm is optimally efficient. In addition, this method gives the possibility to distinguish between single connected components, or contours, of the isosurface.

The problem with this method is knowing at which cell to start. You need at least one *seed cell* for all isovalues and contours in the data. As we will see, the contour tree provides a good structure for solving this as initially shown by van Kreveld *et al.* introducing the *minimal seed sets* [vKvOB<sup>+</sup>97]. Later, Carr and Snoeyink [CS03] introduced the *path seeds*, which will be further described in Section 3.2.2.

## 3.2 The Contour Tree

As introduced in Section 2.3, the contour tree is a topological representation of the contours of a function. It follows from *Morse theory* as a special case of the *Reeb graph* which describes arbitrary contours as a parameter changes. This parameter could be any parameter of the function, such as spatial coordinate or time. When the parameter is the *isovalue*, the Reeb graph is called a contour tree. A special property of the contours states that no two contours can intersect. Thus we can have no cycles in the graph, why the contour tree is indeed a *tree*.

### 3.2.1 Computation

The first contour tree algorithm called *contour nesting* was introduced by Boyell and Ruston [BR63] in 1963. Their method may have been manual as they manually constructed the contours from a topographic map. The second algorithm, using a heuristic technique, was proposed by Itoh and Koyamada [IK94] in 1994. In 1995, Takahashi *et al.* [TIS<sup>+</sup>95] proposed a method called *monotone path search*. This algorithm was later extended to 3D by Takahashi, Fujishiro and

Takeshima [TTF04] with a runtime complexity of  $O(n^2)$  where  $n$  is the number of vertices in the mesh. The *contour sweep* algorithm was introduced by van Kreveld *et al.* in 1997 [vKvOB<sup>+</sup>97], with a time complexity of  $O(N \log N)$  in 2D, where  $N$  is the number of cells in the mesh. However, the implementation was complex, and they stated a runtime of  $O(N^2)$  in dimensions higher than two. Based on this algorithm, Carr, Snoeyink and Axen [CSA03] proposed *sweep and merge* with a runtime complexity of  $O(n \log n + N)$  in all dimensions. We will present this algorithm next.

### Sweep and Merge

In the sweep and merge algorithm, the contour tree is computed by *sweeping* through the data from high to low isovalue, and the reverse, to create the *join tree* and the *split tree* respectively. These trees are *merged* to form the contour tree. Basically, the join tree represents the connectivity of  $\{\mathbf{x} \in \mathbb{R}^d : f(\mathbf{x}) \geq h\}$ , where  $d$  is the dimensions, while the split tree represents the connectivity of  $\{\mathbf{x} \in \mathbb{R}^d : f(\mathbf{x}) \leq h\}$ .

Intuitively, one can compare the process of computing the join tree with draining a landscape flooded with water while keeping track of the peaks appearing at the water surface. Computing the split tree is the exact opposite, flooding a landscape while keeping track of the parts dropping below the water surface. For example, consider the landscape of Vancouver as shown in Figure 2.3(a). The computation of the join tree is illustrated in Figure 3.6. The first peak to appear above surface is  $A$ , shortly followed by  $B$  and  $C$ .  $A$  and  $C$  joins in  $D$  and the process continues. The efficient implementation of this algorithm depends on sorting the vertices according to height and using a union-find structure to handle the connected components in constant time. For further details, the reader is referred to [CSA03] and [Car04]. The final join and split trees are shown in Figure 3.7.

The final contour tree is constructed in the merge phase. One can show that the *up-degree* (number of edges upwards from a node) is the same in the join tree and the contour tree. Likewise, the *down-degree* (number of edges downwards from a node) is the same in the split tree and the contour tree. So, we can assemble the contour tree by picking either upper leaf nodes from the join tree or lower leaf nodes from the split tree, transferring them to the contour tree in an arbitrary order. This process is depicted in Figure 3.8. The final contour tree is depicted in Figure 2.4(b).

An example of a contour tree for a simple 3D dataset is shown in Figure 3.9. Here, we have four contours at isovalue (a). At (b) and (c) these contours join to form two rings. As the isovalue decreases we get two contours at (f). At this stage the inner contour is effectively occluded in the image. Indeed, it would be very hard to tell what is actually occluded without the information provided by the contour tree.

#### 3.2.2 Path Seeds and Piecewise Continuation

The continuation method as introduced in Section 3.1.2 is an optimally efficient method for extracting isosurfaces. In addition, it gives the possibility to distinguish between different contours



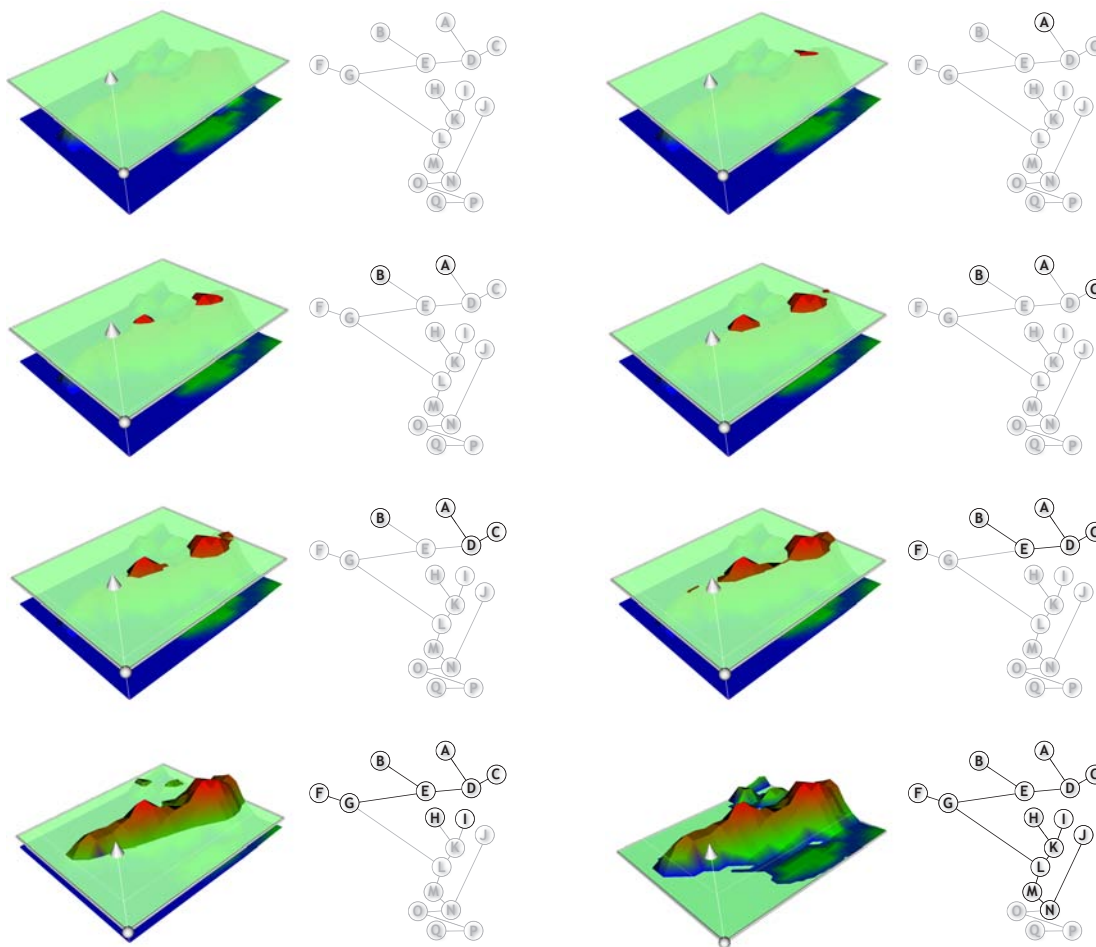
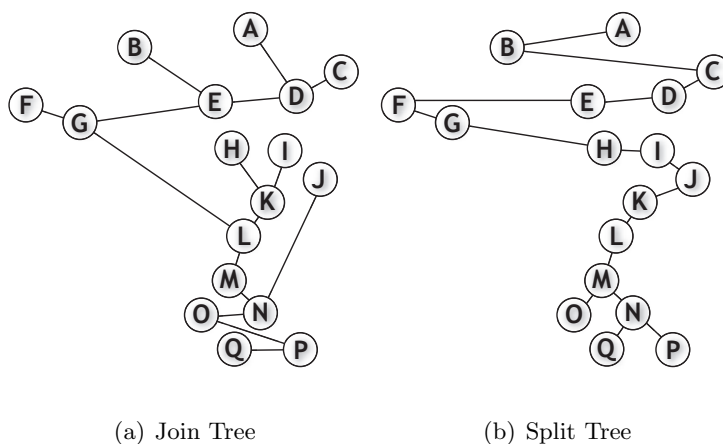


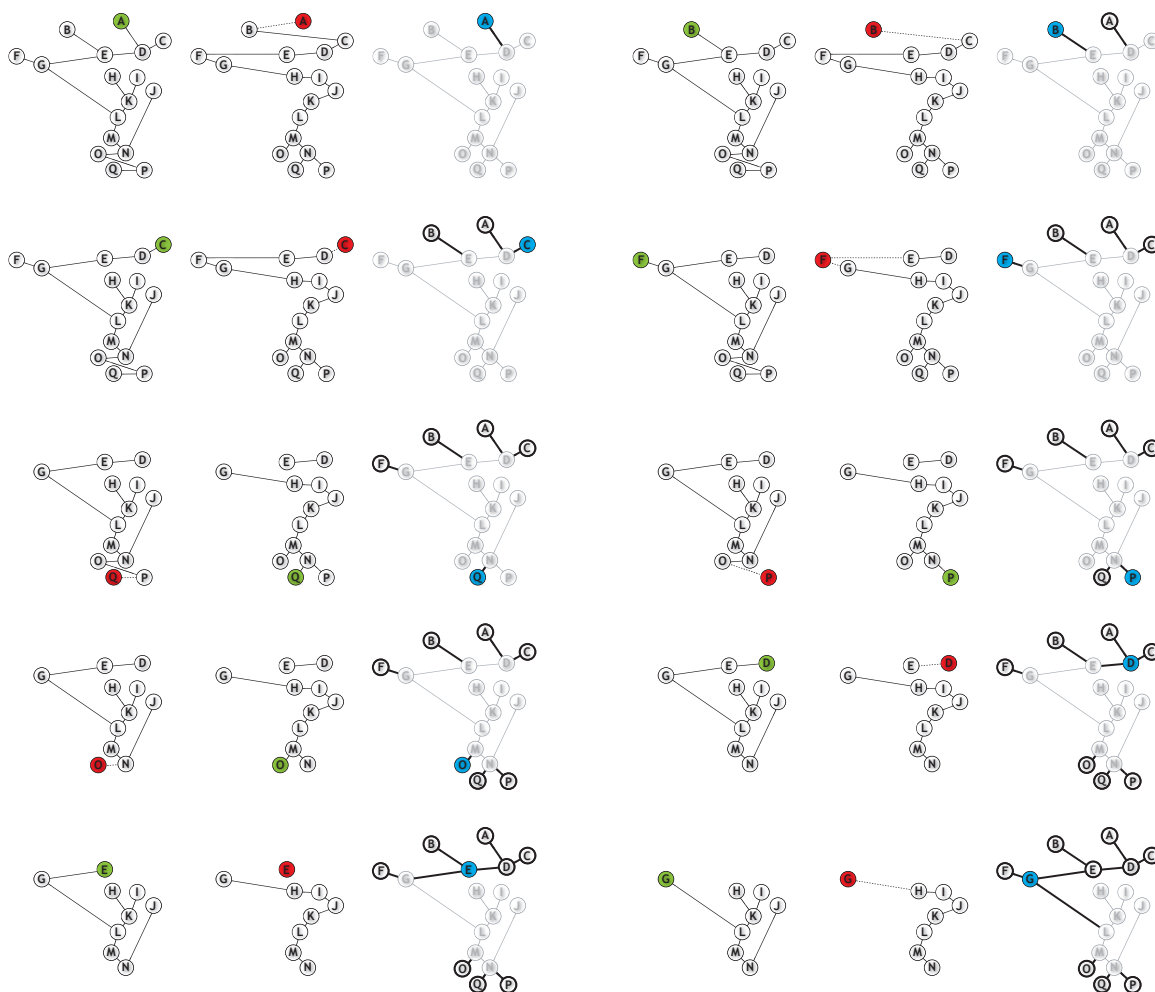
Figure 3.6: Computation of the Join Tree



