TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABREVIATIONS

ANOVA	Analysis of Variance
ATS	ANOVA-Type Statistic
CT	Computed Tomography
DSL	Detail Significance Layers
FastDRaW	Fast Delineation by Random Walker
FBS	Foreground-Background Seeding
FN	False Negative
FP	False Positive
GC	Graph Cuts
GMM	Gaussian Mixture Model
GPU	Graphics Processing Unit
HCI	Human Computer Interaction
IS	Intelligent Scissors
M+FBS	Foreground-Background Seeding using a Mouse
M+RCD	Rough Contour Drawing using a Mouse
Mpixels	Mega pixels
MR	Magnetic Resonance
NNM	Nearest Neighbourhood Map
P+FBS	Foreground-Background Seeding using a Tablet and a Pen

P+RCD	Rough Contour Drawing using a Tablet and a Pen
RCD	Rough Contour Drawing
ROI	Region Of Interest
RUR	Rapid Update Rate
RW	Random Walker
SLIC	Simple Linear Iterative Clustering
SUR	Slow Update Rate
TCIA	The Cancer Imaging Archive
TN	True Negative
TP	True Positive
US	Ultrasound

INTRODUCTION

Image segmentation consists in extracting the boundaries of objects of interest from a given image. Due to its involvement as a pre-processing step in many computer vision problems, image segmentation is of major interest in research and industry. Particularly in medical and biomedical image analysis, segmentation plays a fundamental role. Emerging applications, for example tumour measurement and growth follow up (Mi *et al.*, 2015), 3D reconstruction in orthopaedic surgery (Cevidanes *et al.*, 2005) or retinal image analysis (Chen *et al.*, 2015), typically require a segmentation step to separate and identify structures present on the image. Many methods proposed in the literature tackle the problem from a practical point of view, taking advantage of the specific context of the given application. For example, one can assume a particular geometry (e.g., shape, location and structural anatomy of the heart) or exploit particular properties of the image modality (e.g., speckle characteristics in ultrasound images or bone absorption in x-ray images).

The problem is that such specific tools perform poorly when the context of the application varies, inducing significant additional costs to develop or adapt the existing tools for new applications. For general purpose image segmentation, in which we have little or no prior information about the application, the variability of the context is managed by the *user*. In fact, the segmentation task can be achieved with varying degrees of user involvement, on a continuum from fully manual to fully automated. Despite the fact that manual delineation of the object boundary is tedious, time consuming and subject to large inter-operator variability (Moltz *et al.*, 2011), it is still considered the gold standard for performing segmentation in medical applications. This is because the user has full control over the segmentation process, thereby ensuring satisfactory results regardless of the application context. On the other hand, fully automated methods provide fast and repeatable segmentation results, but are prone to failure, limiting their applicability in complex scenarios. A compromise between these extremes is to *interactively*

assist the user during the segmentation to reduce the user's workload and reduce variability in the results, while allowing the user to supervise and correct occasional segmentation failures.

Because of their flexibility to represent different types of images, graph theoretical methods have been successfully applied to interactive image segmentation (Mortensen & Barrett, 1998; Boykov & Jolly, 2001; Li *et al.*, 2004; Rother *et al.*, 2004; Grady, 2006; Honnorat *et al.*, 2015). Such approaches show a great capability to adapt to a wide range of applications with varying interaction mechanisms. For reasonably sized images, these approaches offer convenient interaction between the user and the computer. However, the response time increases with image size, rendering this communication ineffective for large images. This delay in the response time decreases the user performance. Still, for interactive segmentation approaches, it is the user who judges when the result is satisfactory, making the segmentation process subject to human factors. The impact of these human factors depends on the degree of involvement of the user during the segmentation task. While common approaches in the literature emphasize the computational aspects of segmentation, the present research investigates factors that affect the *user's performance* during an interactive segmentation task. To the best of our knowledge, there has been no progress investigating the user performance in the context of image segmentation since the seminal work of Olabarriaga & Smeulders (2001). In this thesis, the two following questions are explored: (i) from a computational point of view, how can the user's input be fully leveraged to improve the efficiency of existing graph-based segmentation approaches to reduce the response time? and (ii) in what way does the response time affect the user performance during a segmentation task? To the best of our knowledge, this is the first study that involves the analysis of the user performance according to different response time conditions during an interactive segmentation task.

The goal of this thesis is to gain a better understanding of the user performance during an interactive segmentation task. This information is leveraged to improve the design of segmentation

algorithms. The challenge is that the user behaviour is highly variable and mostly depends on the segmentation approach used. For concreteness, we focus our work on the *scribble-based paradigm*, a popular interaction mechanism which has been applied to a wide range of segmentation approaches (Figure 0.1). Briefly described, in scribble-based segmentation, the user draws labels in the form of scribbles directly on the image. In the case of single object segmentation, foreground labels are drawn inside the object and background labels are drawn on the outside. In response, the algorithm recomputes and displays the segmentation results.

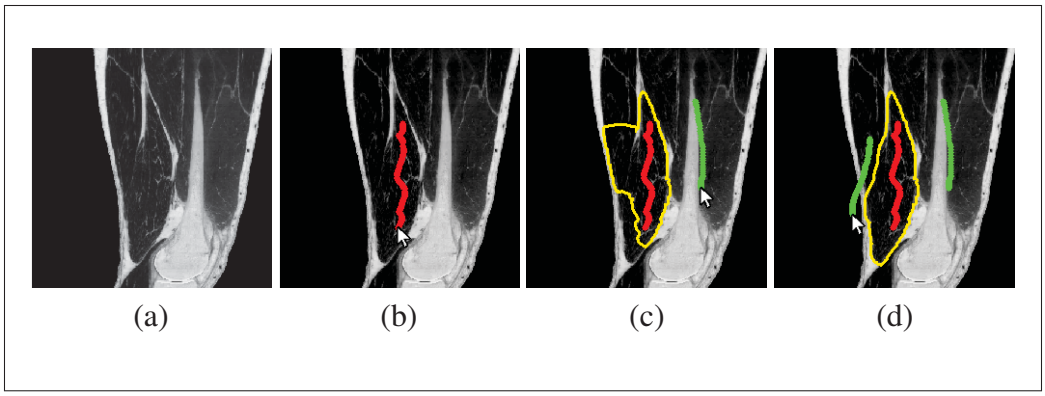

Figure 0.1 Example of scribble-based interactive segmentation: (a) the original image; (b-d) the user-drawn foreground (red) and background (green) labels yielding the computed segmentation (yellow)

In this thesis, we show how the characterization of user input and performance in scribble-based mechanisms can be exploited to improve the segmentation process. The specific objectives were:

- To investigate the relevance of the actions performed by the user during the segmentation task in order to design an interaction mechanism allowing efficient computations;
- To improve the efficiency of the segmentation task by designing an automated algorithm which takes advantage of the relevance of the user's actions;
- To investigate the impacts of computational improvements on the user performance.

This thesis is organized into five chapters. Chapter 1 is an introduction to image segmentation. It contains a survey of the state-of-the-art in graph-based interactive segmentation approaches and describes the details of the segmentation algorithms that we worked with in this thesis.

The following three chapters address each of the specific objectives. The literature related to each specific topic is reviewed in the corresponding chapter.

Contribution 1: relevance of the user actions

The first contribution, presented in Chapter 2, concerns the improvement of segmentation efficiency in terms of computation time. This is important because for heavy computations, the effectiveness of the communication between the user and the algorithm deteriorates. To reduce the computational load, one can reduce the space within which the object boundary is being searched for. However, existing approaches for search space reduction in image segmentation, either downsample the image resolution, inducing a quality loss in the segmentation results (Achanta *et al.*, 2012; Levinshtein *et al.*, 2009; Mori, 2005), or are highly method- and hardware-specific, restricting their usability (Grady & Sinop, 2008; Grady *et al.*, 2005; Delong & Boykov, 2008; Andrews *et al.*, 2010). We propose to exploit user interaction to discard pixels which are of low relevance to the segmentation process. Our hypothesis is that *only the pixels located near the object boundary are needed to achieve a satisfactory segmentation*. Therefore, we can ignore pixels lying far from the object boundary. Based on this assumption, we design a fast segmentation approach that uses an additional user interaction step to locate regions near the object boundary. Then, the search space is reduced to perform computations solely on these selected regions. Finally, we conduct a user study to verify the hypothesis. This contribution has two impacts: (i) to provide an efficient segmentation approach that is applicable to any scribble-based algorithm; and (ii) to gain knowledge about pixel relevance to a segmentation process.

This work has resulted in the following peer-reviewed publications:

- Houssem-Eddine Gueziri, Michael J. McGuffin and Catherine Laporte, "*A Generalized Graph Reduction Framework for Interactive Segmentation of Large Images*", Computer Vision and Image Understanding, Vol. 150, pp. 44-57, (2016);
- Houssem-Eddine Gueziri, Michael J. McGuffin and Catherine Laporte, "*User-guided graph reduction for fast image segmentation*", IEEE International Conference on Image Processing, pp. 286-290, (2015).

Contribution 2: improvement of the segmentation efficiency

In the second contribution (Chapter 3), we propose the Fast Delineation by Random Walker algorithm (FastDRaW), an extension of the approach described in Chapter 2 that automatically locates the useful regions on the image, based on the regular scribbles drawn by the user. In this case, no additional interaction mechanism is required on the part of the user. Our assumption is that the object boundary is more likely to be located somewhere between two labels of different categories (e.g., between the foreground and background labels). We hypothesize that, under this assumption, *the automated reduction should not affect segmentation accuracy*. The computations are further reduced, allowing real-time segmentation. As a result, the computational part of the segmentation process provides a very fast response to the user. However, the delay of the response influences the user performance, which motivates our third contribution.

The algorithm was made open-source and publicly available on GitHub[1]. This work has been published in a peer-reviewed conference paper:

[1] `http://github.com/hgueziri/FastDRaW-Segmentation`

- Houssem-Eddine Gueziri, Lina Lakhdar, Michael J. McGuffin and Catherine Laporte, "*Fast-DRaW – Fast Delineation by Random Walker: application to large images*", MICCAI Workshop on Interactive Medical Image Computing, Athens, Greece, (2016).

Contribution 3: analysis of the user performance

The third contribution of this project, presented in Chapter 4, is to assess the impact of the response time on the user performance during a segmentation task. Our hypothesis is that *there exists a mutual influence between the user performance and the segmentation performance*. In order to investigate the extent of this influence, we conduct a user study that manipulates the delay of response, i.e., feedback latency, provided by our fast segmentation algorithm (developed in Chapter 3). In this experiment, users achieve the segmentation task under different latency conditions. As a result, we characterize the user performance according to different ranges of latencies. The goal is to provide guidelines on how to design effective interaction mechanisms according to the computational efficiency of the segmentation approach.

This work is described in a manuscript currently under review:

- Houssem-Eddine Gueziri, Michael J. McGuffin and Catherine Laporte, "*Latency management in interactive medical image segmentation: guidelines for effective system design*", 10 pages, submitted to IEEE Transactions on Biomedical Engineering. (April 2017) (paper under revision)

The three contributions are related and guided by the analysis of user performance and behaviour during the segmentation task. While existing segmentation approaches emphasize the computational part of the segmentation to improve the task, this research considers the user as an integral part of the segmentation process. Our contributions are more focused on the char-

acterization of the user performance, making the provided improvements adaptable to a variety of interactive segmentation approaches.

CHAPTER 1

BACKGROUND

1.1 Image segmentation

Image segmentation is the process of delineating regions of interest on a given image. Generally, these regions are semantically meaningful and are of particular relevance to a given application. In medical applications, they often represent anatomical structures, tissues with specific properties or target organs. The outcome of segmentation is a labelled image in which pixels are classified into discrete categories, or, equivalently, a list of points located on the boundaries of the different regions of interest. Yet, image segmentation is not an end in itself and is often considered as a pre-processing step. In this case, further processing is applied to the extracted regions to obtain comprehensive information, such as computing the size of the segmented object (Maksimovic *et al.*, 2000), analyzing pathological tissues (Comaniciu *et al.*, 1999; Piqueras *et al.*, 2015) or rendering 3D reconstructions of the organs (Cevidanes *et al.*, 2005).

In an image segmentation problem, it is common to assume that pixels from a single tissue/organ share similar physical properties, making them appear alike on the image. In computer vision, local image properties are called features and express information about the image data, such as pixel intensity, gradient magnitude or texture homogeneity. Based on prior information about the expected segmentation results, a *model* can be used to describe the relationship between image features and a segmented label category, i.e., what makes a pixel more likely to belong to a given category (segmented region). However, for most applications, this is not sufficient. Therefore, regularization constraints are added to the model. For example, pixels from the same label category should satisfy a given homogeneity (smoothness) criterion or a particular shape constraint. Therefore, image segmentation can be related to defining and fitting a representative model which expresses application-specific requirements. Yet, this task is not trivial. Even with a good model, the context of the application may change, causing seg-

mentation failures. For example, in the case of automatic prostate segmentation, in which the context of the application is restrictively targeted, one of the best performance recorded on the MICCAI PROMISE12 challenge database (Litjens *et al.*, 2014) yielded an accuracy score of $86.65\% \pm 4.35$ (Yu *et al.*, 2017). Although the approach achieved a remarkable score, critical applications, e.g., radiotherapy planning would require further expert verification and manual corrections.

Image segmentation is naturally ill-posed and challenging (Peng *et al.*, 2013). Many approaches have been investigated and proposed in the literature. The goal of this chapter is not to do an exhaustive review of the literature of all existing approaches. Rather, we refer interested readers to the following surveys that address specific topics: using deformable models (McInerney & Terzopoulos, 1996), using unsupervised methods (Zhang *et al.*, 2008), applied to ultrasound images (Noble & Boukerroui, 2006) or applied to color images (Lucchesseyz & Mitray, 2001). For a given application, the choice of the segmentation approach depends on the nature of the task to achieve, the type of the images used, the properties of the structure to segment and other information characterizing the context of the application. In this thesis, we are interested in general purpose image segmentation tasks, i.e., when no prior information about the context of the application is known. In this case, instead of assuming any prior information, the context variability is managed by the user. Therefore, during the segmentation task, the user interactively guides the segmentation towards the desired results. These approaches are known as *interactive image segmentation* methods and require the use of an efficient communication mechanism between the user and the segmentation algorithm.

1.2 Segmentation as an interactive task

The interactive segmentation task can be described as a three-block process (see Figure 1.1). The first block is the *interactive block*. It allows bilateral communication between the user and the computer through human-computer interaction (HCI) mechanisms; i.e., it defines the method and the devices used to feed parameters to the algorithm. The inputs/outputs are in a readable format for the user, e.g., numerical values or graphical contours. The second block is

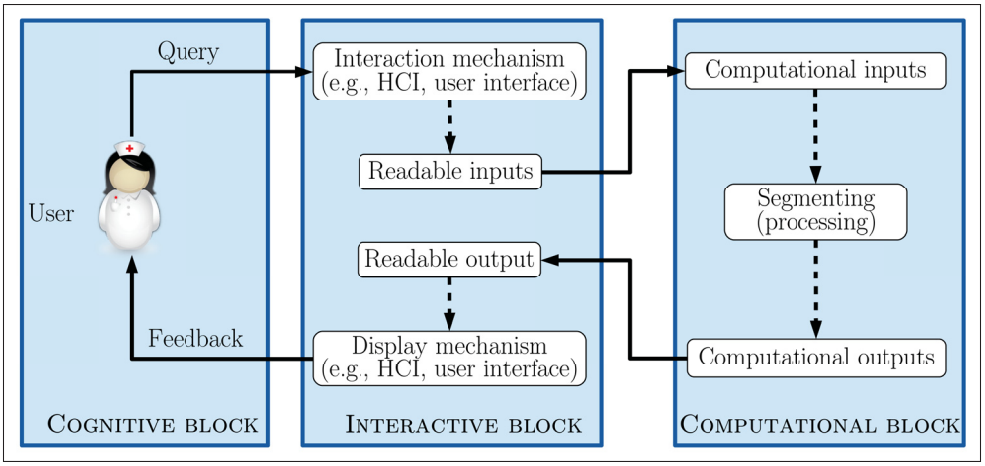

Figure 1.1 Interactive segmentation process: The interactive block contains inputs in understandable format, e.g., drawing. The computational block contains inputs suitable to the segmentation algorithm. The cognitive block involves the user's interpretation of the results and the thinking process. Solid arrows indicate inter-block processes and dashed arrows indicate intra-block processes

the *computational block*. It is in charge of finding the object boundary using a given algorithm. At this step, the inputs/outputs are translated into parameters readable by the algorithm. Once the segmentation results are obtained, they are displayed to the user in a readable format. This leads to the third block, the *cognitive block*, in which the user interprets the results. If they are not satisfactory, the user updates/modifies the inputs and the three blocks are reiterated, thereby creating a feedback loop between the user and the segmentation algorithm.

Based on this, a straightforward interaction approach for image segmentation would be to consider a *trial and error* procedure. In this case, the user provides the input parameters at the beginning of the segmentation process and the results are obtained at the end of the computation. If the results are not satisfactory, the user adjusts the parameters and runs the segmentation again. Here, no intermediate results are recorded. The relationship between two successive iterations is solely based on the knowledge the user has gained from the previous trials. Due to the minimal involvement of the user during the segmentation, this type of approach can be referred to as *semi-automated*. An example of such approaches is the active contour segmentation algorithm (Kass *et al.*, 1988), in which the user specifies the positions of an initial

contour that iteratively converges towards the object boundary. The user can be substituted using learning algorithms to exploit failures from previous trials, e.g., in deep convolutional neural network segmentation (Long *et al.*, 2015).

In this thesis, *interactive segmentation* methods refer to approaches where the intermediate results are displayed. For example, segmentation approaches that can be found in the ITK-SNAP[1] software. These require more involvement on part of the user. Here, the results of the previous iteration, e.g., the last position of the contour obtained, are injected into the next iteration with additional information provided by the user. To be efficient, this type of approach requires a more sophisticated interaction mechanism than semi-automated and automated approaches. Olabarriaga & Smeulders (2001) described three types of interaction mechanisms that can be used in segmentation tasks:

- **Setting parameter values:** requires the user to manipulate numerical parameters during the segmentation.
- **Choosing from a menu:** the information is selected from a pre-defined set of possible actions, usually categorized in a menu.
- **Pictorial input on the image grid:** the information is *directly* introduced on the image grid.

The first two interaction mechanisms are often used with automated and semi-automated approaches, while the third one is more suitable for interactive approaches. This is because it reduces the abstraction layer between the interpretation of the input data and the results, rendering the interaction mechanism more intuitive.

Interactive approaches are, by nature, considerably influenced by user performance. This project aims at understanding the variables that influence the performance of the user during interactive segmentation tasks, and how the user input can, in turn, be exploited to influence

[1] `http://www.itksnap.org/pmwiki/pmwiki.php?n=Main.HomePage`

interactive segmentation algorithms in more effective ways. Throughout this thesis, we design approaches for interactive segmentation improvements which consider the user performance as part of the process. Because of user variability, it is not trivial to characterize the user behaviour. Moreover, there exists a wide variety of valid interaction mechanisms that can be used in interactive image segmentation. It is therefore important to categorize the approaches according to their interaction mechanisms. This is discussed in Section 1.3. Subsequently, in Section 1.4, we review graph-based methods, which are most often used in recent interactive image segmentation approaches. To understand how the information provided by the user can be exploited in the segmentation process, we define, in Section 1.5, the computational principle behind three popular graph-based segmentation methods that have been experimented in this thesis.

1.3 Interactive mechanisms for image segmentation

Most of the image segmentation interaction mechanisms can be classified into two categories: a *contour-based* mechanism, in which the user focuses on tracing the object boundary and a *region-based* mechanism, in which the user focuses on finding pixels belonging to the object (see Figure 1.2).

1.3.1 Contour-based mechanism

Under the contour-based paradigm, the segmentation problem is defined as "finding the *boundaries* of a particular object". Contour-based approaches often rely on image gradient features as cues to guide the search towards the object boundary. Active contours (Kass *et al.*, 1988; Caselles *et al.*, 1997; Chan & Vese, 2001; Wang *et al.*, 2014) are popular algorithms which use a contour-based approach. The original algorithm (Kass *et al.*, 1988) consists in initializing, often manually, a set of points that forms a contour. The configuration of these points determines their global *energy* composed of internal and external energy terms. The internal energy is a regularization term and is defined by the spatial configuration of the points (e.g., curvature and spacing between the points), such that curved configurations would have higher energy. The

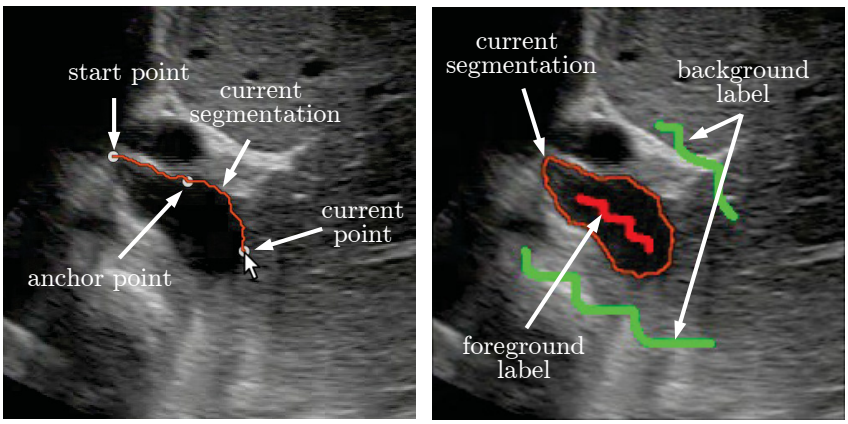

Figure 1.2 Example of two popular interaction mechanisms for image segmentation: (a) the live wire contour-based approach, and (b) the scribble region-based approach

external energy is driven by the content of the image (e.g., intensity and gradient), such that points lying on edges of high gradient magnitude would have low energy. The algorithm moves the points iteratively and computes the energy of the contour. The goal is to find the position of the points that minimizes the global energy. Active contour approaches show robustness in noisy images, such as ultrasound images (Faisal *et al.*, 2015). Moreover, they compensate for missing boundaries by assuming continuity between the points. Variants of active contours have been proposed to embed prior information about the shape of the object Cootes *et al.* (1995) and its appearance Cootes *et al.* (2001). However, the interaction mechanism is limited to the initialization of the points at the beginning of the segmentation.

Another popular interactive mechanism based on object contours is the *live wire paradigm* (Mortensen & Barrett, 1998; Falcão *et al.*, 1998, 2000; Miranda *et al.*, 2012; Mishra *et al.*, 2008) (also known as the magnetic lasso tool). During the segmentation task, the live wire algorithm assists the user in *tracing* the contour of the object. First, the user defines a starting point on the object boundary. Then, he/she moves the mouse cursor along this boundary. The tracing does not require to be precise since the approach displays the *most plausible* path

between the starting point and the current point that passes through the object boundary. When the current segment of the object boundary is satisfying, the user positions an anchor point by clicking. Then, partial segmentation results are validated. The algorithm computes the new path between the last anchor point and the new current point. Similar to active contours, the path minimizes an energy function generated by the configuration of the points that belong to the path. The energy is low if the points are located on strong edges (e.g., where the image gradient is high). The path is updated dynamically when the cursor is moving, and in ideal conditions, real-time feedback is provided to the user.

Contour-based interaction mechanisms constrain the user's attention to focus on the contour of the object during the segmentation task. This is intuitive as it preserves the integrity of a manual delineation task, i.e., drawing the limit separating the object from the background. Nevertheless, they also restrict freedom of action, i.e., the inputs follow the shape of the contour. In the case in where the energy function fails to capture the object boundary, the required effort to trace the contour using the live wire segmentation paradigm would be similar to that required for a manual tracing, i.e., the user needs to position anchor points all along the object boundary.

1.3.2 Region-based mechanism

In region-based approaches, the segmentation problem is defined as "finding pixels that *belong* to a particular object". A typical example of region-based segmentation is the region growing algorithm (Adams & Bischof, 1994). Starting from a region located inside the object, often manually selected, the approach iteratively appends pixels adjacent to the region that share similar properties (e.g., pixel intensity). The process stops when two successive iterations yield the exact same region, meaning that no additional pixels were added to the region. This algorithm is based on the regional property of the object instead of its contour, which makes it more sensitive to heterogeneous tissues, for example.

In the last two decades, *scribble-based* interaction mechanisms for region-based image segmentation have been widely used (Boykov & Jolly, 2001; Grady, 2006; Protiere & Sapiro, 2007; Falcao *et al.*, 2004). Using the mouse to draw on the image, the interaction mechanism consists in labeling a few pixels from each object and a few pixels from the background of the image, with their respective label categories. For a binary segmentation, the two label categories represent foreground and background. Based on the content of the image, the algorithm computes the *most plausible* separation between objects and background, according to these labels. This is similar to a region growing approach in which multiple regions grow at the same time. The assumption is that the speed of the growth is faster between pixels with similar properties, for example when pixels have similar intensities, i.e., with low gradient.

Compared to contour-based approaches, region-based approaches offer the user more freedom. In a typical case, the region occupied by the object is sufficiently large to allow a variety of valid labelling possibilities. Depending on the shape, the position and the order in which the labels were drawn, the response of the segmentation algorithm varies. While the actions of the user consist in following the object boundary during a contour-based segmentation task, there exist a much greater diversity of scenarios in which labels can be drawn during a region-based segmentation task, all leading to similar results. Some of these are more efficient than others, which motivates this thesis to focus on region-based approaches in general, and scribble-based approaches in particular.

1.3.3 Hybrid mechanism

Ramkumar *et al.* (2016) investigated the difference between contour-based and region-based interaction mechanisms in the context of 3D medical image segmentation. The study involved a method using the scribble paradigm to represent the region-based mechanism and a method using the live-wire paradigm to represent the contour-based mechanism. Results revealed that both mechanisms are comparable in terms of accuracy. However, the segmentation results were obtained slightly faster using the scribble-based segmentation method. This was achieved at

the cost of a higher cognitive workload induced by drawing background scribbles, as reported by the authors.