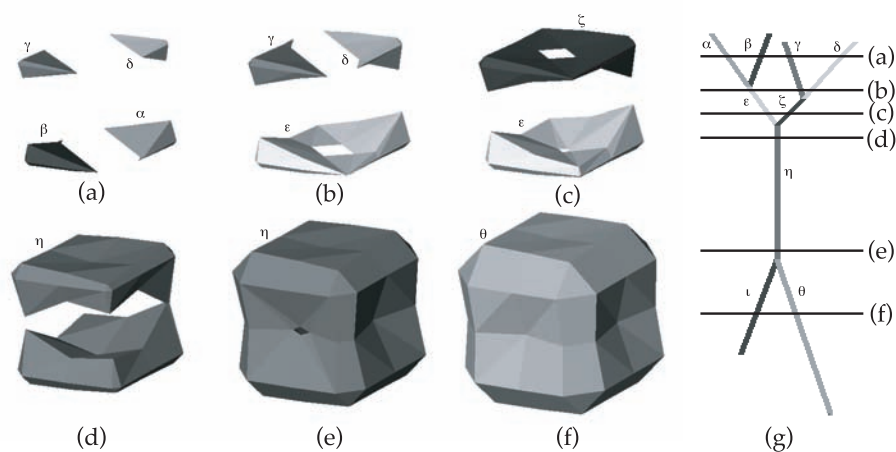
(a) Join Tree

(b) Split Tree

Figure 3.7: Join Tree and Split Trees



**Figure 3.8:** Computation of the Contour Tree. The join and split trees are merged by picking upper leaf nodes from the join tree or lower leaf nodes from the split tree. The green node is selected for transfer to the contour tree. For space reasons, the end of the process is omitted.



**Figure 3.9:** Contour Tree for 3D dataset [CS03]

of the isosurface. The main problem with this method is knowing at which *seed cells* to start. As presented by Carr and Snoeyink in [CS03], the contour tree provides an elegant solution for this. Instead of storing the seed cells explicitly, each edge in the contour tree corresponds to a monotone path which leads to the desired contour. For example, to extract the contour denoted by  $\alpha$  in Figure 3.9, we observe that the contour corresponds to a point on the edge  $(\alpha, \epsilon)$ . Thus, to find a suitable seed cell for continuation, we start at the vertex represented by  $\epsilon$  in the contour tree, and walk up-hill in a given direction until we reach the desired isovalue. The computation of these *path seeds* is a simple extension to the computation of the contour tree and has small storage requirements.

By construction, the path seeds in the contour tree give the additional benefit of extracting *individual contours* of an isosurface by *piecewise continuation* [CS03]. This is achieved by following a specific surface in each cell, in contrast to the original continuation method which tracked *all* surfaces in a cell. This idea is the foundation of the *flexible isosurfaces* as introduced in Section 2.3. This allows the user to select and evolve individual contours by picking the corresponding edge in the tree. Thus, it is possible to locate and extract contours of different isovalues, providing a great tool for exploratory visualization. In this thesis we use the flexible isosurfaces to successfully locate particular objects to initialize the segmentation process. This will be further described in Chapter 4.

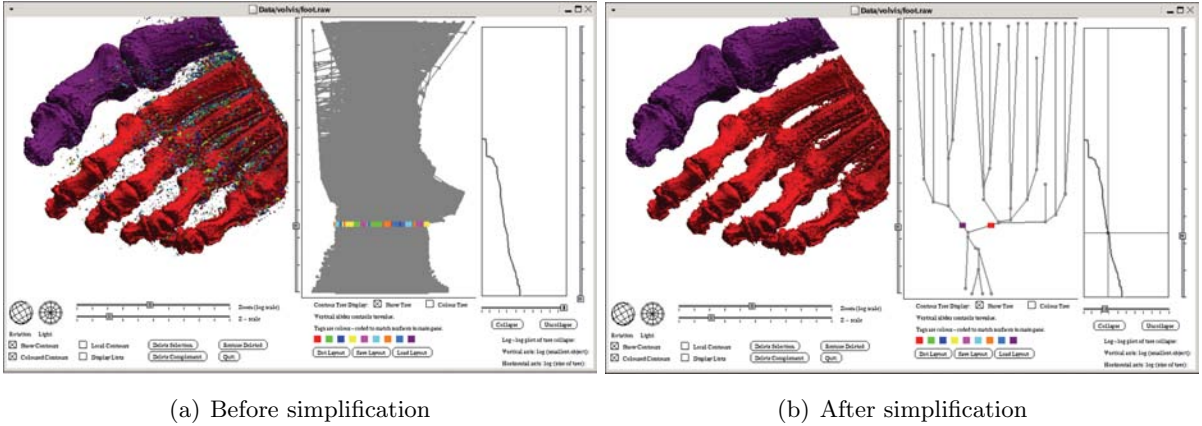
### 3.2.3 Local Geometric Measures and Simplification

In [Car04], Carr shows how the merge phase in the computation is equivalent to sweeping through the dataset one contour at a time. In principal, these sweeps pass through subregions of the data, topologically defined by the contours. A powerful consequence of this is the idea of *local geometric measures*. During the sweep through each subregion, the contours can be augmented with geometric properties in that region such as area, volume, variance etc.

One important application of these measures is the *simplification* of the contour tree as showed in [Car04]. Since the contour tree detects all critical points in a dataset, it is very susceptible to noise. Thus, a standard medical dataset can result in a contour tree with millions of edges, obviously useless for human interaction and for most numerical algorithms. By using the geometric properties of each contour, we can simplify the contour tree by applying a certain geometric condition. For example, we can remove, or flatten, all contours corresponding to a peak below a certain height. This is a very effective and reliable technique for “cleaning up” the contour tree of a noisy medical dataset, making it useful for exploratory visualization and other techniques. An example of a useful simplification is shown in Figure 3.10(a) before simplification and Figure 3.10(b) after simplification. Figure 3.10(b) clearly represents the five toes as five distinct branches in the tree.

### 3.2.4 Extending to any Mesh and Interpolant

The original algorithm assumed a simplicial mesh with barycentric interpolation. As discussed in Section 3.1 this mesh type contains all critical points at the vertices of the mesh, why only



**Figure 3.10:** A noisy contour tree before and after simplification [Car04]

the vertices need to be considered during the computation of the contour tree. When using bilinear or trilinear interpolation, the topological complexity increases since saddle points can exist *inside* each cell.

The computation was first extended to the trilinear interpolant for a cubic mesh by Pascucci and Cole-McLaughlin in [PCM03]. They proposed a *divide-and-conquer* strategy which recursively subdivided the mesh to a single cell. An “oracle” analyzed this cell to provide one of four possible join (or split) trees, which was merged with the neighbors through the recursive call.

Carr [Car04] generalizes and formalizes this approach to form *join graphs* and *split graphs*. By defining all possible join or split trees for a cell, these graphs reveal the contour tree for any mesh and interpolant. In [Car04], Carr presents this approach for the bilinear and trilinear interpolant in addition to the Marching Cubes cases.

### 3.3 Level Set Methods

Level set methods was briefly introduced in Section 2.4. This section will expand on this by explaining the underlying mathematics and the basic numerical implementation. The level set method implicitly represents an interface as a specific level set  $S$  with isovalue  $h$  of the level set function  $\phi$ , such that:

$$S = \{\mathbf{x} \in \mathbb{R}^d : \phi(\mathbf{x}) = h\} \quad (3.2)$$

Although applicable to any dimension, we generally deal with interfaces where  $d = 2$  (isolines) or  $d = 3$  (isosurfaces). For further discussion, we introduce some concepts. It can be shown that the *normal* of any level set of  $\phi$  is  $\mathbf{n} = \nabla\phi / |\nabla\phi|$  [MBW<sup>+</sup>05]. We use this to define the *level set speed function*:

$$F(\mathbf{x}, \mathbf{n}, \phi, \dots) = \mathbf{n} \cdot \frac{d\mathbf{x}}{dt} = \frac{\nabla\phi}{|\nabla\phi|} \cdot \frac{d\mathbf{x}}{dt} \quad (3.3)$$

This can be interpreted as the speed  $d\mathbf{x}/dt$  at a point  $\mathbf{x}$  on the interface projected onto the normal at that point. That is, the speed in the normal direction. As we will see, the speed function

provides the user with the means of modifying the level set function, and thus the interface, over time. To make this possible, we derive equations of motion for the level set function by introducing time dependence to Equation 3.2. There are two distinct ways of doing this, the first of which varies the isovalue  $h$  over time, such that  $S = \{\mathbf{x}(t) \in \mathbb{R}^d : \phi(\mathbf{x}(t)) = h(t)\}$ . This is called the *static level set formulation*. It describes the evolution of level sets of a function as the isovalue changes. However, the level sets cannot intersect by definition, imposing a great constraint on the deformations possible. The second option of introducing time dependence is to vary the level set function itself over time, such that  $S = \{\mathbf{x}(t) \in \mathbb{R}^d : \phi(\mathbf{x}(t), t) = h\}$ . To derive equations of motion for the interface, we differentiate with respect to time, rearrange and apply the definition of the speed function in Equation 3.3:

$$\frac{\partial \phi}{\partial t} = -\nabla \phi \cdot \frac{d\mathbf{x}}{dt} \quad (3.4a)$$

$$= -|\nabla \phi| F(\mathbf{x}, \mathbf{n}, \phi, \dots) \quad (3.4b)$$

These partial differential equations (PDEs), known as the *fundamental level set equations*, give the *dynamic level set formulation*. This allows for any deformations of the interface, making it the method of choice for most applications. In theory, the choice of  $h$  is arbitrary. However, it makes practical sense to use  $h = 0$  as this allows the definition of an *inside* and an *outside* of the interface using a simple sign convention. In this thesis we define the set of points inside,  $S_{inside}$ , and the set of points outside,  $S_{outside}$ , such that:

$$S_{inside} = \{\mathbf{x} \in \mathbb{R}^d : \phi(\mathbf{x}(t), x) < 0\} \quad (3.5a)$$

$$S_{outside} = \{\mathbf{x} \in \mathbb{R}^d : \phi(\mathbf{x}(t), x) > 0\} \quad (3.5b)$$

Equation 3.4 assumes that the level set function  $\phi$  is continuous with well defined gradients everywhere in the domain. However, to use these mathematical expressions on a computer, we need to discretize the function, only computing values at discrete points. More specifically, we have to discretize both the *temporal domain* and the *spatial domain*. Depending on the particular PDE used, this introduces a number of numerical implications, constraining the speed at which we can evolve the PDE. Next we will study the temporal and spatial discretizations separately.