Instead of comparing contour-based and region-based mechanisms, other approaches combined both mechanisms to leverage their advantages. Yang *et al.* (2010) proposed a method to improve the interpretation of the user's drawing. In addition to the conventional foreground-background labels, the method allows the user to draw *soft* and *hard* constraint inputs. The soft constraint labels are interpreted as where the boundary should pass. The hard constraint labels indicate the pixels which the boundary must align with. Both constraints are contour-based drawings with different levels of interpretation, offering the user an advanced interaction mechanism. Spina *et al.* (2014) used a contour-based drawing to automatically generate foreground and background labels on each part of the drawn contour. Then, the segmentation is processed as a scribble-based segmentation. This allows the user to switch between the use of the region-based scribble drawing method and the contour-based live-wire drawing method. The idea behind this combination is to prevent leakage that can result from scribble-based segmentation. For example, if the segmentation fails to detect weak/missing boundaries, the segmentation result can overflow the object boundary, requiring the user to draw additional labels.

The approaches proposed by Yang *et al.* (2010) and Spina *et al.* (2014) do not consider the user performance in the segmentation assessment process. Even if the combination of contour- and region-based mechanisms allows a better interpretation for what the user wants to achieve through his/her drawings, switching between both mechanisms induces an additional workload. It is worth investigating whether such a combination is effective while taking into account the user performance. During the evaluation of the segmentation approaches proposed in this thesis, the user performance is considered through controlled user experiments.

1.3.4 Sketching assistance

In the same context of interactive image segmentation, *drawing assistance* is a field that has increasingly gained the research community's attention. In drawing assistance, the goal is to understand the user's intentions from a rough drawing or specific gestures. These inputs are then translated into more refined and detailed data that represent what the user intended to do. The challenge is to capture the knowledge encoded in the user's sketches. Because of the wide range of possibilities generated by the sketches, the complexity of the task is tremendous. Proposed methods often restrict these possibilities to be associated with a specific topic. The idea of assisting the user with sketching gestures is not recent. For example, in the context of software design, Landay & Myers (1995) proposed an approach that translates the user sketches into user interfaces. In their work, the input categories are already known and the goal is to associate each sketch to a widget. Forbus & Usher (2002) proposed a more generic approach where they attempt to understand the sketches drawn by a user. The goal was to facilitate the communication between people from different backgrounds by annotating the ideas behind the drawings.

In the context of 2D drawing, the approach introduced by Simhon & Dudek (2004) aims at refining the user's sketches to facilitate the drawings. First, in a training step, different sketches representing diverse scenes (more detailed drawings) are used to learn the relationship between the sketches and the scenes. During the sketching task, the method recognizes the different type of sketches and automatically classifies them according to the appropriate scene. Then, the sketches are replaced with associated scenes to create a more detailed drawing. In order to compensate for lack of drawing skills of novice artists, Xie *et al.* (2014) proposed a system called PortraitSketch, that helps the user to interactively generate the portrait of a person from an existing image. Authors argued the benefits of using an interactive system that preserves the *user's drawing style* compared to a fully automatic solution. Krs *et al.* (2017) proposed a method to assist the user in 3D modeling using sketches. The user draws a rapid sketch of a 2D curve that automatically wraps around 3D existing models.

Although the aforementioned methods do not address the segmentation problem directly, they highlight an important aspect of the user interaction mechanism which is present in interactive segmentation tasks: the interpretation of the user inputs. In this thesis, we investigate the relevance of the user drawings with respect to the image information. In Chapter 2, we show how to leverage the user's drawing to accelerate the segmentation process. In this approach, we ask the user to explicitly draw a region near the object's boundary where to focus the computations. Then, in Chapter 3, a similar region is used during the segmentation task. However, this time, the region is extracted implicitly, without interfering with the user drawing paradigm.

1.4 Graph-based segmentation

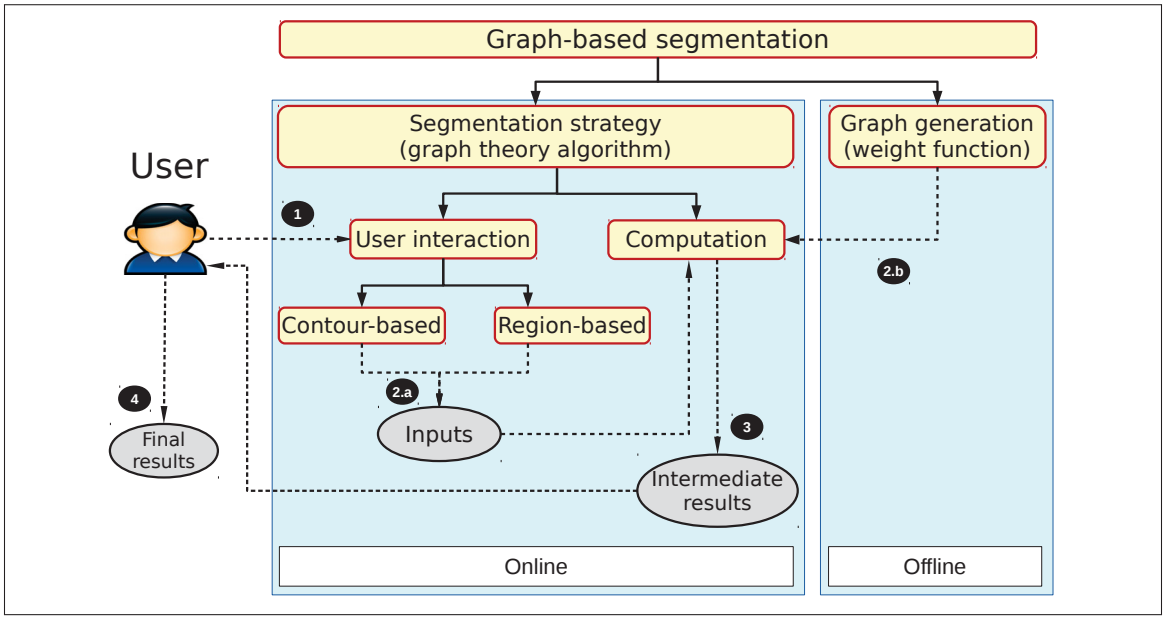

Figure 1.3 Graph-based segmentation flowchart: during the segmentation (1) the user provides input data using ether contour- or region based interaction mechanisms, then the input data (2.a) and the graph structure (2.b) are passed to the algorithm, which provides a result (3). The segmentation ends when the result is satisfactory, otherwise input data are added/modified (1) and the process is repeated

Many modern interactive methods use graph theoretical approaches to address the segmentation problem. This is because graph structures offer a flexible representation of the image,

allowing an easy adaption to different applications. Typically, a graph-based segmentation process involves two stages (Figure 1.3): (i) building a graph from an image, and (ii) applying a segmentation strategy, using graph theory.

The first stage often does not require heavy computations and can be performed offline, i.e., before the segmentation takes place. The goal is to prepare the data for the segmentation process. The graph structure can vary depending on the segmentation strategy used in the second stage. The second stage consists in computing the segmentation online. In the case of interactive segmentation, it involves reading the user's input data, then, computing and displaying the results.

The approach can easily be adapted to different applications by either: (i) adapting the graph structure, keeping the same segmentation strategy for example, in image segmentation (Boykov & Jolly, 2001), image registration (Tang & Chung, 2007) or stereo vision correspondence (Kolmogorov & Zabih, 2001); or (ii) adapting the segmentation strategy, keeping the same graph structure. For example in image segmentation, the approach proposed by Grady (2006) and that proposed by Protiere & Sapiro (2007) use similar graph structures.

1.4.1 Building the graph

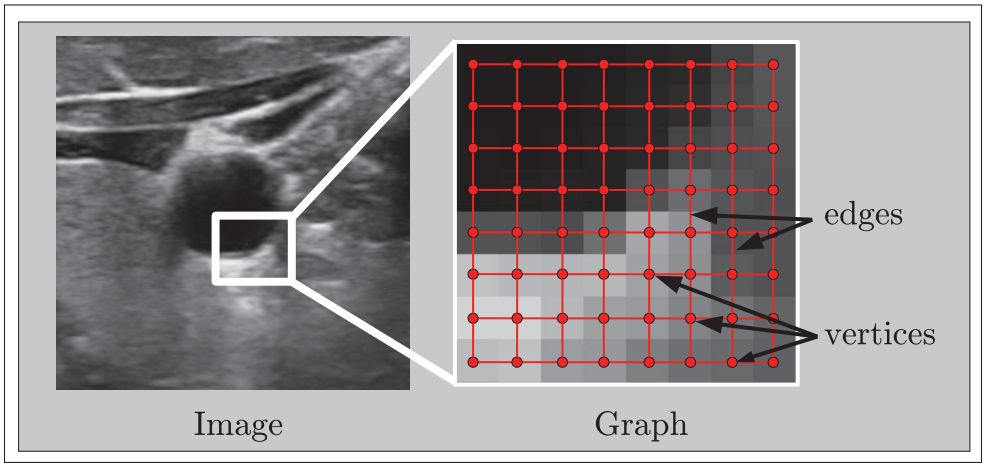

Figure 1.4 Example of image representation using a graph

In graph-based segmentation, the image is viewed as a graph $\mathscr{G} = \langle \mathscr{V}, \mathscr{E} \rangle$, where $v \in \mathscr{V}$ are the vertices corresponding to pixels of the image and $e \in \mathscr{E} \subseteq \{\{u,v\} : u,v \in \mathscr{V}\}$ are the edges connecting each pair of adjacent pixels (see Figure 1.4). A weight w_{ij} is assigned to the edge e_{ij} that connects vertices v_i and v_j. The weights encode the similarity between vertices. For example, the Gaussian weighting function has been used by Boykov & Jolly (2001); Grady (2006)

$$w_{ij} = \exp(-\beta ||g_i - g_j||^2), \tag{1.1}$$

while Li *et al.* (2004) used

$$w_{ij} = \frac{1}{||g_i - g_j||^2 + 1}, \tag{1.2}$$

where g_i and g_j are the pixel intensities at vertices v_i and v_j, respectively, and β is a user-supplied constant. A large β results in high sensitivity to weak boundaries, i.e., to differences in pixel intensity. Both functions have similar behaviour; i.e., they tend to decrease when the difference in intensity becomes larger. Let $d = ||g_i - g_j||$ represent the difference between two pixel intensities. Figure 1.5 shows the evolution of w_{ij} as a function of d for intensities between 0 and 1. Note that a large value of β causes a rapid decrease of w_{ij}, meaning that a small variation in the difference between pixel intensities induces a large variation of w_{ij}. Therefore, the computation becomes more sensitive to weak boundaries.

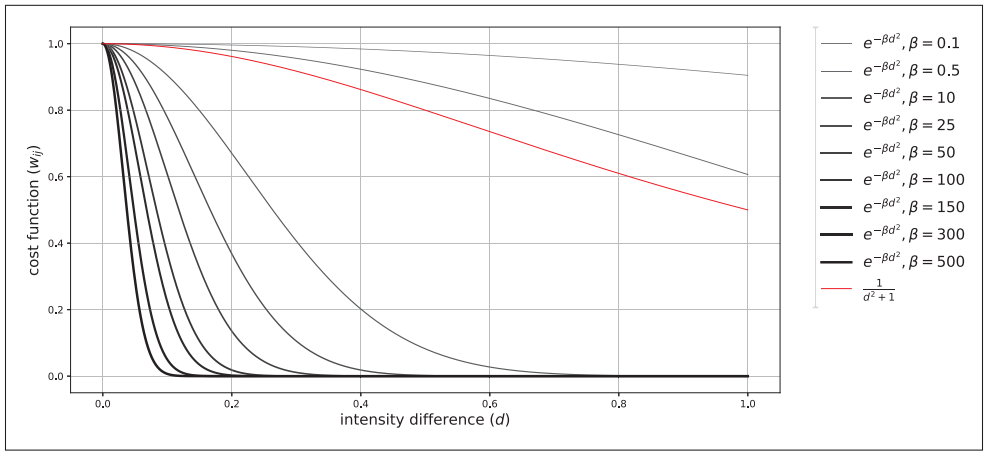

Figure 1.5 Weight function behaviour using Equation (1.1) with different values of β and Equation (1.2)

1.4.2 Segmentation strategy

The *Intelligent Scissors* (IS) proposed by Mortensen & Barrett (1998) is a technique based on the live wire paradigm, which uses a graph model for segmentation. While the user is tracing the contour, the algorithm adjusts the results on the fly using Dijkstra's algorithm (Dijkstra, 1959) so that it follows a minimum-cost path in the graph. Extensions of IS, including work by Falcão *et al.* (1998), Falcão *et al.* (2000), Mishra *et al.* (2008) and the Magnetic Lasso available in Adobe's commercial Photoshop software, were proposed to enhance segmentation flexibility. IS and its variants have two drawbacks: since the minimum-cost path must be computed efficiently during user interaction, the approach suffers from interaction feedback lags when applied to large images. Moreover, IS requires relatively high accuracy from the user when drawing the contour, which makes segmentation laborious (Li *et al.*, 2004).

In contrast to the *contour-based interaction* required by IS, Boykov & Jolly (2001)'s *graph cut* (GC) segmentation is a popular approach which typically uses a scribble-based interaction mechanism. GC segmentation uses foreground / background labels to remove edges to maximize flow (Boykov & Kolmogorov, 2004) (see Section 1.5.1), breaking the graph into two sub-graphs (foreground and background).