### 3.3.1 Temporal Discretization

The temporal discretization determines how we evolve Equation 3.4 in time. Since the time needs to be discrete, we evolve the PDE using discrete time steps of  $\Delta t$ , using either an *implicit* or *explicit* integration scheme. The implicit schemes have the advantage of being *unconditionally stable*, regardless of the time step  $\Delta t$ . However, since the computation of these schemes relies on solving a large system of equations, they are computationally heavy and often hard to implement. Explicit methods, on the other hand, are easy to implement but suffer from stability constraints imposed on the time step. Despite this constraint, explicit methods are often used for solving level set equations. A simple, first order accurate, explicit scheme is the *forward Euler* given by:

$$\frac{\partial \phi}{\partial t} \approx \frac{\phi^{n+1} - \phi^n}{\Delta t} \quad (3.6)$$

where  $\phi^n$  denotes the values of  $\phi$  at time instance  $t^n$  while  $\phi^{n+1}$  denotes the values of  $\phi$  at the time instance  $t^n + \Delta t$ . Although practical experience shows that this first order scheme usually suffices when solving level set equations, the accuracy can be improved by using the *total variation diminishing Runge-Kutta* (TVD RK) schemes proposed by Shu and Osher [SO88]. The first order TVD RK scheme is simply the forward Euler. The accuracy is increased by sequentially taking Euler steps and linearly combining the results. The second order TVD RK scheme, also known as the midpoint rule, advances the solution by two Euler steps to get  $\phi^{n+2}$  at time instance  $t^n + 2\Delta t$ . The solution at  $\phi^{n+1}$  is the average:

$$\phi^{n+1} = \frac{1}{2}\phi^n + \frac{1}{2}\phi^{n+2} \quad (3.7)$$

The third order TVD RK scheme proceeds by advancing the solution by Euler steps to  $\phi^{n+2}$ , followed by a linear combination to get  $\phi^{n+\frac{1}{2}}$ . This solution is advanced to  $\phi^{n+\frac{3}{2}}$ , followed by a final linear combination to get  $\phi^{n+1}$ :

$$\phi^{n+\frac{1}{2}} = \frac{3}{4}\phi^n + \frac{1}{4}\phi^{n+2} \quad (3.8a)$$

$$\phi^{n+1} = \frac{1}{3}\phi^n + \frac{2}{3}\phi^{n+\frac{3}{2}} \quad (3.8b)$$

### 3.3.2 Spatial Discretization

Since the spatial discretization strongly depends on the behavior of the particular PDE used we will study two different versions of Equation 3.4, giving rise to two fundamental types of PDEs, *hyperbolic* type and *parabolic* type.

#### Hyperbolic Advection

To introduce *hyperbolic advection*, two versions of Equation 3.4 can be stated as:

$$\frac{\partial \phi}{\partial t} = \mathbf{V} \cdot \nabla \phi \quad (3.9a)$$

$$= a |\nabla \phi| \quad (3.9b)$$

This corresponds to *advecting* (transporting) the interface in a vector field  $\mathbf{V}$  (Equation 3.9a) or in the direction of the surface normal (Equation 3.9b). This type of equation could be used for erosion or dilation of a surface and is often found in computational physics describing a *flow* of information in a certain direction. Imagine a *numerical wave*, propagating information in a flow field as directed by Equation 3.9. The Courant-Friedrichs-Lewy (CFL) stability condition [CFL28] states that only the sample points *behind*, or *up-wind* to, the wave should be used in the discretization. This makes intuitive sense, since sample points in front of the wave have not been “touched” by the flow field. Thus, they do not contain any reliable information needed to evaluate the partial derivatives. Considering Equation 3.9a, this results in the finite difference (FD) approximation given by:

$$\frac{\partial \phi}{\partial x} \approx \begin{cases} \phi_x^+ = (\phi_{i+1,j} - \phi_{i,j}) / \Delta x & \text{if } V_x < 0 \\ \phi_x^- = (\phi_{i,j} - \phi_{i-1,j}) / \Delta x & \text{if } V_x > 0 \end{cases} \quad (3.10)$$

This introduces some new notation. Without loss of generality, we assume a 2D level set function  $\phi$ . Then,  $\phi_{i,j}$  is the discrete sample point at position  $(i, j)$  in the grid, while  $\Delta x$  is the grid spacing along the  $x$  axis. The *stencil* defines the grid points which are used by the operator. In Equation 3.10, the stencil is two grid points wide, meaning the operator uses information from two sample points. For Equation 3.9b the direction of the flow is not known explicitly. For this we can use Godunov’s method [RT92] to evaluate the partial derivatives:

$$\left(\frac{\partial\phi}{\partial x}\right)^2 \approx \begin{cases} \max[\max(\phi_x^-, 0)^2, \min(\phi_x^+, 0)^2] & a < 0 \\ \max[\min(\phi_x^-, 0)^2, \max(\phi_x^+, 0)^2] & a > 0 \end{cases} \quad (3.11)$$

These discretizations result in a *first order accurate* approximation of the partial derivatives. For higher order approximations, the *essentially non-oscillatory* (ENO) [OS91] or *weighted ENO* (WENO) [LOC94] polynomial interpolation techniques can be used. The basic idea is to provide better approximations to  $\phi_x^-$  and  $\phi_x^+$  by the smoothest combination of grid points in an up to six point wide stencil. These methods give a 3rd to 5th order accurate approximation depending on the smoothness around a grid point  $(i, j)$ . For more information on the actual implementation, we refer the reader to [OF03].

When using an explicit method for the temporal discretization, the CFL condition defines the stability condition for the time step  $\Delta t$ . Since information is propagated one grid point each time step the numerical wave speed along the  $x$  axis is  $\Delta x/\Delta t$ . This speed needs to be at least as fast as the physical wave speed given by  $\mathbf{V}$  or  $a$  in Equation 3.9 to ensure stability. It is common to use the *CFL number*  $\alpha \in (0, 1)$  to specify the rate of stable propagation. When considering a two dimensional level set function, the time step can be chosen such that:

$$\Delta t = \alpha \min \left\{ \frac{\Delta x}{|V_x|}, \frac{\Delta y}{|V_y|} \right\} = \alpha \frac{\min \{\Delta x, \Delta y\}}{|a|} \quad (3.12)$$

### Parabolic Diffusion

To introduce *parabolic diffusion*, a version of Equation 3.4 can be stated as:

$$\frac{\partial\phi}{\partial t} = \alpha K |\nabla\phi| \quad (3.13)$$

where  $\alpha$  is a scaling parameter and  $K$  is curvature. This equation has its correspondence in physics as the geometric heat equation, describing a *diffusion* problem. For level set surface deformations it has been shown to minimize surface area, or to *smooth* the surface [MBW<sup>+</sup>05]. This type of PDE is very different from the hyperbolic type in that the information flow has no particular direction. This means that a solution at a point at a particular time step can depend on any other point in the domain at the previous time step. For the spatial discretization, we need to use the second order accurate central difference given by:

$$\frac{\partial\phi}{\partial x} \approx \phi_x^\pm = \frac{\phi_{i+1,j} - \phi_{i-1,j}}{2\Delta x} \quad (3.14)$$

In [MBW<sup>+</sup>05] it is shown that the curvature  $K$  is described by the divergence of the normal, that is  $K = \nabla \cdot \mathbf{n}$  (in 2D). Expanding this gives the following expression for  $K$ :

$$K = \frac{\phi_x^2 \phi_{yy} + \phi_y^2 \phi_{xx} - 2\phi_x \phi_y \phi_{xy}}{(\phi_x^2 + \phi_y^2)^{2/3}} \quad (3.15)$$

where  $\phi_x = \partial\phi/\partial x$  and  $\phi_{xy} = \partial^2\phi/\partial x\partial y$ . This expression can be discretized using the following second order central difference scheme:

$$\frac{\partial^2\phi}{\partial^2x} \approx \frac{\phi_{i+1,j} - 2\phi_{i,j} + \phi_{i-1,j}}{\Delta x^2} \quad (3.16a)$$

$$\frac{\partial^2\phi}{\partial x\partial y} \approx \frac{\phi_{i+1,j+1} - \phi_{i+1,j-1} + \phi_{i-1,j-1} - \phi_{i-1,j+1}}{4\Delta x\Delta y} \quad (3.16b)$$

The same derivations can be made in 3D, where the mean curvature is given by  $K = (\nabla \cdot \mathbf{n})/2$  [MBW<sup>+</sup>05]. When considering stability constraints for the time step  $\Delta t$ , we note that the information travels at an infinite speed. Thus, the CFL stability condition in Equation 3.12 is not applicable for the parabolic type PDE. For a two dimensional level set function, the stability constraint can be derived by a von Neumann stability analysis [Str89]:

$$\Delta t < \left( \frac{2\alpha}{\Delta x^2} + \frac{2\alpha}{\Delta y^2} \right)^{-1} = \{\Delta x = \Delta y\} = \frac{\Delta x^2}{4\alpha} \quad (3.17)$$

### 3.3.3 The Signed Distance Property and Reinitialization

We previously stated that  $\phi$  needs to be continuous with well defined gradients, in order to solve Equation 3.4. However, because of the spatial discretization, this requirement can be relaxed. More specifically, we require  $\phi$  to be *Lipschitz continuous*, satisfying the following condition:

$$|\phi(\mathbf{x}_0) - \phi(\mathbf{x}_1)| \leq C|\mathbf{x}_0 - \mathbf{x}_1| \quad (3.18)$$

where  $C \geq 0$ . This means that the gradients of  $\phi$  can be discontinuous, although the *rate-of-change* of  $\phi$  must be bounded by the finite Lipschitz constant  $C$ . This constant is affected by the numerical schemes used to solve the discretized equations. In order to assure stability,  $C \approx 1$  meaning that  $\phi$  has to approximately satisfy the *Eikonal equation*:

$$|\nabla\phi| = 1 \quad (3.19)$$

Intuitively, this means that the level sets of  $\phi$  have a uniform density. In other words, as you walk in a direction orthogonal to the level set interface, the distance increases one unit each step. Combined with Equation 3.5, this means that  $\phi$  is a *signed distance function*. This constraint can be motivated by considering Equation 3.9b and Equation 3.13. If  $|\nabla\phi|$  would drift away from one, the speed of the *information flow* would increase, implying that we need to evolve the PDE with smaller time steps. However, it is often more efficient to keep the time step fixed while making sure  $\phi$  approximately satisfies the Eikonal equation.

This process is called *reinitialization* and should be performed more or less frequently depending on how sensitive the discretization of the particular PDE is to numerical fluctuations. There are a number of ways of doing this, the first of which solves the *reinitialization equation* to steady state [SSO94]:

$$\frac{\partial\phi}{\partial t} = S(\phi)(1 - |\nabla\phi|) \quad (3.20)$$



where  $S(\phi)$  is the *sign* of  $\phi$ , with a smooth numerical approximation given by [PMO<sup>+</sup>99]:

$$S(\phi) = \frac{\phi}{\sqrt{\phi^2 + |\nabla\phi|^2 \Delta x^2}} \quad (3.21)$$

At steady state  $\partial\phi/\partial t = 0$ , implying  $|\nabla\phi| = 1$ . Since Equation 3.20 is an hyperbolic type PDE,  $\nabla\phi$  needs to be discretized using an up-wind scheme as discussed in Section 3.3.2. This approach has a runtime complexity of  $O(N)$ , where  $N$  is the number of grid points.

Another way of maintaining the signed distance property is to solve the Eikonal equation directly, using for example the *fast marching method* as proposed by Sethian in [Set96]. The idea behind this method can be described by sequentially adding “layers” to the interface. It walks around the interface, computing the *time-of-arrival* at each grid point immediately adjacent to the current layer. When a layer is completed, a second layer is added with a later time-of-arrival. Using a heap datastructure, this algorithm has a runtime complexity of  $O(N \log N)$ . A faster, and simpler, algorithm called *fast sweeping* was later proposed by Zhao [Zha05]. This is based on a number of sequential sweeps through the grid, following the characteristics of the Eikonal equation in a certain direction in each sweep. The complexity is optimally  $O(N)$  and in  $d$  dimensions,  $2^d$  sweeps are necessary. However, this method is not easily adapted to narrow band schemes, which we will present next.

### 3.3.4 Narrow Band Schemes

The level set method as initially proposed solves the level set equation on the entire domain of  $\phi$ . However, for most applications, the method is used to track the motion of a particular interface, or level set, of  $\phi$  with isovalue zero. Therefore, the computations can be restricted to a neighborhood around the zero crossing, with the same results as solving the equations on the entire domain. Using such a *narrow band* scheme, the computational cost is proportional to the size of the interface. This makes a considerable difference for the running time, especially when propagating surfaces in three dimensions or higher. This idea was first introduced by Adalsteinsson and Sethian in [AS95] and further improved by Peng *et al.* in [PMO<sup>+</sup>99]. We will describe the scheme by Peng *et al.* further.