Variants of GC segmentation have reduced the required user interaction (Gulshan *et al.*, 2010). In *GrabCuts* (Rother *et al.*, 2004), for example, the user first frames the object inside a bounding box to reduce the search space. A Gaussian mixture model (GMM) is fitted to the cropped image intensities and labels are automatically generated according to the modes of the GMM. An initial segmentation result is then obtained using GC. The user can then add explicit foreground and background labels to adjust the segmentation. GrabCuts fails in the presence of weak boundaries, mostly because of the limited ability of the GMM to capture the true object intensity distribution.

In the presence of weak boundaries, GC leads to the "small cuts" miss-segmentation problem (Figure 1.6). To address this, Grady (2006) proposed random walker (RW) segmentation, wherein unlabelled pixels are assigned *probabilities* of belonging to each label category (fore-

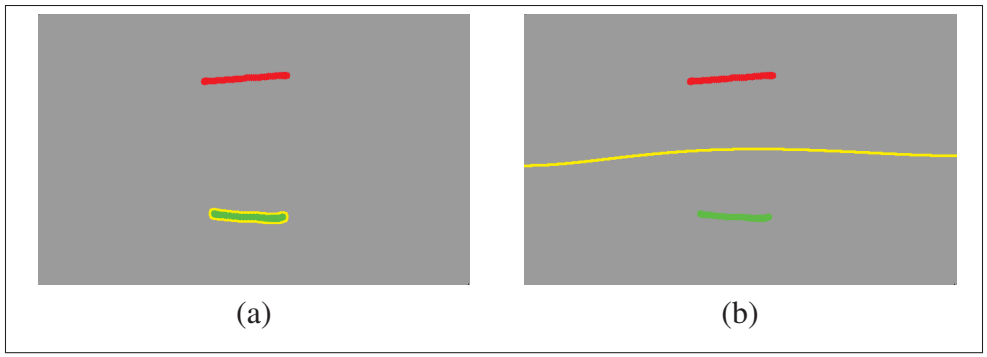

Figure 1.6 Illustration of the small cut problem in the absence of boundaries (uniform intensity allover the image). Because all the image graph edges have the same cost, the solution given by graph-cut segmentation consists in the smallest cut (the minimum number of edges in the cut). Segmentation results (yellow) obtained using: (a) the graph cut algorithm, and (b) the random walker algorithm

ground or background). Segmentation consists of selecting the most probable label for each pixel. In the absence of edges in the image, an unlabelled pixel is assigned equal probability of belonging to equidistant labels, thereby overcoming the small cuts problem. The contributions proposed in this thesis apply to most interactive segmentation methods using the scribble-based paradigm (Boykov & Jolly, 2001; Li *et al.*, 2004; Grady, 2006; Protiere & Sapiro, 2007). However, for concreteness, we focus on the graph cut and the random walker algorithms, which are briefly reviewed in what follows.

1.5 Computational properties of graph-based segmentation

In this section, we review three popular graph-based segmentation approaches from a computational point of view. Precisely, we identify the computational bottleneck for the *graph-cut*, the *lazy snapping* and the *random walker* segmentation algorithms, all of which were experimented with in this thesis.

1.5.1 Graph cut segmentation

The idea behind GC is to partition the image graph $\mathscr{G} = \langle \mathscr{V}, \mathscr{E} \rangle$ into two separate subgraphs corresponding to foreground and background categories. To achieve this, two special vertices, s and t, respectively called the *source* and *sink* terminals, are added to the image graph (Figure 1.7). The source terminal represents the foreground and sink terminal represent the background. Each terminal is connected to all the image graph vertices, therefore creating a new graph $\mathscr{G}' = \langle \mathscr{V}', \mathscr{E}' \rangle$, such that

$$\mathscr{V}' = \mathscr{V} \cup \{s,t\}. \tag{1.3}$$

$$\mathscr{E}' = \mathscr{E} \cup \{e_{i,s}, \forall i \in \mathscr{V}\} \cup \{e_{i,t}, \forall i \in \mathscr{V}\}. \tag{1.4}$$

A weight is associated with each edge connecting a terminal to an image graph vertex $i \in \mathscr{V}$, representing the likelihood of i belonging to the foreground $w_{i,s}$, and the likelihood of i of belonging to the background $w_{i,t}$. The graph is partitioned by removing a subset of edges $C \subset \mathscr{E}'$, such that the terminals s and t become disconnected. This particular subset of edges is called a *cut* and the *energy* of this cut is defined by the sum of the weights of its edges

$$|C| = \sum_{e \in C} w_e. \tag{1.5}$$

In graph cut segmentation, a weight represents the penalty of a transition between two vertices. Therefore, a cut is penalized if it contains edges connecting: (i) vertices to terminals with *strong links*, i.e., a high likelihood that the vertex belongs to the terminal, or (ii) vertices with similar intensities (according to Equation (1.1) or Equation (1.2)). Segmentation is defined as the problem of minimizing the cut energy given by Equation (1.5).

Two types of edges are involved in a cut: (i) image graph edges, connecting two adjacent image graph vertices and, (ii) terminal edges, connecting image graph vertices to a terminals.

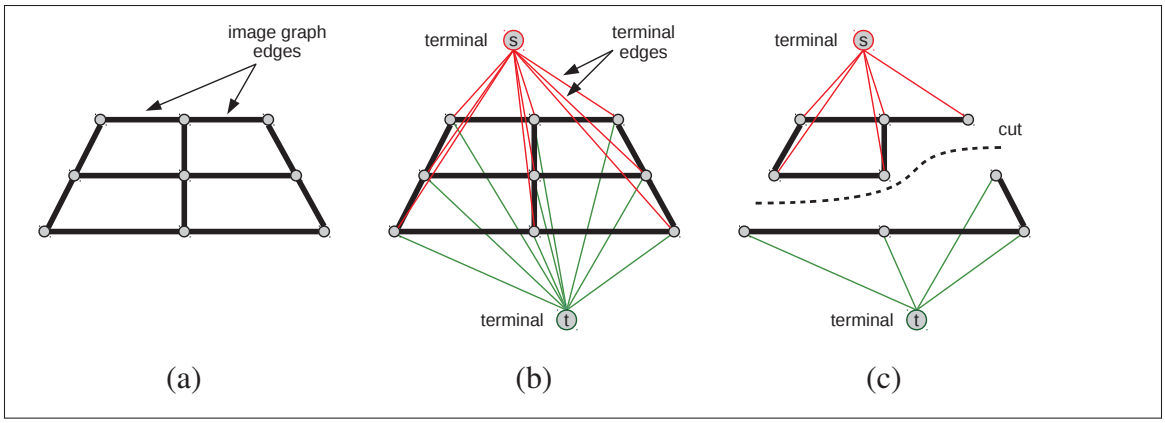

Figure 1.7 Illustration of the graph cut principle: (a) a tilted view of the image graph, (b) representation of the source s and sink t terminals, and (c) representation of a cut partitioning the graph into two subgraphs

Therefore, Equation (1.5) can be expressed as

$$|C| = \sum_{ij} E_{binary}(i,j) + \lambda \sum_{i} E_{unary}(i), \tag{1.6}$$

where E_{binary} represents the energy associated with image graph edges, i.e., the similarity between pairs of adjacent image graph vertices and E_{unary} represents the energy associated with terminal edges, i.e., the likelihood of belonging to one of the terminals. The parameter λ balances the two energy terms.

The binary term E_{binary} is computed using, for example, Equation (1.1) or Equation (1.2). The unary term E_{unary} is computed using the labels provided by the user. For example, in the original algorithm (Boykov & Jolly, 2001), the authors use the histogram of intensity distributions to capture the conditional likelihood of a vertex to belong to a given category (Greig *et al.*, 1989) as

$$\begin{cases} w_{i,s} = -\ln Pr(g_i|F), & \text{and} \\ w_{i,t} = -\ln Pr(g_i|B), \end{cases} \tag{1.7}$$

where g_i is the intensity at the vertex i, and F (resp. B) is the histogram extracted from foreground (resp. background) labelled pixels.

There exist many algorithms to solve the minimum cut optimization problem in polynomial time, as a function of vertices and edges (Goldberg & Tarjan, 1988; Ahuja *et al.*, 1993; Boykov & Kolmogorov, 2004; Orlin, 2013; Yuan *et al.*, 2014). In graph theory, solving the minimum cut problem is equivalent to finding the maximum flow going from the source terminal to the sink terminal (Ford Jr & Fulkerson, 1962). These primal-dual problems are often referred to as the min-cut/max-flow problem. They represent the computational bottleneck in graph cut image segmentation. The efficiency of the algorithms used to solve them depends on the number of edges and vertices in the graph. Therefore, the complexity increases polynomially with image size.

1.5.2 Lazy Snapping segmentation

A popular variant of graph cut-based segmentation is the *Lazy Snapping* algorithm (Li *et al.*, 2004), which provides good results in practice. Based on the original graph cut algorithm, the lazy snapping algorithm introduces two additional features: (i) the unary term in Equation (1.6) is computed using k-means distance (Duda *et al.*, 2000) instead of the histograms of intensity distributions, and (ii) in the case of segmentation errors, it provides a post-processing interaction mechanism which allows correcting the segmentation using a live-wire-based approach. In this section, we focus on the first point, which modifies the computational aspects of the algorithm and influences computation time. The unary energy E_{unary} is computed in two steps. In the first step, the labelled pixels are extracted to form two sets, namely a foreground set of pixels S^F and a background set of pixels S^B. For each set, the pixel intensities are clustered into n classes using a k-means algorithm (in the original algorithm $n = 64$). In the second step, for each vertex in the graph, the minimum distance is computed

$$\begin{cases} d_i^F = \min_n ||g_i - K_n^F||, & \text{and} \\ d_i^B = \min_n ||g_i - K_n^B||, \end{cases} \tag{1.8}$$

where g_i is the pixel intensity at vertex i, K_n^F (resp. K_n^B) denotes the mean intensity of the n^{th} cluster of S^F (resp. S^B). Finally, similar to the graph cut segmentation, E_{unary} is defined by the

wights given to terminal edges, for unlabelled vertices as

$$\begin{cases} w_{i,s} = \frac{d_i^F}{d_i^F + d_i^B}, & \text{and} \\ w_{i,t} = \frac{d_i^B}{d_i^F + d_i^B}, \end{cases} \tag{1.9}$$

and for labelled vertices as

$$\begin{cases} w_{i,s} = 0, & \text{and} \quad w_{i,t} = \infty \quad \text{if } i \in S^F, \\ w_{i,s} = \infty, & \text{and} \quad w_{i,t} = 0 \quad \text{if } i \in S^B. \end{cases} \tag{1.10}$$

Because the lazy snapping algorithm is based on the graph cut approach, it inherits its computational complexity regarding the min-cut/max-flow computation. Moreover, in the case of a large number of labelled vertices $|S_F \cup S_B|$, the computation of the unary energy using the k-means distances becomes time consuming. Therefore, the more labels there are the more time it takes to compute the weights of the graph. It is interesting to investigate the effect of this property on the computational time while designing a segmentation method. In fact, it creates a sensitivity to the number of labelled pixels used to perform the segmentation task.

1.5.3 Random walker segmentation

Assume that the user has manually labelled sets of foreground and background pixels (seeds), using a scribble-based approach. The random walker segmentation approach computes the probability that a random walk starting at each unlabelled pixel reaches a labelled pixel belonging to each of the foreground and background label categories first. To achieve this, the vertices $\mathscr{V}$ are partitioned into a set S of seeds and a set U of unlabelled vertices. By definition, the graph's Laplacian matrix $\mathbf{L}$ is given by

$$L_{ij} = \begin{cases} d_i & \text{if } i = j \\ -w_{ij} & \text{if } v_i \text{ and } v_j \text{ are adjacent} \\ 0 & \text{otherwise} \end{cases}, \tag{1.11}$$

where $d_i = \sum_j w_{ij}$ is the degree of the vertex i.

$\mathbf{L}$ can be rewritten in terms of seeded and unlabelled components as

$$\mathbf{L} = \begin{bmatrix} \mathbf{L_S} & \mathbf{B} \\ \mathbf{B^T} & \mathbf{L_U} \end{bmatrix},$$

where the subscript S (resp. U) denotes the seeded (resp. unlabelled) components of the Laplacian matrix $\mathbf{L}$, and $\mathbf{B}$ is the submatrix composed of the remaining elements of $\mathbf{L}$ (see the example in Figure 1.8).

Let $\mathbf{x}$ be an $N \times 2$ matrix such that each column contains the probabilities of the vertices belonging to one of the two label categories. We can represent $\mathbf{x}$ as

$$\mathbf{x} = \begin{bmatrix} \mathbf{x_S} \\ \mathbf{x_U} \end{bmatrix},$$

where $\mathbf{x_U}$ is the $|U| \times 2$ probability matrix of the unlabelled vertices and $\mathbf{x_S}$ is the $|S| \times 2$ probability matrix of the seeded vertices. The unknown probabilities $\mathbf{x_U}$ are obtained by solving

$$\mathbf{L_U x_U} = -\mathbf{B^T x_S}. \tag{1.12}$$

The speed of the algorithm depends on how efficiently Equation (1.12) is solved. Specifically, the solution requires the inversion of $\mathbf{L_U}$, which is, by construction, sparse and positive semi-definite so that a solution is guaranteed. However, the complexity of this operation scales linearly with the number of the unknown variables $|U|$, representing the computational bottleneck of the algorithm. Generally, $|S| \ll |U|$, so the segmentation time is strongly dependent on the image size $N = |\mathscr{V}|$.

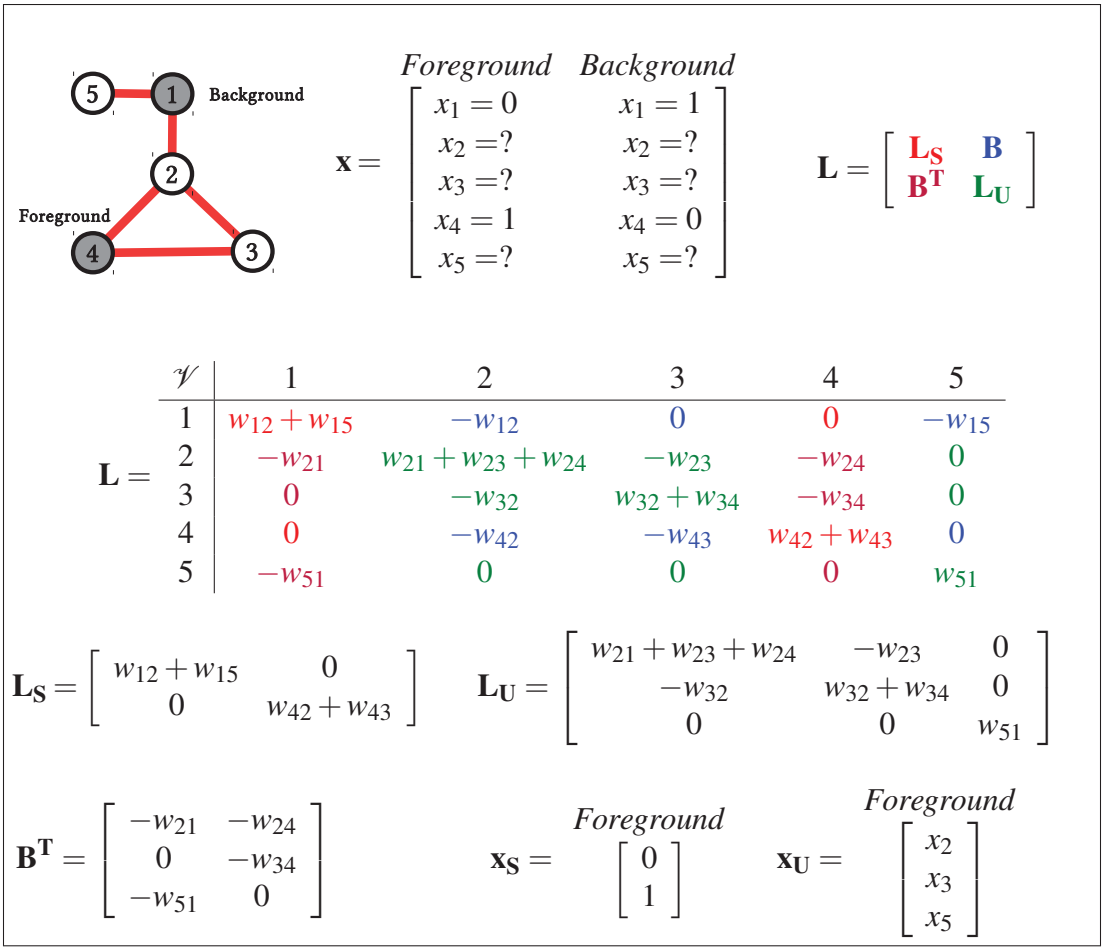

Figure 1.8 Example of Laplacian matrix computation: vertices 4 and 1 are labelled as foreground and background, respectively. The submatrices $\mathbf{L_S}$, $\mathbf{L_U}$, $\mathbf{B^T}$, $\mathbf{x_S}$ and $\mathbf{x_U}$ are highlighted in the Laplacian matrix $\mathbf{L}$

1.6 Graph reduction

One important drawback of graph-based segmentation algorithms is that the computation time increases with the image size. A common method to relax this computation burden is to reduce the graph dimensionality. This is performed by downsampling the image to reduce the number of vertices in the graph, i.e., the graph size. This approach reduces the resolution of the image, therefore affecting the quality of the final segmentation. There exist different strategies to downsample an image. We classify these strategies into two categories that are discussed in the remainder of this section: grid resampling and arbitrary-shaped resampling.

1.6.1 Grid resampling

In grid resampling methods, the original image I is considered as a grid of pixels. Then, every group of neighbouring $h_I \times w_I$ pixels is clustered into a single *pixel* in the resulting image. A straightforward approach is to compute the intensity value of the resulting pixel according to a *resampling function*, e.g., by taking the average intensity value, the maximum intensity value or the median intensity value of the $h_I \times w_I$ pixels. Figure 1.9.a shows an example of downsampling an image from 16×16 pixels to 8×8 pixels using the average intensity value.

This type of strategy is often used for image and video compression, in which the goal is to reduce the spatial domain to be represented by fewer information. For example, Zhang & Wu (2006) proposed to use edge structure information to guide the resampling method and to preserve structural information. Merhav & Bhaskaran (1997) considered a Direct Cosine Transform to reduce the information redundancy in the image. Trentacoste *et al.* (2011) proposed a framework in which they took into account blur information to preserve quality of the downsampled image.

1.6.2 Arbitrary-shaped resampling

The advantage of grid resampling is that the downsampled image can be computed very efficiently. Recent NVIDIA (Santa Clara, CA) graphical cards include resampling techniques on GPU using Cuda[2] technology. However, performing the segmentation on a downsampled grid results in a coarse segmentation contour (Figure 1.9.b). Alternative approaches consist in grouping pixels that share similar properties into a single element, called *super-pixel*, without grid restriction on the shape of the resulting element.

Super-pixel clustering methods are well suited for graph representation. Each cluster of pixels can be represented by a vertex in the graph (Figure 1.10). Computing super-pixels is a segmentation step in itself, where the goal is to find the best configuration such that the super-pixel

[2] `http://docs.nvidia.com/cuda/index.html`

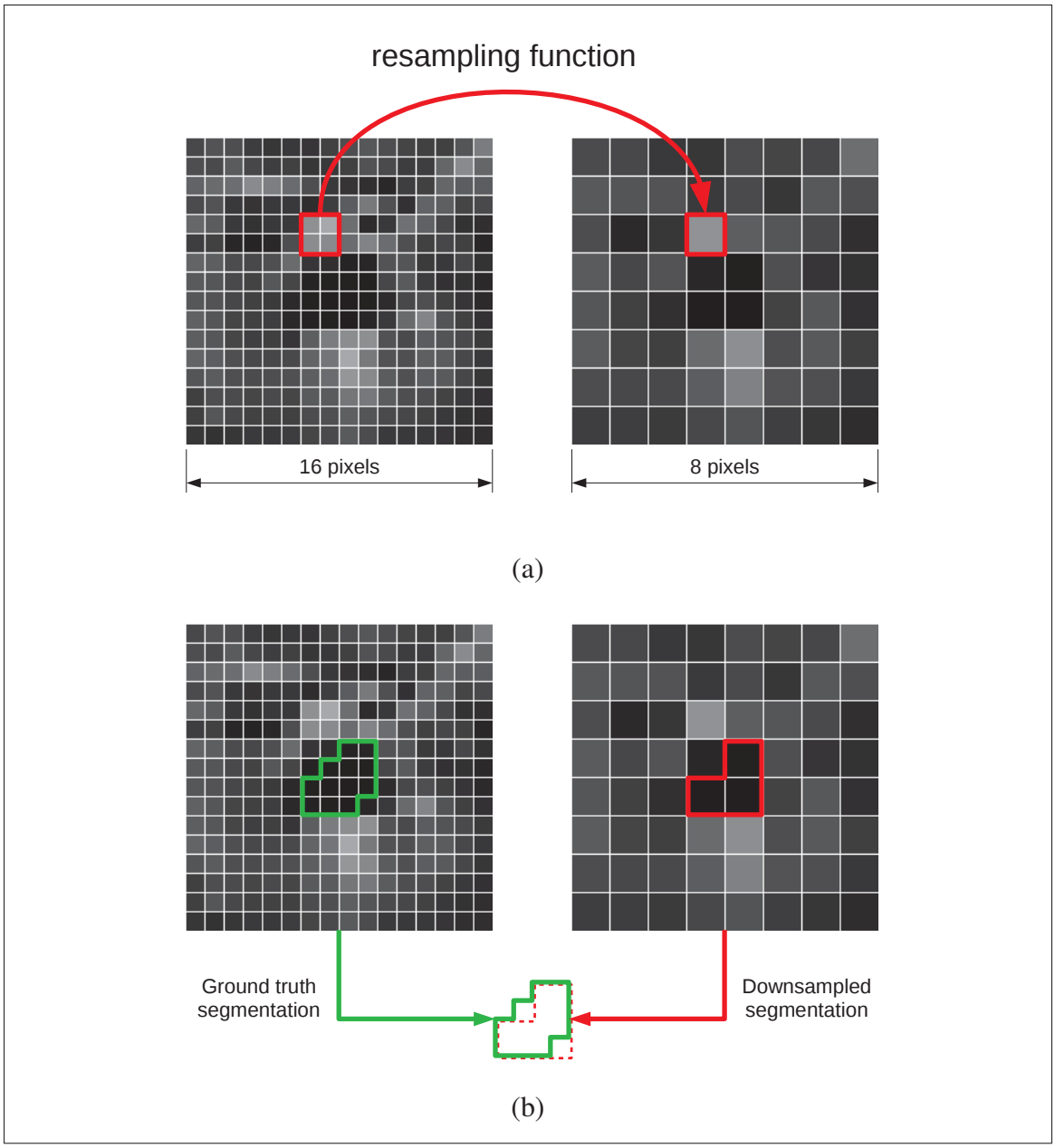

Figure 1.9 Example of reducing the image size using grid resampling method: (a) grid resampling and (b) segmentation on downsampled image

boundaries align with the boundaries of the structures present on the image. Therefore, when a super-pixel image is segmented, the structural information of the image is preserved, reducing segmentation errors.

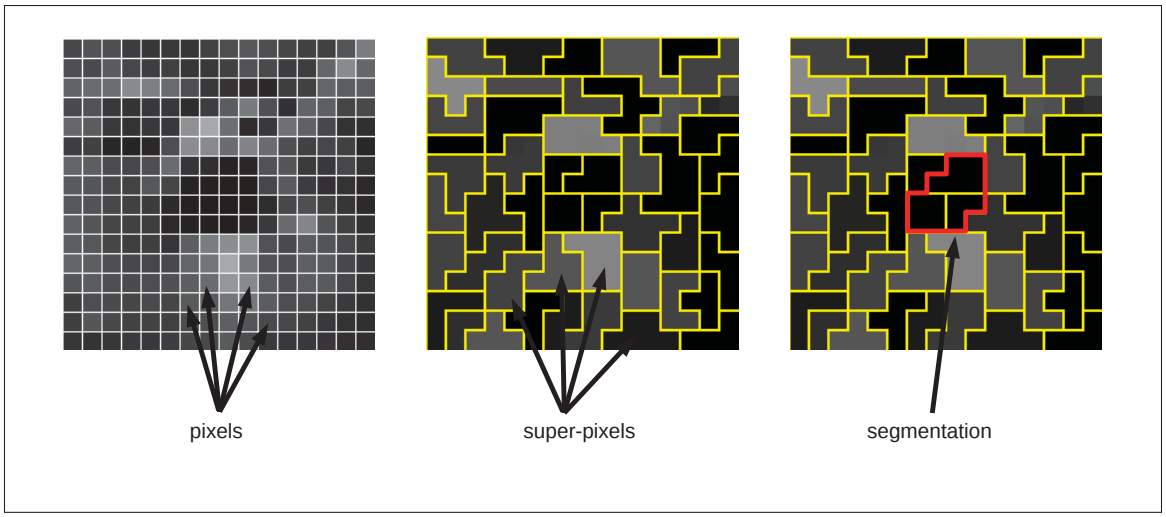

Figure 1.10 Example of image segmentation using super-pixels

There exist a variety of algorithms to compute super-pixels in an image (Saraswathi & Allirani, 2013). Among those techniques, the Simple Linear Iterative Clustering (SLIC) approach showed good performance for super-pixel clustering (Achanta *et al.*, 2012). The general idea of SLIC is to divide the image into k super-pixels arranged in a grid. Then, a k-means algorithm is used to iteratively deform each super-pixel, such that the size, the position and the color components of the super-pixels satisfy a compactness criterion.

Specifically, we first generate a grid on the image, representing an initialization of the super-pixel clusters. Within each grid cell, a seed point, called the *cluster center*, is generated. The seed point corresponds to the lowest gradient point within the cluster. Then, a k-means algorithm is used to iteratively assign pixels lying on neighbouring clusters to the closets clusters in terms of color intensity. Once a pixel is assigned to a seed cluster, the cluster center is updated according to the average color and location of the cluster.