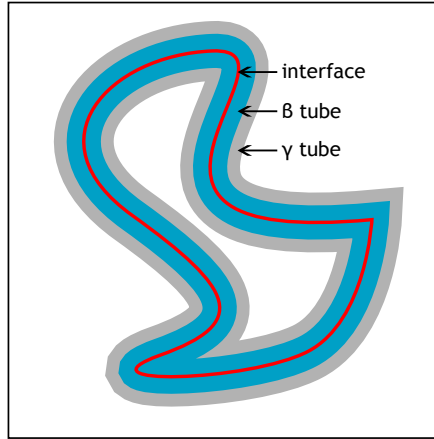
Using this approach we define two narrow band *tubes*,  $T_\beta$  and  $T_\gamma$  such that:

$$T_\beta = \{\mathbf{x} \in \mathbb{R}^d : \phi(\mathbf{x}) < \beta\} \quad (3.22a)$$

$$T_\gamma = \{\mathbf{x} \in \mathbb{R}^d : \phi(\mathbf{x}) < \gamma\} \quad (3.22b)$$

where  $0 < \beta < \gamma$ . This is depicted in Figure 3.11. The level set equations will now be solved only within these tubes. However, to avoid numerical oscillations at the tube boundary, we introduce a cut-off function:

$$c(\phi) = \begin{cases} 1 & \text{if } |\phi| \leq \beta \\ \frac{(2|\phi| + \gamma - 3\beta)(|\phi| - \gamma)^2}{(\gamma - \beta)^3} & \text{if } \beta < |\phi| \leq \gamma \\ 0 & \text{if } |\phi| > \gamma \end{cases} \quad (3.23)$$



**Figure 3.11:** Narrow band displaying the interface, the  $\beta$  tube and the  $\gamma$  tube by Peng *et al.* [PMO<sup>+</sup>99]

Equation 3.4 is modified using the cut-off function such that:

$$\frac{\partial \phi}{\partial t} = -c(\phi) \nabla \phi \cdot \frac{d\mathbf{x}}{dt} \quad (3.24a)$$

$$= -c(\phi) |\nabla \phi| F(\mathbf{x}, \mathbf{n}, \phi, \dots) \quad (3.24b)$$

Thus, the level set equation is solved exactly only within  $T_\beta$ . In order to rebuild the tubes, we note that the numerical scheme can only move the interface at a maximum of one grid point each time step. Assuming a grid spacing of  $\Delta x$ , we use this to define a tube  $N = \{\mathbf{x} \in \mathbb{R}^d : \phi(\mathbf{x} + \mathbf{x}_0) < \gamma, |\mathbf{x}_0| \leq \Delta x\}$  in which we perform the reinitialization described in the previous section. In other words, we reinitialize in a tube corresponding to  $T_\gamma$  dilated by one grid point. After the reinitialization, we have the correct signed distance function in  $N$  which is required to contain all  $\{\mathbf{x} \in \mathbb{R}^d : \phi(\mathbf{x}) < \gamma\}$ . Using  $N$ , we can now rebuild the tubes  $T_\beta$  and  $T_\gamma$ . The actual choice of  $\beta$  and  $\gamma$  depends on the stencil used for the spatial discretization. For a first order scheme,  $\beta = 2\Delta x, \gamma = 3\Delta x$  could prove sufficient, while the 3rd to 5th order WENO scheme could require  $\beta = 4\Delta x, \gamma = 6\Delta x$ .

This approach is *computationally efficient* since it only computes the level set equations and reinitializes in a narrow band around the interface of principal interest. The implementation is simple and the rebuilding of the tubes can be optimized to give an overall runtime of  $O(N)$  where  $N$  is the size of the interface. However, it is not *memory efficient* since it stores the full level set function. Moreover, since the narrow band is stored scattered in memory, the CPU cache can not be efficiently used.

An approach which is both computationally and memory efficient is the *dynamic tubular grid* (DT-grid) proposed by Nielsen and Museth [NM06]. It is based on the method by Peng *et al.* but only stores the actual narrow band in memory. Not only does this reduce storage requirements, but it also very efficiently uses the CPU cache for sequential memory access. Moreover, since the narrow band can be seen as “floating” in free space, it is not constrained within any grid boundaries and can expand freely in an *out-of-the-box* like behavior.

### 3.4 Level Set Segmentation with Edges

Level set segmentation with edges aims to find a segmentation of a particular object based on sharp edges. These methods are often based on the classical “snakes” model of [KWT88] which minimizes a *variational functional* based on balancing the influence of sharp edges against curve smoothness. This idea was originally formulated using a parameterized curve but later adapted to the level set framework in [CCCD93] and [CKS97]. The latter work define an implicit energy  $F$  over the full domain  $\Omega$  of the level set function  $\phi$ , such that:

$$F(\phi) = \int_{\Omega} E(I(\mathbf{x})) |\nabla H(\phi(\mathbf{x}))| d\Omega \quad (3.25)$$

where  $H$  is a *Heaviside* function, usually numerically approximated by:

$$H(z) \approx \begin{cases} 1 & \text{if } \phi < -\epsilon \\ \frac{1}{2} - \frac{z}{2\epsilon} + \frac{1}{2\pi} \sin\left(-\frac{\pi z}{\epsilon}\right) & \text{if } |\phi| \leq \epsilon \\ 0 & \text{if } \phi > \epsilon \end{cases} \quad (3.26)$$

and  $E$  is an edge detector of the image  $I$  subject to segmentation, often defined such that:

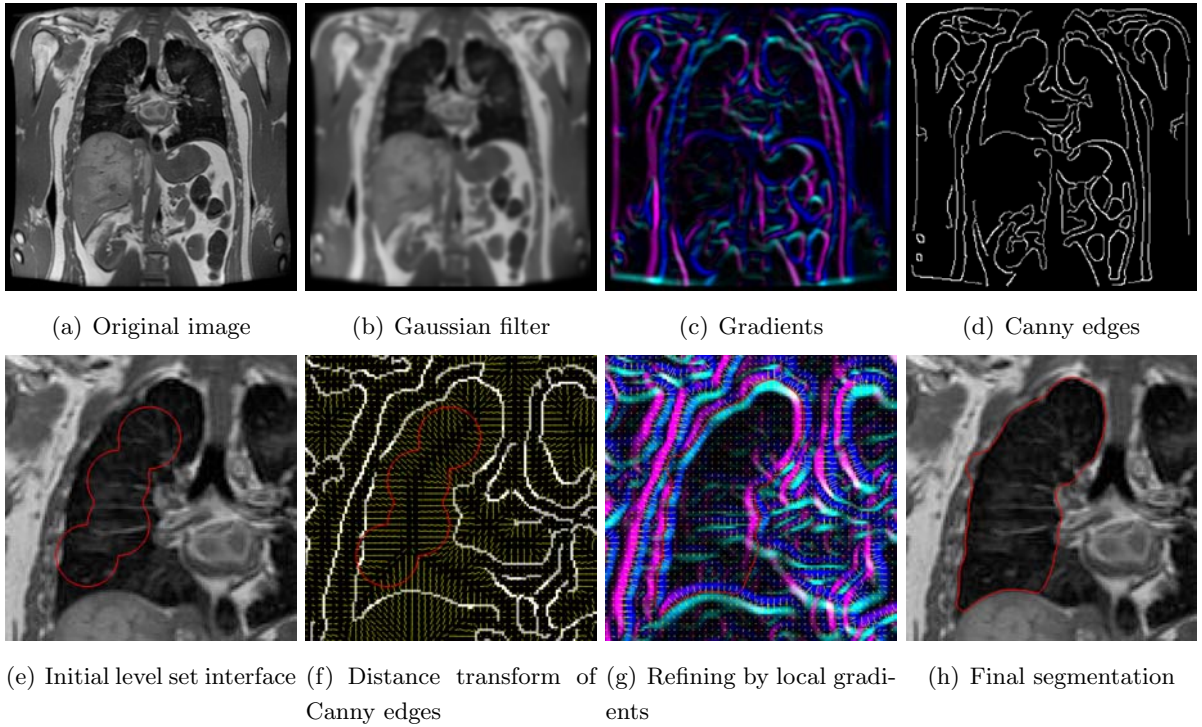
$$E(I(\mathbf{x})) = \frac{1}{1 + |\nabla(G_{\sigma} * I(\mathbf{x}))|^p} \quad (3.27)$$

where  $p \geq 1$  and  $G_{\sigma} * I$  is a convolution of the image with a Gaussian filter kernel of variance  $\sigma$ . We note that  $E \approx 1$  as  $\nabla(G_{\sigma} * I(\mathbf{x})) \approx 0$  and that  $E \approx 0$  as  $\nabla(G_{\sigma} * I(\mathbf{x}))$  grow. In other words,  $E$  is approximately zero at the edges in the image and approximately one elsewhere. We also note that the term  $|\nabla H(\phi(\mathbf{x}))|$  in Equation 3.25 is only non-zero in a neighborhood around the level set interface. Thus, the functional  $F$  is minimized only when the level set interface coincides with the edges in the image. This problem can be solved using level set methods by deriving the gradient descent minimization of the corresponding Euler-Lagrange equation as described in [ZCMO96]. This results in the following PDE:

$$\frac{\partial \phi}{\partial t} = E(I(\mathbf{x}))K |\nabla \phi| + \nabla E(I(\mathbf{x})) \cdot \nabla \phi \quad (3.28)$$

where  $K$  is mean curvature. We recognize these terms from Equation 3.9 and Equation 3.13. The first term is a diffusion term corresponding to a “smoothing” of the interface, while the second term advects the interface in the direction of the edges in the image.

While this formulation has proven very successful it suffers from some limitations. Firstly, the target object for segmentation must be well defined by its edges. This is further complicated by the Gaussian filter which naturally smoothes all edges in the image in addition to suppressing high frequency noise. Secondly, since most edge detectors are short-ranged the *initialization* (initial level set interface) must be close to the edges of the object. The latter problem can be solved by using a long-range edge detector, like the Canny edges [Can86], for the initial advection as described in [MBZW02]. However, since the gradients of the image are bounded, the edge detector  $E$  can never be zero. Thus, stable convergence is a common problem when using this method. A sequence of images showing a segmentation with the Canny edge detector is shown in Figure 3.12. We see that the final result is not completely satisfactory, since the initial level set interface is too far from the edges of the lung, particularly in the lower right region.



**Figure 3.12:** A sequence of images showing a segmentation with edges using the Canny edge detector.

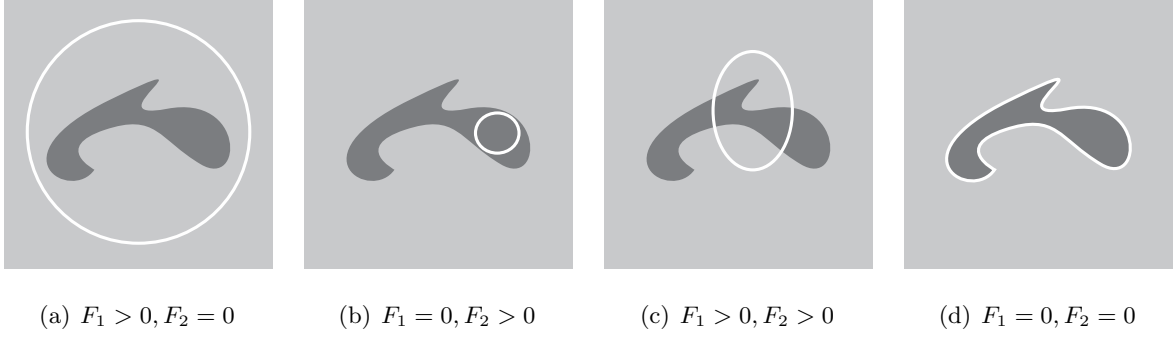
### 3.5 Level Set Segmentation without Edges

Recent work by Chan and Vese starting with [CV01] describes a level set segmentation approach which does not rely on edge information. They replace the edge based model with a reduced case of the Mumford-Shah functional for segmentation [MS89] called the minimal partition problem. The aim of this partition problem is to find a boundary in the image which best approximates the image by two piecewise-constant regions. This approach solves the main problems with the method described in the previous section. In addition to finding objects not necessarily defined by gradient, the final solution is global and insensitive to the initialization. However, the global behavior is not always appropriate, and can be considered a disadvantage of this method. We will explain this further in Chapter 4.