Figure 1.11 shows examples of super-pixel clustering obtained using the SLIC algorithm. In the absence of gradient information (uniform intensity), the super-pixel clusters are unchanged, resulting in clusters with the initial grid shape. The key feature of the SLIC algorithm is to use the *CIE-Lab* color space (Wyszecki & Stiles, 1982) along with the (x, y) pixel position on the

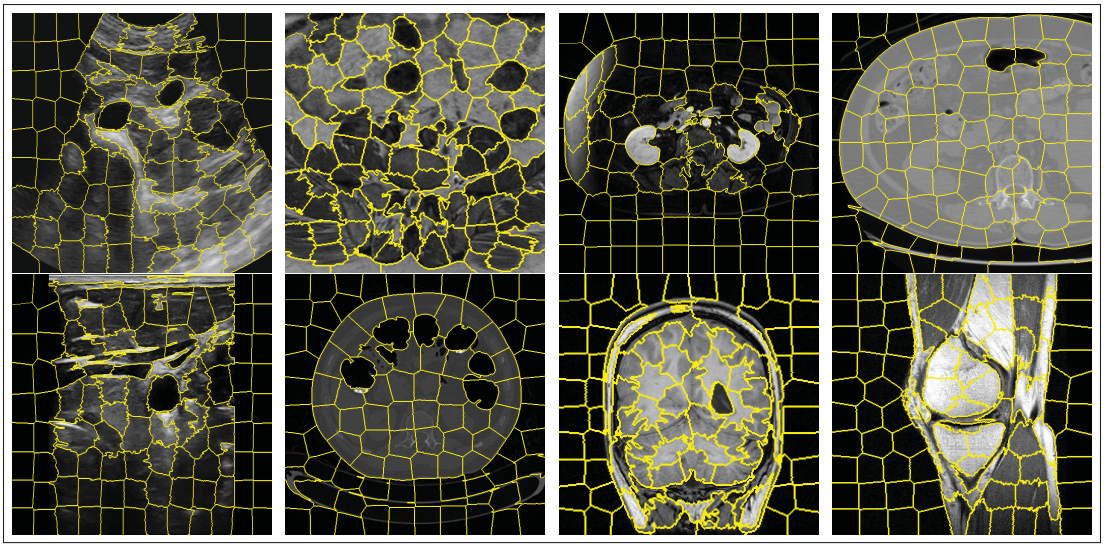

Figure 1.11 Example of super-pixel clustering using the SLIC algorithm

image to compute the distance between the pixel and a cluster. The SLIC algorithm is simple to use and provides good results compared to other approaches, e.g., (Levinshtein *et al.*, 2009; Mori, 2005), which makes it very practical.

CHAPTER 2

RAPID INTERACTIVE SEGMENTATION USING A ROUGH CONTOUR DRAWING

2.1 Introduction

With the growing popularity of scribble-based segmentation methods, many approaches have been proposed (Boykov & Jolly, 2001; Grady, 2006; Rother *et al.*, 2004; Li *et al.*, 2004; McGuinness & O'Connor, 2010; Protiere & Sapiro, 2007). For these, the computation of the segmentation must be efficient, enabling a tight feedback loop between the user and the algorithm. However, as noted in Section 1.5 the computation time increases with graph size, often precluding interactive segmentation of large images. For example, graph cut (GC) and random walker (RW) methods provide segmentation at interactive speeds for reasonably sized images (Li *et al.*, 2004) (512×512 pixels), but are not fast enough for high resolution images.

Fast interactive segmentation is not trivial because reducing the computation time does not necessarily lead to a reduction in overall segmentation time. The total time to perform a segmentation also depends on human factors, the input device used and the kind of input required by the segmentation algorithm. In this chapter, we propose a novel method, using a contour roughly sketched by the user, to reduce the size of the graph before passing it on to a segmentation algorithm such as RW or GC. This enables a significantly faster feedback loop at the cost of an additional drawing action. Beyond the acceleration of the computations, the purpose of the rough contour drawing is to investigate the relationship between the relevance of pixels and their distance from the object boundary. In fact, our hypothesis is that only the pixels near the object boundary are relevant to the segmentation process. Therefore, at the beginning of the segmentation, the user draws a rough contour of the object that will help to discard distant pixels while computing the segmentation results. Then, he/she proceeds to a regular scribbling to rapidly complete the segmentation task. Such a combination between contour-based and region-based interaction mechanisms has been investigated by Yang *et al.* (2010) and Spina *et al.* (2014). However, unlike our approach, these approaches do not address the issue of com-

putation time. A key contribution is that the rough contour interaction mechanism proposed in our method does not require an accurate tracing of the object boundary. It can be interpreted as a way to delimit the search space. A similar interaction mechanism is proposed by the GrabCuts algorithm, in which the user defines a bounding box around the object to characterize the object within the search space with a Gaussian mixture model. In our method we exploit the rough contour to reduce the search space. However, instead of confining the search space to the inside of a bounding box, the rough contour drawing is used to reduce the search space to pixels *near the object boundary*. This has three effects: (i) whereas the bounding box is constrained by object shape (e.g., a large bounding box is needed to surround a thin diagonally-oriented object), the proposed approach is more flexible and optimizes graph reduction for complex shaped objects, (ii) our approach ignores pixels that are sufficiently *far inside* the drawn contour, resulting in a speed-up, and (iii) our approach allows more flexibility in the drawing, i.e., the drawn contour may lie slightly inside and/or outside the object to segment.

The contributions of this work are threefold:

- We investigate the effect of two input techniques and two input devices (mouse and stylus pen) on RW segmentation performance;
- We demonstrate how the proposed graph reduction approach generalizes to different interactive graph-based segmentation approaches ensuring a precise, high-resolution segmentation;
- We evaluate our approach alongside, and in combination with, graph reduction methods based on single and multi-resolution super-pixels (Achanta *et al.*, 2012) to benefit from further speed-ups.

The remainder of this chapter is organized as follows. Section 2.2 reviews work related to interactive graph-based segmentation. Section 2.3 describes our user-guided graph reduction approach, and Section 2.4 discusses some of its key properties. Section 2.5 presents the user study, and Section 2.6 presents benchmarks obtained by generalizing our approach to other

graph-based segmentation methods and exploiting super-pixel-based reductions. Finally, Section 2.7 discusses the benefits and limitations of the proposed approach, and leads into the next chapter about exploiting the user interaction to reduce the graph size with a multi-resolution approach.

2.2 Related work

In order to improve the computational speed of RW segmentation, Grady & Sinop (2008) proposed to pre-compute the eigen-decomposition of the image graph's Laplacian matrix off-line. Therefore, the inverse of the Laplacian matrix $\mathbf{L}_U^{-1}$ is estimated rapidly during the segmentation. However, the pre-computation itself is time and memory consuming and unfeasible for live applications. Lermé *et al.* (2010) proposed a different method to reduce graph size for GC segmentation while preserving high resolution in parts of the image graph where maximum flow is high. During construction, vertices are discarded if they do not contribute significantly to min-cut/max-flow computation (Boykov & Kolmogorov, 2004). Unfortunately, for highly textured images, the graph is only reduced slightly and the time spent on graph reduction may not be compensated by the time gained during segmentation. GPU parallelization has also been considered to accelerate RW (Grady *et al.*, 2005) and GC (Delong & Boykov, 2008; Vineet & Narayanan, 2008), but is still constrained by hardware limitations because of the storage required for large datasets. The aforementioned works are specific to the segmentation approach used, in this case RW or GC. For a generalizable solution, another approach to improve the computation time consists in reducing the size of the graph.

Graph structures are well suited for dimensionality reduction. In fact, adjacent vertices can be grouped together according to a homogeneity criterion to form a single vertex called a *super-pixel*. This is typically performed before user interaction as a pre-segmentation step. Several approaches have been proposed to extract super-pixel structures from an image, such as normalized cuts (Yu & Shi, 2003), TurboPixels (Levinshtein *et al.*, 2009) and simple linear iterative clustering (SLIC) (Achanta *et al.*, 2012). For interactive graph-based segmentation, each super-pixel forms a single vertex in a new graph of smaller size. Super-pixels have been

used with a GPU implementation of RW segmentation (Gocławski *et al.*, 2015), with watershed clustering and RW segmentation (Couprie *et al.*, 2009), using hierarchical graph clustering and GC for video segmentation (Galasso *et al.*, 2014), and using random seed generation with a lazy RW strategy (Shen *et al.*, 2014). However, the super-pixel extraction step affects the quality of the segmentation results provided by the *main segmentation* algorithm (e.g., RW or GC). If super-pixel extraction fails to detect weak boundaries, the final segmentation inherits these errors and, more importantly, these cannot be corrected through user interaction. Moreover, super-pixels effectively reduce spatial resolution over the entire image, affecting the segmentation.

To our knowledge, very little work has investigated the relevance of user drawings for graph reduction. Such a drawing hints as to where the object boundary is likely to be. Although the bounding box method used by Hebbalaguppe *et al.* (2013) also leverages user input, this is basically to crop the image, and the user still needs to scribble the foreground. GrabCut (Rother *et al.*, 2004) also uses a bounding box, but fails in images with low contrast due to Gaussian mixture model fitting. The proposed approach maintains the same quality as the primary segmentation algorithm (e.g., GC or RW) used. Moreover, it addresses the lack of flexibility of bounding boxes for dealing with objects whose dimensions are not aligned with the pixel grid or whose shape is not convex.

2.3 Proposed graph-reduction method

Figure 2.1 illustrates our approach. First, the user roughly draws the object boundary. The object of interest is not required to fit inside the contour, making our approach more flexible than bounding box approaches. Starting from this user-drawn contour, a distance map is computed containing the distance from each pixel to the contour. Then, the distance map is used to partition the pixels (vertices) into *layers*, whose thicknesses increase according to the Fibonacci sequence (see Figure 2.1.b). Foreground and background labels are then automatically generated on two selected layers, called the *detail significance layers* (DSLs), and vertices beyond the DSL are eliminated from the graph. Finally, a segmentation algorithm (e.g., RW or GC) is

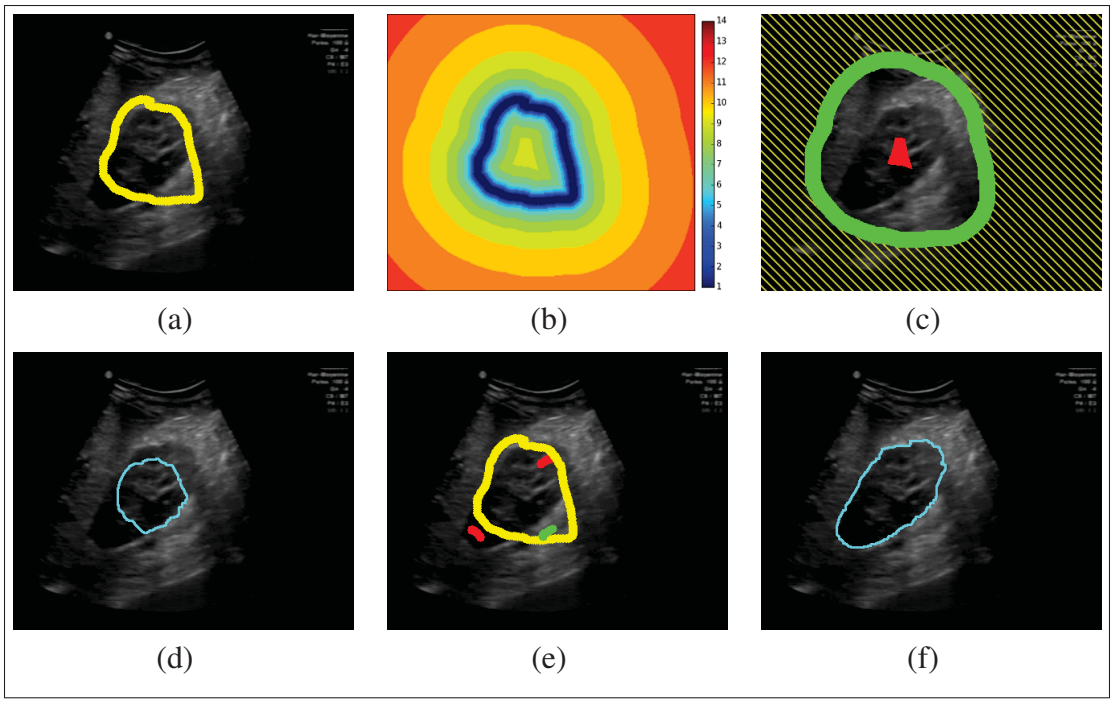

Figure 2.1 Example of a segmentation using our graph reduction approach: (a) Rough boundary quickly drawn by the user; (b) Layer thicknesses increasing according to the Fibonacci sequence; (c) Seed generation in the inner (red) and outer (green) regions, corresponding to the detail significance layers (*DSL*s); the hatched region (yellow) contains ignored vertices; (d) initial RW segmentation result; (e) refinement by the user with foreground (red) and background (green) labels and (f) final RW segmentation result

run on the reduced graph, thereby accelerating computation. Further benefits of our approach are: (i) it easily extends to super-pixel representations (Gocławski *et al.*, 2015; Couprie *et al.*, 2009; Galasso *et al.*, 2014; Shen *et al.*, 2014) for further graph reduction; (ii) it is parallelizable using a GPU implemention of the distance transform (Schneider *et al.*, 2009), so that the entire segmentation can be run on a GPU (Grady *et al.*, 2005; Delong & Boykov, 2008); and (iii) unlike super-pixel-based graph reduction, our approach preserves full resolution near the boundary and only one segmentation algorithm is required (e.g., RW or GC), thereby preserving the performance and homogeneity of the approach.