To simplify terminology, we assume that the level set interface is a *curve* defined using a two dimensional level set function. The energy functional in [CV01] is given as:

$$F(\phi) = \int_{\Omega} [\mu |\nabla H(\phi)| + \nu H(\phi) + \lambda_1 H(\phi)(I - c_1)^2 + \lambda_2 H(-\phi)(I - c_2)^2] d\Omega \quad (3.29)$$

Note that  $\int_{\Omega} |\nabla H(\phi)| d\Omega$  and  $\int_{\Omega} H(\phi) d\Omega$  give the *curve length* and *area* inside the curve respectively, where  $H$  is the Heaviside function (Equation 3.26). Thus, the first two terms correspond to minimization of curve length (smoothing) and area. Secondly we note that the third term is non-zero only *inside* the curve while the fourth term is non-zero only *outside* the curve. The constants  $\mu$ ,  $\nu$ ,  $\lambda_1$  and  $\lambda_2$  control the influence of each term. Usually,  $\lambda_1 = \lambda_2 = 1$  and  $\nu = 0$  are kept fixed while  $\mu$  is varied to control the smoothness of the solution. The constants  $c_1$  and  $c_2$  are the approximate piecewise-constant intensities inside and outside the curve respectively.



**Figure 3.13:** Motivation for the functional in [CV01] using a simple example,

where  $F_1 = \int_{\Omega} H(\phi)(I - c_1)^2 d\Omega$  and  $F_2 = \int_{\Omega} H(-\phi)(I - c_2)^2 d\Omega$

To compute these intensities, we keep  $\phi$  fixed and minimize Equation 3.29 with respect to  $c_1$  and  $c_2$  to get:

$$c_1(I, \phi) = \frac{\int_{\Omega} I(\mathbf{x})H(\phi(\mathbf{x}))d\mathbf{x}}{\int_{\Omega} H(\phi(\mathbf{x}))d\mathbf{x}} \quad (3.30a)$$

$$c_2(I, \phi) = \frac{\int_{\Omega} I(\mathbf{x})H(-\phi(\mathbf{x}))d\mathbf{x}}{\int_{\Omega} H(-\phi(\mathbf{x}))d\mathbf{x}} \quad (3.30b)$$

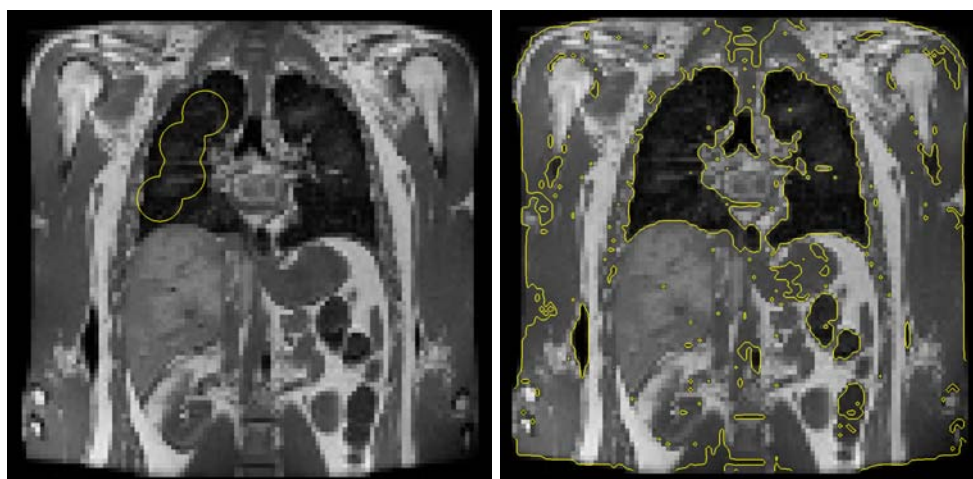
Thus, the interpretation of  $c_1$  and  $c_2$  is simply the *average image intensity* inside and outside the curve. For an intuitive motivation of Equation 3.29, the terms  $F_1 = \int_{\Omega} H(\phi)(I - c_1)^2 d\Omega$  and  $F_2 = \int_{\Omega} H(-\phi)(I - c_2)^2 d\Omega$  can be considered the *total variations* inside and outside the curve. Equation 3.29 is minimized as the variations inside and outside are balanced as depicted in Figure 3.13.

In order to derive the level set equation for solving Equation 3.29, we employ the gradient descent of the corresponding Euler-Lagrange equation [ZCMO96]:

$$\frac{\partial \phi}{\partial t} = \delta(\phi) [\mu K - \nu - \lambda_1(I - c_1)^2 + \lambda_2(I - c_2)^2] \quad (3.31)$$

where  $K$  is mean curvature and  $\delta(z) = dH(z)/dz$  is the Dirac delta function. In the discretization of this equation, Chan and Vese use a Dirac delta function which is non-zero over the entire domain of  $\phi$ , contributing to the global nature of the final segmentation. Figure 3.14 shows an example segmentation using this technique, with  $\mu = \nu = 0$ ,  $\lambda_1 = \lambda_2 = 1$ . Ironically, the strength of producing a global solution is also the main weakness of this method. For example, the segmentation would give the same result as in Figure 3.14(b) *independently* of the initial curve in Figure 3.14(a). In order to successfully segment particular objects the process must be *localized*. We address this issue in Chapter 4.

To conclude, this chapter has presented the previous work starting with the foundation of isosurface extraction. We introduced the important concepts of mesh and interpolant for reconstructing a discrete function and described the contour tree representing the contours of this function. Applications of the contour tree for piecewise continuation and the flexible isosurface interface were presented. Thereafter we described the mathematical foundation and numerical discretization of the level set method. Finally, segmentation *with* and *without* edges based on



(a) Initial curve

(b) Final segmentation

**Figure 3.14:** Images showing the initial curve and the segmented result by “segmentation without edges”

the level set method were presented. The next chapter will describe the combination of these techniques, as the main contribution of this thesis.

---

## Chapter 4

# Combining Topological Structures with Local Segmentation

---

This chapter presents the main contribution of this thesis. We will start by describing the basic idea and motivation for our segmentation pipeline. We will proceed through this pipeline, explaining and motivating each step taken. The first step consists in converting the initialization produced by the flexible isosurfaces into an implicit level set representation. The level set interface will then be subject to a localized version of the segmentation without edges method.

### 4.1 Motivation

As introduced in Section 2.2, there are two principal methods for segmentation. The first of which relies on a distinct isovalue, or a range of values, to represent a particular object. This is commonly manifested using isosurface extraction, explained in Section 3.1. The second method relies on minimizing a functional based on edge information or other variational information. This approach is efficiently handled using the level set method as described in Section 3.4 and Section 3.5.

However, both of these techniques have drawbacks. The boundaries extracted by isosurfaces are seldom perfect due to noise in the data or non-uniform measurements. Level set segmentation techniques are cumbersome to use since they require a careful choice of parameters and a close initial guess for stable convergence. The idea behind the work in this thesis is to *combine the two approaches* for solving these fundamental problems. We will use the isosurface extraction, with the flexible isosurface interface in particular, as the input to the level set segmentation.

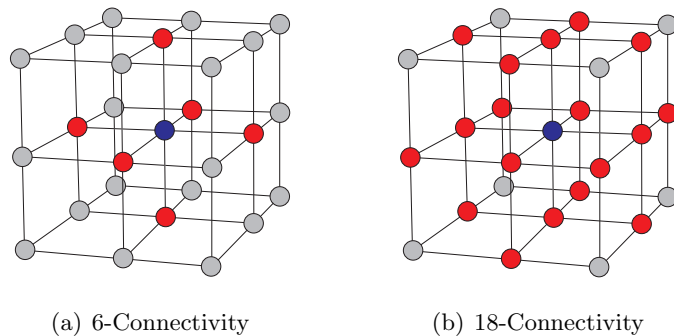
### 4.2 Topological Computation on Discrete Data

The input to our segmentation pipeline is a contour of an isosurface selected by the user. For the work in this thesis we employed the flexible isosurface interface by Carr and Snoeyink [CS03]. This is a tool for exploratory visualization based on the contour tree, allowing the user to pick and evolve individual contours in a powerful way. Once the object targeted for segmentation has been found by its contour, it is selected and converted to a level set representation for

refinement. Depending on the quality of the data, the initial contour may be very far from, or very close to, the actual object being segmented.

This raises a number of questions. The first issue is what interpolating function the contour tree should represent. As discussed in Section 3.2.4, different interpolants define a different set of critical points. For a correct conversion from an isosurface to a level set, the isosurface resulting from a particular interpolant must be representable by the level set function. Interestingly, this is not at all an issue. Recall that a level set surface is in fact an isosurface of the level set function. As such, it must also be extracted using the techniques discussed in Section 3.1. As long as the interpolants used to extract the initial isosurface and the level set surface are the same, the conversion can be exact. However, we have stated that the initial isosurface seldom provides a good final segmentation. The purpose of the level set segmentation is to refine an initial guess. So, the conversion can in reality be approximate, and the choice of interpolant for the topological analysis does not really matter.

However, for practical reasons, it is desirable to compute the contour tree based on the topology of the Marching Cubes cases. Firstly, Marching Cubes itself provides a good balance between accuracy of the isosurfaces and simplicity, compared to simplicial meshes and multilinear interpolants. Secondly, this better match the convention used for level set methods, which structure the level set function in a regular grid with cubic cells. Carr [Car04] has shown that the topology of Marching Cubes matches 6–18 digital image connectivity. Recall that Marching Cubes uses linear interpolation between black (inside) and white (outside) vertices (Figure 3.4). The 6–18 connectivity rule means that the black vertices are connected along 6 edges, while the white vertices are connected along 12 face diagonals in addition to 6 edges, as depicted in Figure 4.1. This rule guarantees that the black vertices inside a surface form a watertight boundary and simplifies the initialization of the level set function as we describe in the next section.

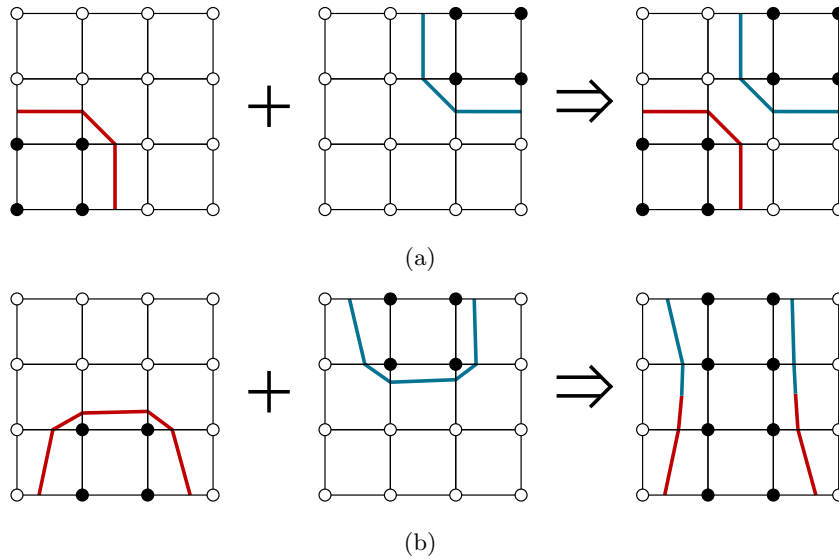


**Figure 4.1:** 6–18 Connectivity. The red vertices are connected with the center blue vertex by different connectivity rules.