2.3.1 Layer construction

Assuming a cooperative user, the true object boundary is most likely to be near the drawn contour. To focus the search for the boundary near the drawn contour and ignore details in distant regions, we adaptively reduce image resolution according to the distance from the drawn contour. A Euclidean distance map D is computed as (Maurer *et al.*, 2003)

$$D(p) = \sqrt{\sum_i^d (p_i - l_i)^2}, \tag{2.1}$$

where the subscript i indicates the i^{th} coordinate in a d dimensional space, and l is the labelled pixel with the smallest Euclidean distance to the unlabelled pixel p. Thus, D is the distance from each pixel to the drawn contour. Pixels are then grouped into layers which quantify the significance, or relative *scale*, of the information contained in the image, based on the distance map. This notion of scale is naturally embedded in layers whose thicknesses increase multiplicatively with distance. Thus, the thickness $t(n)$ of the n^{th} layer is given by the exponential relationship

$$t(n) = ka^n, \tag{2.2}$$

where $a > 1$ is a constant representing the thickness ratio between layers n and $n+1$, and $k > 0$ is a multiplicative constant representing the thickness assigned to the contour drawing itself. For example, with a constant $a = 2$ each layer n is twice as thick as the previous layer $n-1$. We want to find, for each pixel p, the index (or scale) n that corresponds to the highest $t(n)$ that is lower than its distance $D(p)$ to the drawn contour. Hence, we assign a layer number, $\mathscr{L}(p)$, to each pixel p in the image such that

$$\mathscr{L}(p) = \left\lfloor \frac{\log(D/k)}{\log(a)} \right\rfloor. \tag{2.3}$$

In the particular case where

$$a = \frac{1+\sqrt{5}}{2\sqrt{5}}$$

thickness grows according to the Fibonacci sequence:

$$t(n) = \begin{cases} n, & n \in \{0,1\} \\ t(n-1)+t(n-2), & \forall n \geq 2. \end{cases} \tag{2.4}$$

Figure 2.2 shows examples of layer maps generated using different $t(n)$. Experimentally, we observe that the Fibonacci sequence provides a reasonable trade-off between the goals of maintaining high resolution near the drawing and rapidly decreasing resolution far away from it. For the remainder of this chapter, we choose the Fibonacci sequence to build the layers. However, a and k can be adjusted to adapt layer growth depending on the application.

In practice, Equation (2.4) is computed using Binet's formula (Stakhov, 2009)

$$t(n) = \frac{\phi^n - (-\phi)^{-n}}{\sqrt{5}}, \tag{2.5}$$

where the constant $\phi = (1+\sqrt{5})/2 \simeq 1.618$ is the *golden ratio*. Since

$$\left| -\frac{1}{\phi} \right|^n < \frac{1}{2}, \ \forall n > 1, \tag{2.6}$$

Equation (2.3) can be rewritten as

$$\mathscr{L}(p) = \left\lfloor \frac{\log(\sqrt{5}D(p) + \frac{1}{2})}{\log(\phi)} \right\rfloor, \tag{2.7}$$

were $\frac{1}{2}$ is added for rounding convenience based on Equation (2.6).

Note that the distance map D in Equation (2.7) need not be Euclidean; different distance metrics may suit different applications or imaging modalities. For example, a gradient-based geodesic distance would grow slowly in homogeneous regions, increasing the thickness of the associated layers. On the other hand, highly textured regions would be associated with a locally fast increasing distance map, leading to thinner layers. Therefore, image information can be integrated to the distance map to facilitate layer growth according to local image characteristics,

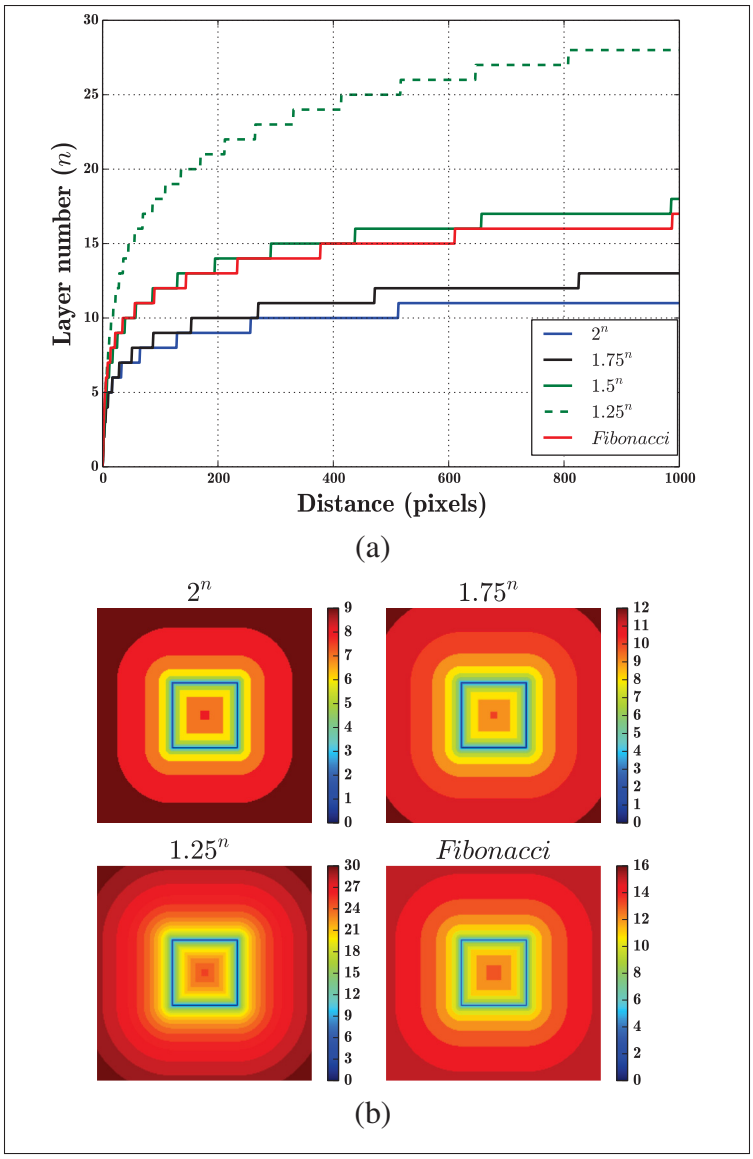

Figure 2.2 Effect of thickness function on layer generation: (a) plot of number of layers generated according to the distance from the drawn contour using different thickness functions $t(n) = ka^n$ with $k = 1$ and $a = 2$, $a = 1.75$, $a = 1.5$ and $a = 1.25$ and the Fibonacci function $t(n) = t(n-1) + t(n-2)$, (b) results of layer generation using a square drawing (blue)

e.g., with a texture-based geodesic distance map (Protiere & Sapiro, 2007). For the remainder of this chapter, we consider the Euclidean distance metric in Equation (2.1).

2.3.2 Segmentation

We assume that the user roughly labelled only one object boundary, thereby defining an *inner* region R_{in} and *outer* region R_{out}[1]. Recall that the flexibility of the contour drawing paradigm allows the true object boundary to lie on both the inner and the outer regions, which is not the case for the previously proposed user-selected bounding box (Hebbalaguppe *et al.*, 2013). Next, pixels are assigned to their respective layers using Equation (2.7) and the most distant layer for each region is computed:

$$L_{in} = \max(\mathscr{L}(p)),\ \forall p \in R_{in}$$

and

$$L_{out} = \max(\mathscr{L}(p)),\ \forall p \in R_{out}.$$

The smaller of these two numbers determines the index of the *detail significance layers (DSLs)*

$$DSL = \min(L_{in}, L_{out}). \tag{2.8}$$

Pixels lying on the *DSLs*, inside and outside the user-drawn contour, are automatically labelled as foreground and background seeds, respectively, and all vertices beyond the *DSLs* are discarded from the graph[2]. The runtime of RW segmentation is $O(|U|)$; thus, reducing the the number $|U|$ of unlabelled vertices improves computation time.

The proposed layer formulation naturally lends itself to a user-defined multi-resolution representation of the image. This can be obtained using super-pixel clustering, where the size of the super-pixels grows according to the layer where they reside. This highly general approach will be illustrated in Section 2.6.3. For the moment, we consider the binary case where the image is

[1] In a multi-label segmentation, the user must create separate objects by roughly marking their boundaries. This is the only constraint imposed on the user.

[2] We assume that no background regions (holes) are contained inside the target object to segment. This limitation can be addressed by drawing multiple contours inside the object to separate the background regions.

represented using only two resolution categories: (i) a *pixel-resolution* below the *DSLs*, and (ii) a *one-region-resolution* above the *DSLs*. This amounts to treating pixels lying above the *DSL* as a single vertex in the graph, leading to an inner vertex inside the object and an outer vertex outside the object. This particular case, achieved by thresholding the distance map according to Equation (2.8), allows us to validate our hypothesis that pixels far from the contour drawing (i.e., above the *DSL*) do not contribute to the segmentation.

The *DSL* can also be selected manually. This controls how loosely the user-drawn contour can fit the true contour. Increasing the *DSL* reduces the number of ignored vertices (the dashed yellow area in Figure 2.1.c), providing a larger unlabelled area where the contour is sought. Decreasing the *DSL* leads to a smaller graph and, therefore, to fewer computations, but may require a more accurate drawing to achieve good results. The *DSL* computed automatically with Equation (2.8) provides a reasonable compromise between these two extremes. Appendix I provides details about the computational complexity of the proposed graph reduction approach.

2.4 Interaction constraints and segmentation behavior

In this section, we highlight the explicit and implicit constraints imposed by the contour drawing paradigm compared to standard foreground-background seeding (FBS) input. Then, we discuss the sensitivity of our approach to inaccuracies in the contour drawing.

2.4.1 User interaction constraint

Most cases require approximately surrounding the object with the contour drawing to obtain a satisfactory segmentation. In the presence of weak boundaries, the FBS approach implicitly embeds a similar spatial constraint on seed positioning. Figure 2.3 shows an example of a RW segmentation using FBS interaction. Due to the initial seed positions, the user is forced to correct the segmentation with background labels through several feedback exchanges with the segmentation algorithm. For high resolution images, segmentation feedback can take a longer time, rendering this interaction tedious. It is possible to anticipate the behavior of FBS

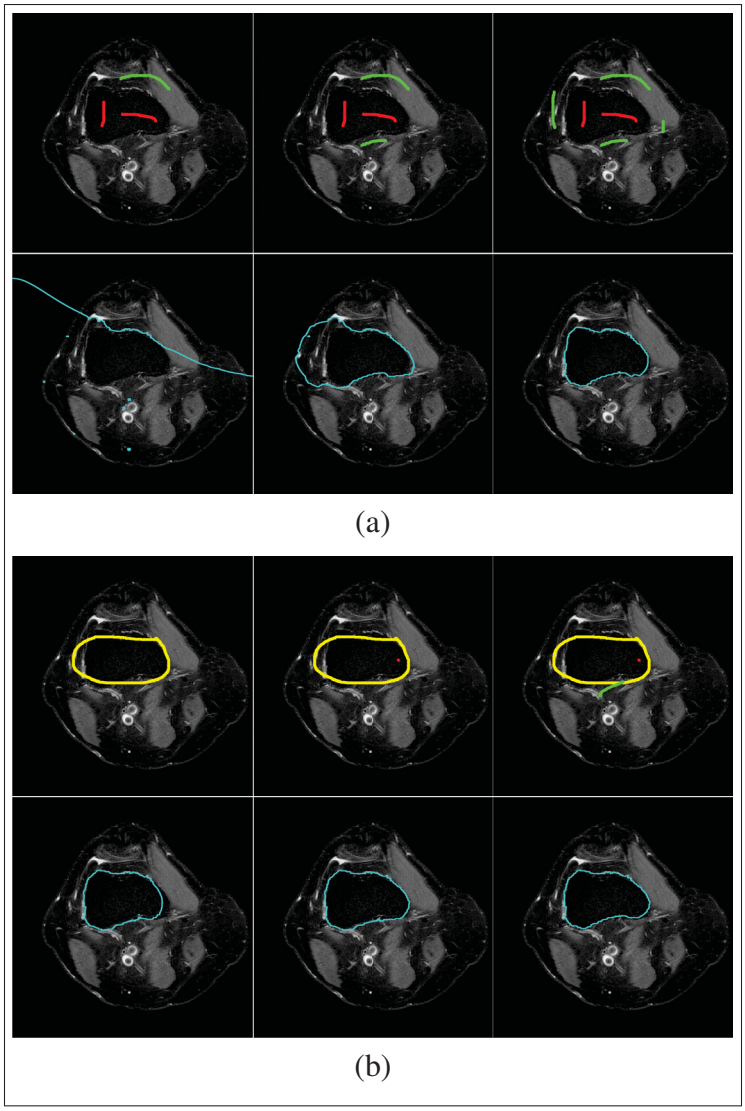

Figure 2.3 Step-by-step RW segmentation example showing the geometric constraints of label positioning: (a) conventional RW segmentation approach; (b) our segmentation approach. Top rows show foreground (red) and background (green) seeding and rough contour drawing (yellow), respectively. Bottom rows show results (blue). Label positioning constraints force the user to correct the segmentation through multiple iterations, surrounding the object with background labels. This is explicitly expressed with the rough contour drawing in our approach

segmentation by labelling the potential segmentation overflow areas surrounding the object. This very nearly amounts to a rough contour drawing.