### 4.3 From Contours to Initial Level Set

One option for converting the contours to a level set representation is to extract the polygonal mesh of the contours and run a scan conversion algorithm [MBW<sup>+</sup>05]. However, this relatively complex procedure is not necessary since we noted that the level set surface is in fact an isosurface





**Figure 4.2:** Union operation for two contours intersecting the same cell

with isovalue zero. So, to convert any isosurface with isovalue  $h$  to a level set surface we could simply offset the data values by  $h$ , followed by reinitialization as described in Section 3.3.3.

Recall, however, that we are interested in specific contours, or single connected components, of an isosurface. Moreover, the flexible isosurface interface provides the possibility to manipulate and evolve each of these contours individually, so the contours may in fact be described by different isovalues. So we need to approach this problem differently.

We solve this by using the assumption of digital connectivity as described in the previous section. Since the black vertices inside a contour guarantees a watertight boundary, we can simply *add* them to the level set function to initialize the boundary. This operation corresponds to a *union* of the contours. We will motivate this idea using the marching squares topology as depicted in Figure 3.5, which is the 2D analogy of marching cubes. This topology deploys the 4–8 connectivity rule. Imagine that you stand on the inside, at a black vertex. The 4 connectivity rule then states that you can only “leak” to another black vertex along any of the four edges. If you stand at a white vertex, the 8 connectivity rule states that you can “leak” to a white vertex along any diagonal in addition to the edges.

Consider two contours, possibly described by different isovalues, intersecting the same cell. Figure 4.2(a) depicts a situation where the level set function can maintain the union of these contours as in fact two distinct contours. Note that the contours are watertight with respect to the 4–8 connectivity rule. However, we may have the situation depicted in Figure 4.2(b), where the union is not representable by the level set function. This is not really a problem though, since the conversion from contours to level set surface is not required to be perfect. More importantly, we require the conversion to be robust and to produce a watertight boundary which is guaranteed by the union of these black vertices.

In [Car04], Carr has shown how to modify the continuation method of Wyvill *et al.* [WMW86] to extract individual contours using piecewise continuation. We use this approach also for initializing the level set boundary. However, instead of generating geometry for each cell, we

add the black vertices in the cell to the level set function as described above. When this process is completed, we have successfully initialized the boundary, or the grid points immediately adjacent to the level set surface. An iterative algorithm for this is shown in Algorithm 1.

**Require:** Sets of seed cells  $S_{cell}$ , seed edges  $S_{edge}$  and isovalues  $S_h$ , given by the contour tree

**Output:** A level set function with an initialized boundary

```

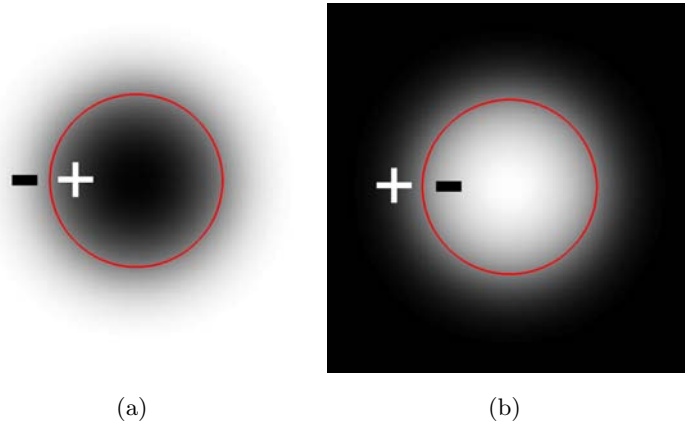
1: Initialize the level set function  $\phi$  with large negative numbers
   {Loop through all contours}
2: for each cell, edge and isovalue in  $S_{cell}$ ,  $S_{edge}$  and  $S_h$  do
3:   Push the edge to the queue  $Q$ 
4:   while  $Q$  is not empty do
5:     Pop an edge from  $Q$ 
6:     Classify the cell and convert the edge to a surface
7:     if the surface is visited then
8:       continue
9:     end if
10:    Mark the surface as visited
        {"Add" vertices to the level set function by a union of the "black" vertices}
11:    for each vertex in the cell do
12:      if  $\phi(\textit{vertex}) \leq 0$  then {the vertex is "white"}
13:        Set  $\phi(\textit{vertex})$  to the value at the vertex, subtracted by the isovalue
14:      end if
15:    end for
        {Follow the surface to neighboring cells}
16:    for each face intersected by the surface do
17:      Push an edge of the face, intersected by the surface, to  $Q$ 
18:    end for
19:  end while
20: end for
21: return  $\phi$ 

```

**Algorithm 1:** INITIALIZELEVELSETBOUNDARY()

The next step in the process is to determine the inside and the outside of the level set surface. The convention used for isosurfaces is that black vertices are “above” and “inside”, whereas the white are “below” and “outside”<sup>1</sup>. One would think that relying on this convention should provide the correct inside and outside for the level set surface. However, there is an ambiguity to this as depicted in Figure 4.3. By convention, the positive side, or the side *above* the isovalue, is *inside*. But, for reasons explained in the next section, we want the target object to be inside the level set surface. Clearly, the signs in Figure 4.3(b) need to be swapped. This is corrected

<sup>1</sup>Recall that we use a negative-inside, positive-outside sign convention for the level set surface. This confuses the convention used in the isosurface community, which uses the exact opposite.



**Figure 4.3:** The inside/outside ambiguity (white is “low”, black is “high”)

by *flood filling* the level set function in order to determine a particular inside/outside situation suitable for the segmentation. For example, this can be performed by starting the flood filling at a point which is known to be outside, which is usually the case for any point at the boundary. The flood filling algorithm using 6–18 connectivity is shown in Algorithm 2.

After flood filling, the final step is to employ a reinitialization algorithm as described in Section 3.3.3 to compute the distances in the full level set function. In this work we use the *fast sweeping method* by Zhao [Zha05] which is efficient and simple to implement.

In practice, medical datasets are noisy, resulting in a very large contour tree. For memory efficiency, it is beneficial to base the contour tree and the flexible isosurfaces on a low resolution version of the dataset. This makes the exploratory visualization smoother and reduces the memory footprint considerably. Thus, a low resolution level set function is initialized using the steps described above. A high resolution level set function is then created by trilinear sampling and used by the level set segmentation method. To conclude, we give the algorithm for the complete initialization process in Algorithm 3.

## 4.4 Localizing Segmentation without Edges

As noted in Section 3.5, the global behavior of the Chan and Vese method is not always desirable. Firstly, it prevents the segmentation of isolated objects chosen by the user. Secondly, since the computation of the averages  $c_1$  and  $c_2$  in Equation 3.31 is global, the solution actually depends on the background, or the embedding, of the object. A simple example is shown in Figure 4.4 where we run the segmentation on a soft object embedded in a black background. The thick line shows the final segmentation, whereas the thin line displays an isoline for reference. We see that different results are produced, simply by varying the size of the embedding space.

In addition to the problems stated above, solving the level set equation in the entire domain is computationally very demanding and simply not feasible for standard medical datasets of today. Thus, in order to use the otherwise beneficial method of Chan and Vese, we need to localize the behavior and the computation.

**Require:** A level set function  $\phi$

**Output:**  $\phi$  with a particular inside/outside situation

```

1: Set the current sign to positive (outside)
2: Add a vertex of  $\phi$  which is known to be on the outside, to the queue  $Q$ 
3: loop {loop until  $\phi$  has been entirely flooded}
4:   while  $Q$  is not empty do {loop until the current region is flooded}
5:     Pop a vertex from  $Q$ 
6:     Mark vertex as visited
7:     if the sign of vertex is not current sign then
8:       Add vertex to the list  $V_{flips}$  for later sign change
9:     end if
10:    if  $\phi(\textit{vertex}) > 0$  then {the vertex is “black”}
11:      The set  $S_{neighbors}$  is the 6 neighbors along the edges
12:    else {the vertex is “white”}
13:      The set  $S_{neighbors}$  is the 18 neighbors along the edges and the face diagonals
14:    end if
15:    for each neighbor in  $S_{neighbors}$  do
16:      if neighbor is not visited and the signs of neighbor and vertex are the same then
17:        Push neighbor to  $Q$ 
18:      end if
19:    end for
20:  end while
21:  if all vertices of  $\phi$  have been visited then
22:    break
23:  end if
24:  Push a non visited vertex of  $\phi$  to  $Q$ 
25:  Find a visited neighbor to vertex
26:  Set current sign to the sign of visited neighbor
27:  if the visited neighbor is not in  $V_{flips}$  then
28:    Flip current sign
29:  end if
30: end loop
31: for each vertex in  $V_{flips}$  do
32:   Flip the sign of vertex
33: end for
34: return  $\phi$ 

```

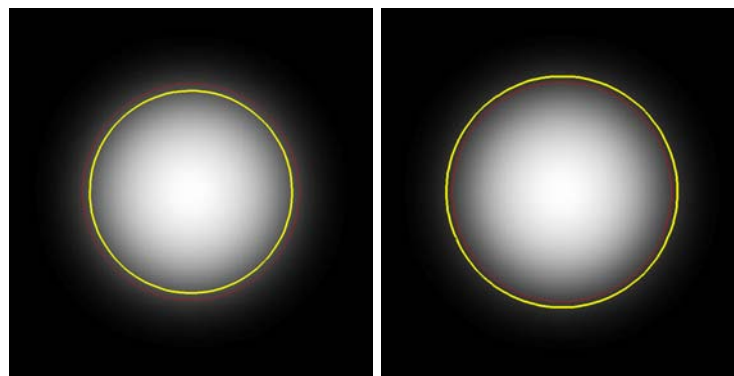
**Algorithm 2:** FLOODFILL()

**Input:** A set of contours given by seed cells  $S_{cell}$ , seed edges  $S_{edge}$  and isovalues  $S_h$

**Output:** A level set function  $\phi$

- 1: Create a low resolution level set function  $\phi_{low}$
- 2: Initialize the boundary of  $\phi_{low}$  by INITIALIZELEVELSETBOUNDARY() (Algorithm 1)
- 3: Flood fill  $\phi_{low}$  by FLOODFILL() (Algorithm 2)
- 4: Compute the distances in  $\phi_{low}$  by Fast Sweeping [Zha05]
- 5: Create a high resolution level set function  $\phi_{high}$
- 6: Compute  $\phi_{high}$  by trilinear sampling in  $\phi_{low}$
- 7: **return**  $\phi_{high}$

**Algorithm 3:** CONVERTFROMCONTOURSTOLEVELSET()

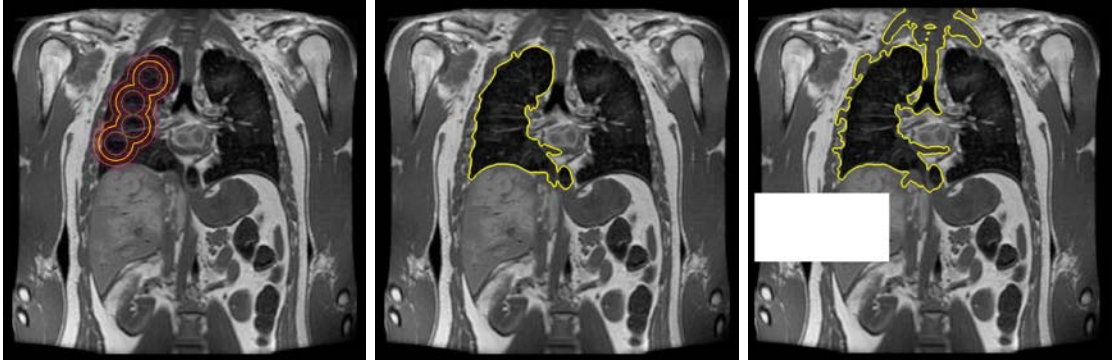


(a) Embedded in 128x128

(b) Embedded in 256x256

**Figure 4.4:** Different solutions depending on the embedding space. The thick line displays the final solution whereas the thin line displays an isoline for reference.

The first obvious solution is to use an existing narrow band scheme as presented in Section 3.3.4, and simply restrict the level set propagation to the narrow band tube. The result is depicted in Figure 4.5 where the actual narrow band is visible in Figure 4.5(a). While the result in Figure 4.5(b) is promising the computation of the averages  $c_1$  and  $c_2$  is still global, making the solution very sensitive to changes in fundamentally unrelated parts of the image. This is seen in Figure 4.5(c) where the insertion of an artificial object drastically changes the resulting segmentation.

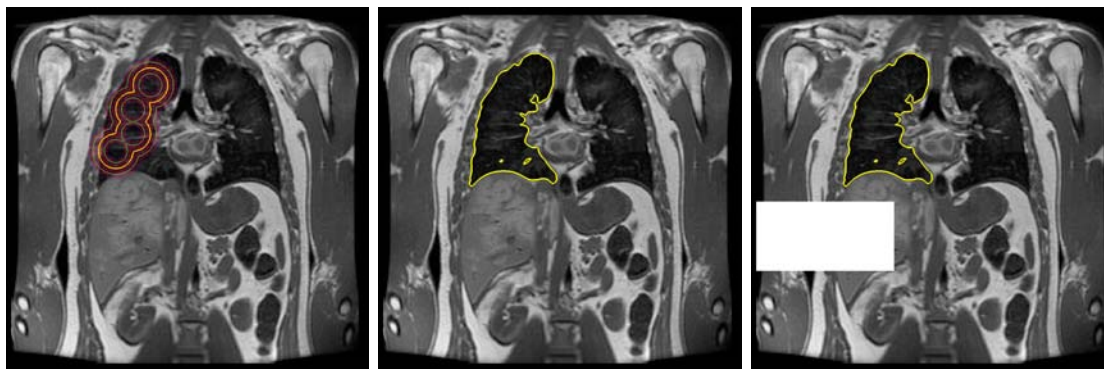


(a) Initial curve and narrow band (b) Solution with original image (c) Solution with artificial object

**Figure 4.5:** Segmentation without edges using narrow band level set propagation. In (c), an artificial object fundamentally unrelated to the lung is inserted, resulting in a drastically different solution.

The reason why Figure 4.5(b) produces such a good result is much due to *numerical diffusion*. Recall the level set equation in Section 3.3.2 corresponding to parabolic diffusion. The geometrical interpretation of this is *smoothing*, or minimization of high curvature spikes. It can be shown that the numerical approximation (discretization) of the level set equation itself introduces numerical diffusion. As the order of the spatial discretization increases, the numerical diffusion decreases. Thus, the level set method has an inherent difficulty of representing sharp edges. For the segmentation, the effect of this is the inability to develop high curvature spikes along the curve. In other words, the curve cannot leak through holes below a certain size, which is why the solution in Figure 4.5(b) is maintained inside the lung.

In order to further localize the computation of  $c_1$  and  $c_2$  we need to localize the computational domain of the segmentation method. In this work, we restrict this domain by using the level set function itself. More precisely, we define the computational domain to be the set  $\Omega_{local} = \{\mathbf{x} \in \mathbb{R}^d : \phi(\mathbf{x}) < \alpha\}$ , where  $\alpha > 0$  defines the area of influence around the level set curve. In other words, we compute the averages only inside the level set and in its immediate neighborhood as specified by  $\alpha$ . This makes sense since we rely on the user to select an appropriate initial contour, and thus computational domain. In addition to restricting the computations, this formulation allows the computational domain to move as the level set deforms. Thus, it can adopt very well to initial surfaces that are very far from the final solution. In practice, using  $\alpha = \gamma$  where  $\gamma$  specifies the narrow band width (Section 3.3.4), amounts to iterating through the narrow band and the inside of the level set. Figure 4.6 shows the same sequence of images using this idea. We see that the segmentation is not sensitive to changes in unrelated parts of the image.



(a) Initial curve and narrow band (b) Solution with original image (c) Solution with artificial object

**Figure 4.6:** Segmentation without edges using local computations. Compared with Figure 4.5(c), the insertion of an object does not change the segmentation of the lung.

This chapter has described the main contributions of this thesis. By using the flexible isosurface interface, the user selects any object by its contour for segmentation. We explained what interpolant to use for the contour tree computation, in order to ensure a simple and watertight conversion from a contour to a level set. We described the implications of this process including the union of the black vertices and the inside/outside ambiguity. When the level set function has been initialized by the fast sweeping method, the segmentation without edges is applied. We confirmed the weakness of the original method and proposed solutions for localizing the behavior. The next chapter will show further results using this approach.

---

## Chapter 5

# Implementation and Results

---

This chapter will briefly outline the implementations for the 2D and the 3D versions used to test and verify our ideas presented in previous chapters. This is followed by some resulting segmentations.

### 5.1 Implementation

Our idea was first tested in a 2D implementation using the narrow band scheme by Peng *et al.* [PMO<sup>+</sup>99]. This allowed us to thoroughly test and evaluate the setup, while having complete overview of the full image. A task which is not equally simple in 3D. Actually, the major part of the work was spent in 2D to avoid the additional complexity of visualizing and debugging 3D output.

For the 3D version we used a reference implementation of the DT-Grid [NM06] by the courtesy of Michael Bang Nielsen. This is a very efficiently implemented and complex datastructure and it was out of the scope of this thesis to implement from scratch.

The contour tree algorithms and the piecewise continuation for Marching Cubes were also reference implementations provided by the courtesy of Hamish Carr. This implementation assumes a 3D dataset input, but is general enough for computing the topology of 2D images as well. During the limited time span of this thesis, we did not fully incorporate the flexible isosurfaces interface into the segmentation interface, but we consider this future work.

The codebase is written in C++ verified to compile using the GNU C++ compiler and Visual C++. The GLUT and OpenGL APIs are used for the visualization which runs on any standard graphics card.

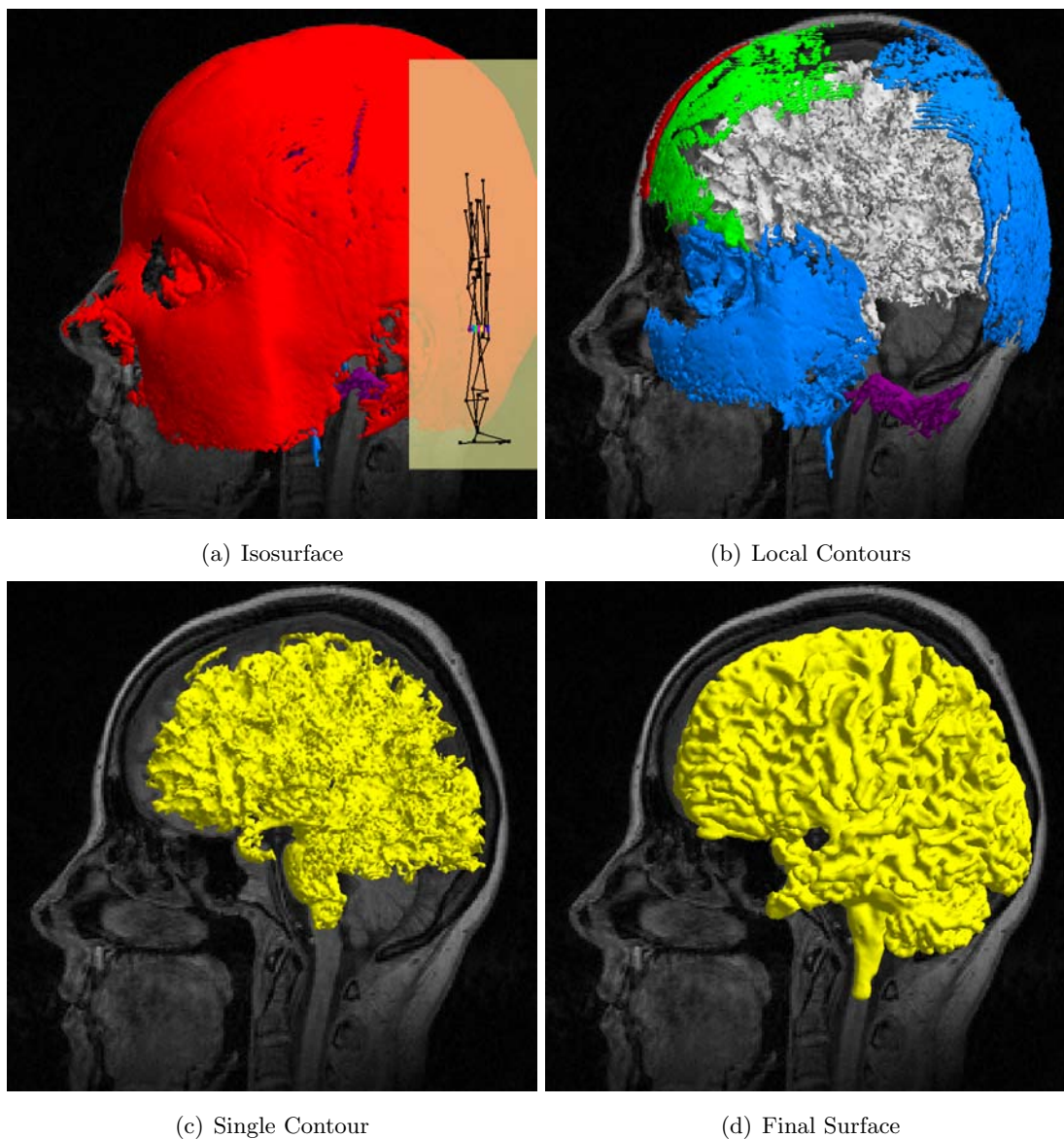
### 5.2 Results

We show two sample segmentations in Figure 5.1 and Figure 5.2. For Figure 5.1 we used the “UNC head” MRI dataset and computed the contour tree on the full resolution of (256x256x109). In (a) we see the problem of occlusion when using a regular isosurface. In (b) we use the contour tree to extract so called *local contours* which represent peaks in the dataset. We pick the brain

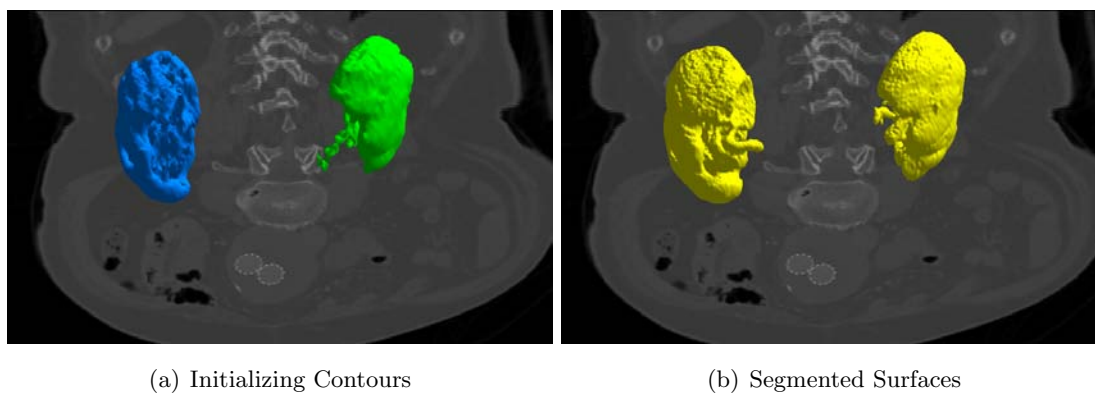


as the target for segmentation as shown in (c). This segmentation is not perfect however, so it is refined using the level set technique to give the final result in (d).

In Figure 5.2 we used the “stent” dataset (512x512x174) from [www.volvis.org](http://www.volvis.org). For this dataset it is beneficial to compute the contour tree on a lower resolution of (256x256x87) for two reasons. Firstly, the memory footprint of the computation is very large, and unmanageable for most standard workstations. Secondly, exploring the dataset using the flexible isosurfaces are much smoother using the lower resolution. When the contours in (a) had been selected, the level set segmentation refined this using the high resolution dataset to give the final segmentation in (c).



**Figure 5.1:** Interactive Brain Segmentation. In (a), an isosurface has been chosen, but the skull surface occludes the brain. In (b), *local contours* are used to extract contours around the largest “peaks” (by volume) in the dataset. In (c), one of these contours has been selected visually as the initialization surface for level set segmentation. This contour fails to segment the brain properly since at lower isovalues a spurious connection is formed with the skull fragments. In (d), the final level set segmentation is shown. Using a level set segmentation instead of the single-isovalue contour of (c) allows us to produce a better boundary of the actual brain, in this case the grey matter.



**Figure 5.2:** Kidney Segmentation. As in Figure 5.1, individual contours were selected interactively (a) to initialize the level set segmentation (b). Here, the level set segmentation is able to cope with internal variation in the isovalue of the kidneys to produce a better result than contours alone.

---

## Chapter 6

# Conclusions and Future Work

---

### 6.1 Conclusions

In this thesis we have described two principal methods used for segmentation - isosurface extraction and level set methods. We pointed out that each of these have drawbacks. Isosurfaces rely on a particular isovalue, while level set segmentation methods require a close initial guess for stable convergence.

We explained how these techniques can be combined to remedy each others weaknesses by using the flexible isosurfaces as an input to the level set segmentation. In our work we used a level set method *without* edges, although a method *with* edges could just as easily be deployed, depending on the characteristics of the input data.

We further showed that using a lower resolution dataset for the flexible isosurfaces is beneficial for two reasons. Firstly, it shortens the computation time and the large memory footprint of the contour tree. Secondly, it makes the exploratory visualization smoother, providing a better experience for the user. The first initial segmentation is later refined using level set methods based on the high resolution data.

### 6.2 Future Work

In the future, we would like to further investigate later work by Chan and Vese and see if this can be successfully applied to medical data. Furthermore, we want to combine the segmentation with and without edges in a general functional, giving the user better control over the segmentation process.

We would also like to improve the interface by bringing in the full idea of flexible isosurfaces. Since the rendering of the contour tree itself is non trivial, we want to study different solutions to this for an optimal and intuitive interface.

The output can be further improved by incorporating direct volume rendering. Using the final segmentation, the transfer functions for volume rendering can be used to provide a better view of the interior of the segmented object.

We want to study *query driven* segmentation. The contour tree can be augmented with various properties of each topological subregion in the data. In the future we would like to fetch the

contours matching a certain query, such as volume or mass, and run the segmentation in an automatic manner.

We would like to explore methods for reducing the memory footprint of the contour tree, since this is very large for noisy medical datasets. We also want to further investigate whether topological analysis can be performed for specific level set methods, allowing visual exploration of possible segmentations.

---

# Bibliography

---

- [AS95] David Adalsteinsson and James A. Sethian, *A fast level set method for propagating interfaces*, Journal of Computational Physics **118** (1995), no. 2, 269–277.
- [Ben75] Jon Louis Bentley, *Multidimensional binary search trees used for associative searching*, Commun. ACM **18** (1975), no. 9, 509–517.
- [BR63] Roger L. Boyell and H. Ruston, *Hybrid techniques for real-time radar simulation*, Proceedings of the 1963 Fall Joint Computer Conference, IEEE, November 1963, pp. 445–458.
- [Can86] J Canny, *A computational approach to edge detection*, IEEE Transactions on Pattern Analysis and Machine Intelligence **8** (1986), no. 6, 679–698.
- [Car04] Hamish Carr, *Topological manipulation of isosurfaces*, Ph.D. thesis, The University of British Columbia, April 2004.
- [CCCD93] Vincent Caselles, Francine Catté, Tomeu Coll, and Françoise Dibos, *A geometric model for active contours in image processing*, Numerische Mathematik **66** (1993), no. 1, 1–31.
- [CFL28] R. Courant, K. Friedrichs, and H. Lewy, *Über die partiellen differenzgleichungen der mathematischen physik*, Mathematische Annalen **100** (1928), 32–74.
- [CKS97] Vincent Caselles, Ron Kimmel, and Guillermo Sapiro, *Geodesic active contours*, International Journal of Computer Vision **22** (1997), no. 1, 61–79.
- [CMM<sup>+</sup>97] Paolo Cignoni, Paola Marino, Claudio Montani, Enrico Puppo, and Roberto Scopigno, *Speeding up isosurface extraction using interval trees*, IEEE Transactions on Visualization and Computer Graphics **3** (1997), no. 2, 158–170.
- [CMS06] Hamish Carr, Torsten Möller, and Jack Snoeyink, *Artifacts caused by simplicial subdivision*, IEEE Transactions on Visualization and Computer Graphics **12** (2006), no. 2, 231–242.
- [CS03] Hamish Carr and Jack Snoeyink, *Path seeds and flexible isosurfaces using topology for exploratory visualization*, VISSYM '03: Proceedings of the symposium on Data visualisation 2003 (Aire-la-Ville, Switzerland, Switzerland), Eurographics Association, 2003, pp. 49–58.
- [CSA03] Hamish Carr, Jack Snoeyink, and Ulrike Axen, *Computing contour trees in all dimensions*, Computational Geometry **24** (2003), no. 2, 75–94.

- [CV01] Tony Chan and Luminita Vese, *Active contours without edges*, IEEE Transactions on Image Processing **10** (2001), no. 2, 266–277.
- [Dur88] Martin J. Durst, *Re: Additional reference to "marching cubes"*, SIGGRAPH Comput. Graph. **22** (1988), no. 5, 243.
- [Ede80] Herbert Edelsbrunner, *Dynamic data structures for orthogonal intersection queries*, Tech. report, Inst. Informationsverarb, Tech. Uniz. Graz, Graz, Austria, 1980.
- [IK94] Takayuki Itoh and Koji Koyamada, *Isosurface generation by using extrema graphs*, VIS '94: Proceedings of the conference on Visualization '94 (Los Alamitos, CA, USA), IEEE Computer Society Press, 1994, pp. 77–83.
- [KWT88] Michael Kass, Andrew Witkin, and Demetri Terzopoulos, *Snakes: Active contour models*, International Journal of Computer Vision **1** (1988), no. 4, 321–331.
- [LC87] William E. Lorensen and Harvey E. Cline, *Marching cubes: A high resolution 3d surface construction algorithm*, SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques (New York, NY, USA), ACM Press, 1987, pp. 163–169.
- [LOC94] Xu-Dong Liu, Stanley Osher, and Tony Chan, *Weighted essentially non-oscillatory schemes*, Journal of Computational Physics **115** (1994), 200–212.
- [LSJ96] Yarden Livnat, Han-Wei Shen, and Christopher R. Johnson, *A near optimal isosurface extraction algorithm using the span space*, IEEE Transactions on Visualization and Computer Graphics **2** (1996), no. 1, 73–84.
- [MBW<sup>+</sup>05] Ken Museth, David E. Breen, Ross T. Whitaker, Sean Mauch, and David Johnson, *Algorithms for interactive editing of level set models*, The International Journal of Eurographics Assoc.: Computer Graphics Forum **24** (2005), no. 4, 821–841.
- [MBZW02] Ken Museth, David E. Breen, Leonid Zhukov, and Ross T. Whitaker, *Level set segmentation from multiple non-uniform volume datasets*, VIS '02: Proceedings of the conference on Visualization '02 (Washington, DC, USA), IEEE Computer Society, 2002, pp. 179–186.
- [MS89] D. Mumford and J. Shah, *Optimal approximation by piecewise smooth functions and associated variational problems*, Commun. Pure Applied Mathematics **42** (1989), 577–685.
- [MSS94] Claudio Montani, Riccardo Scateni, and Roberto Scopigno, *A modified look-up table for implicit disambiguation of marching cubes*, The Visual Computer **10** (1994), no. 6, 353–355.
- [NM06] Michael B. Nielsen and Ken Museth, *Dynamic tubular grid: An efficient data structure and algorithms for high resolution level sets*, Journal of Scientific Computing (2006), 1–39.

- [OF03] Stanley Osher and Ronald Fedkiw, *Level set and dynamic implicit surfaces*, Springer-Verlag New York Inc., 2003.
- [OS88] Stanley Osher and James A. Sethian, *Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations*, *Journal of Computational Physics* **79** (1988), 12–49.
- [OS91] Stanley Osher and Chi-Wang Shu, *High-order essentially nonscillatory schemes for hamilton-jacobi equations*, *SIAM Journal on Numerical Analysis* **28** (1991), no. 4, 907–922.
- [PCM03] Valerio Pascucci and K Cole-McLaughlin, *Parallel computation of the topology of level sets*, *Algorithmica* **38** (2003), no. 2, 249–268.
- [PMO<sup>+</sup>99] Danping Peng, Barry Merriman, Stanley Osher, Hong-Kai Zhao, and Myungjoo Kang, *A pde-based fast local level set method*, *Journal of Computational Physics* **155** (1999), no. 2, 410–438.
- [RT92] Elisabeth Rouy and Agnes Tourin, *A viscosity solutions approach to shape-from-shading*, *SIAM Journal on Numerical Analysis* **29** (1992), no. 3, 867–884.
- [Set96] J. Sethian, *A fast marching level set method for monotonically advancing fronts*, *Proceedings of the National Academy of Science*, vol. 93, 1996, pp. 1591–1595.
- [SO88] Chi-Wang Shu and Stanley Osher, *Efficient implementation of essentially non-oscillatory shock-capturing schemes*, *Journal of Computational Physics* **77** (1988), no. 2, 439–471.
- [SSO94] Mark Sussman, Peter Smereka, and Stanley Osher, *A level set approach for computing solutions to incompressible two-phase flow*, *Journal of Computational Physics* **114** (1994), no. 1, 146–159.
- [Str89] John C. Strikwerda, *Finite difference schemes and partial differential equations*, Wadsworth Publ. Co., Belmont, CA, USA, 1989.
- [TIS<sup>+</sup>95] Shigeo Takahashi, Tetsuya Ikeda, Yoshihisa Shinagawa, Tosiyasu L. Kunii, and Minoru Ueda, *Algorithms for extracting correct critical points and constructing topological graphs from discrete geographical elevation data*, *Computer Graphics Forum* **14** (1995), no. 3, 181–192.
- [TTF04] Shigeo Takahashi, Yuriko Takeshima, and Issei Fujishiro, *Topological volume skeletonization and its application to transfer function design*, *Graph. Models* **66** (2004), no. 1, 24–49.
- [vKvOB<sup>+</sup>97] Marc J. van Kreveld, Rene van Oostrum, Chandrajit L. Bajaj, Valerio Pascucci, and Daniel Schikore, *Contour trees and small seed sets for isosurface traversal*, *Symposium on Computational Geometry*, 1997, pp. 212–220.



- 
- [WMW86] Geoff Wyvill, Craig McPheeters, and Brian Wyvill, *Data structure for soft objects*, *The Visual Computer* **2** (1986), no. 4, 227–234.
- [ZCMO96] Hong-Kai Zhao, Tony Chan, Barry Merriman, and Stanley Osher, *A variational level set approach to multiphase motion*, *Journal of Computational Physics* **127** (1996), no. 1, 179–195.
- [Zha05] Hong-Kai Zhao, *A fast sweeping method for eikonal equations*, *Mathematics of Computation* (2005), no. 74, 603–627.