Figure 2.4 shows more examples of RW segmentation using our approach. In most cases, the contour drawing is sufficient to correctly segment the object. For other complex scenarios where the boundary is weak or missing, few additional foreground-background labels are required (e.g., the second row of Figure 2.4).

2.4.2 Sensitivity of the contour drawing

To measure the sensitivity of our method to the drawing, we systematically evaluated the quality of the segmentation as this drawing departs from a ground truth segmentation. That is, the drawing was iteratively shrunk inward (respectively expanded outward) the object. At each iteration, 50 contour drawings were generated using a random path that roughly follows the shrunk or expanded curve. A random path starts at a given point p_i on the curve. Then, it iteratively generates the position of the next point p_{i+1}, by slightly varying the angle and distance formed by p_i and p_{i+1} using a Gaussian random variable. The details of the algorithm can be found in Appendix II. The segmentation result associated to each trial is evaluated using the harmonic mean of precision and recall (also known as the Dice index), denoted

$$F_1\text{-}Score = \frac{2TP}{2TP + FP + FN} \in [0,1], \tag{2.9}$$

where TP, FP and FN are the true positive (object surface), false positive and false negative scores, respectively. The F_1-*Score* tends to 1 as the segmentation result approaches the ground truth. Figure 2.5.a shows the F_1-*Score* as a function of a drawing's distance from the true boundary. The results prompt two important observations. First, note that the curve is skewed in the outward (negative) direction meaning that the interaction is slightly more robust to errors when the contour is drawn outside the object boundary. Second, significant cyclic drops of the F_1-*Score* can be observed. These are related to the quantization of the layers generated using the Fibonacci sequence (see Figure 2.6). In fact, Equation (2.8) selects the largest layer number. However, the layer with the largest number in the inner region can occasionally contain too few pixels, leading to segmentation errors. Examples of segmentation failure are shown in Figure 2.5.b-c. Note that the location of the F_1-*Score* valleys are specific to the image and

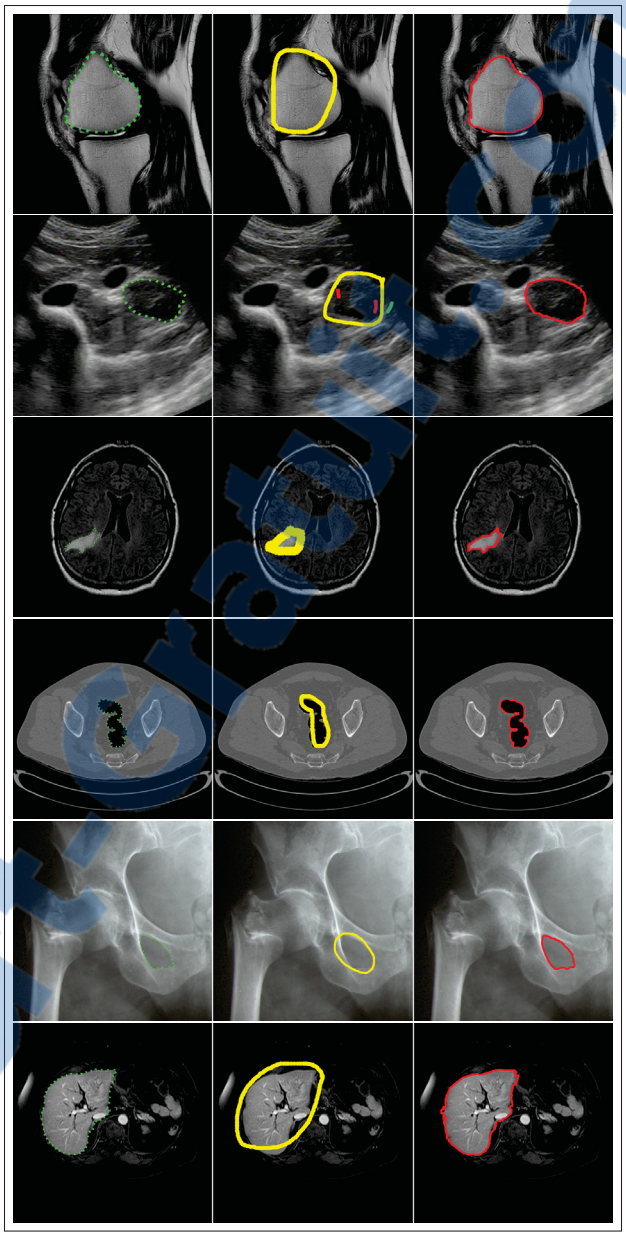

Figure 2.4 Examples of segmentation using the random walker with our approach: (left) ground truth image, (middle) rough contour drawn by the user, and (right) segmentation results. For most cases, a rough contour drawing is sufficient to obtain a satisfactory segmentation. Note that for the complex segmentation of the ultrasound image of kidney (second row), a few additional foreground-background labels are required

the considered object. In practice, this does not affect the quality of segmentations obtained interactively, as rapid response from the segmentation algorithm allows for a quick interactive

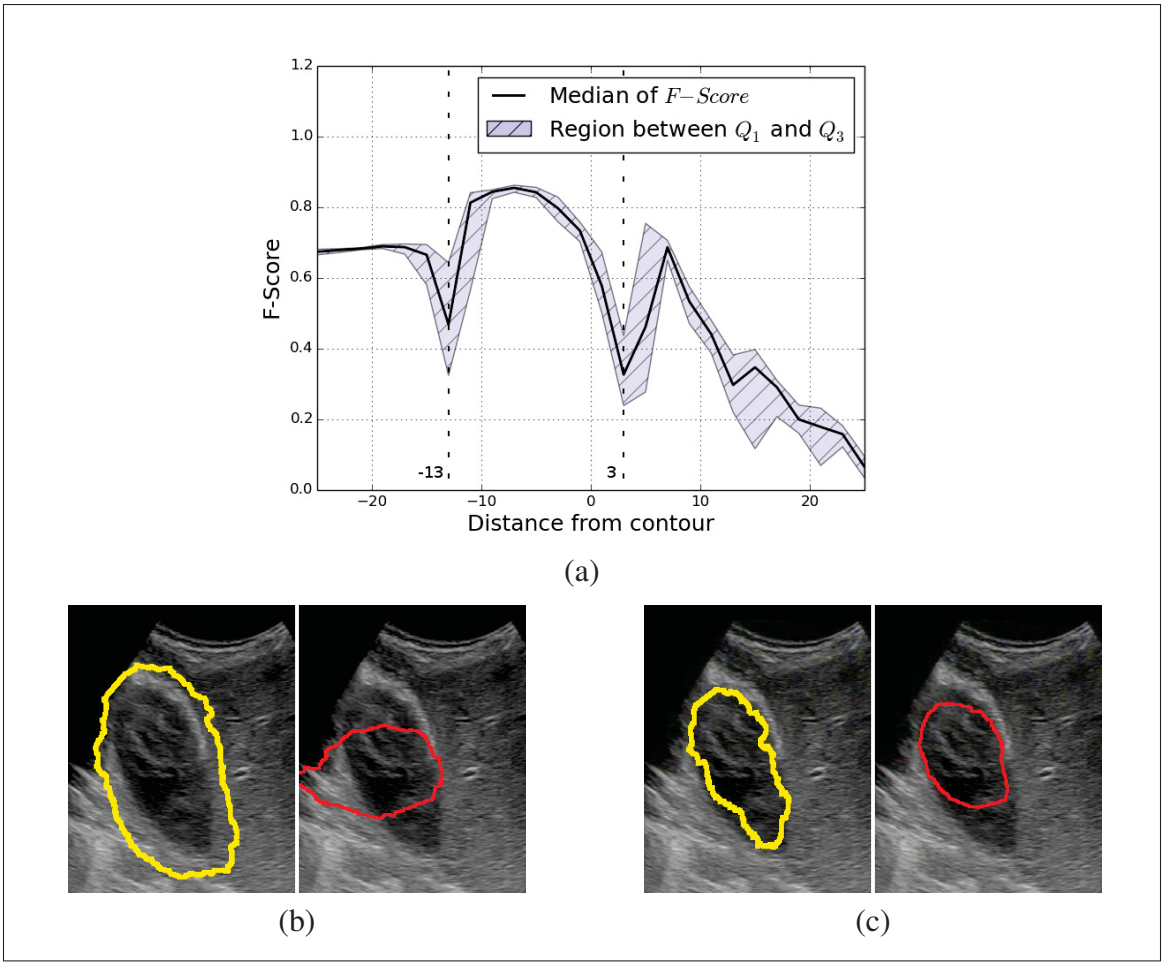

Figure 2.5 Sensitivity of the algorithm to the accuracy of the contour drawing: (a) plot of median value of F_1-*Score* and range between first and third quartiles, respectively Q_1 and Q_3 as function of drawing distance to the true object boundary; positive (resp. negative) distance indicates an inward (resp. outward) drawing distance. Examples of segmentation failure at distance -13 pixels (b) and 3 pixels (c)

correction with few additional labels, and all corrections benefit from the initial speed-up. This is demonstrated experimentally in Section 2.5.

2.5 User study

For this experiment, we use RW as the segmentation algorithm. We conducted a controlled experiment to compare our rough contour drawing (RCD) method against the conventional foreground-background seeding (FBS) approach for RW segmentation. RCD has the advantage

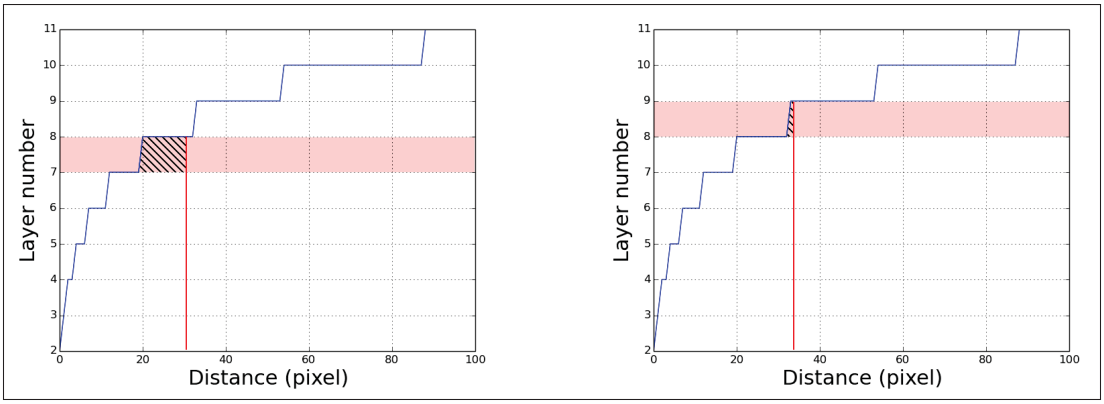

Figure 2.6 Illustration of the quantization effect of the Fibonacci sequence on seed generation: The left image represents a distance of 31 pixels generating a $DSL = 8$, the number of seeds generated is related to the size of the dashed area. On the right image, a higher $DSL = 9$ is generated with a distance of 36 pixels. However, the number of generated seeds is smaller

of reduced computation time for each iteration of segmentation, but incurs the up-front cost of drawing a contour, hence the need for an experimental comparison. The following questions are addressed: (i) Does RCD provide satisfactory results in terms of segmentation quality? (ii) Does RCD's reduced computation time help the user reduce the overall segmentation time? (iii) Is performance affected by the input device used? We assessed performance by measuring the quality of the final segmentation, the overall time to complete a segmentation, and the number of labels drawn by the user.

2.5.1 Study design

With the FBS drawing technique, the user labels the foreground object and the background as is conventionally done (Grady, 2006). With the RCD drawing technique, the user draws a rough contour of the object to reduce the graph and later uses foreground / background labelling to refine the segmentation. Two input devices were used: a standard mouse, and a "Grip Pen" on a Wacom Cintiq Companion Hybrid graphics tablet connected to the desktop PC. The mouse acceleration was disabled to prevent any effect caused by speed variability. The Wacom device allows the user to physically draw on the tablet screen using a handheld stylus pen. There were thus two experimental factors crossed to form four main conditions: *Device* (Mouse or Pen) ×

Drawing (FBS or RCD) yielding the combinations ***M+FBS***, ***P+FBS***, ***M+RCD*** and ***P+RCD***. Equation (1.1) with $\beta = 300$ was used in all cases, and all processing was done on an Intel© Core i7-2630QM 2GHz $\times$ 4 machine.

An ethical approval was obtained from Ecole de technologie superieure to conduct the study. Sixteen participants (13 male, 3 female), primarily undergraduate and graduate students with no particular expertise in medical imaging, were recruited. Some had prior experience with interactive segmentation using FBS with GC and/or RW.

An initial dataset of 22 images was prepared, ranging from 256×256 to 1348×1101 pixels. Images comprised computed tomography (CT), magnetic resonance (MR) and X-ray images from the cancer imaging archive database (Clark *et al.*, 2013), to which we added ultrasound (US) images acquired with an Ultrasonix SonixTablet (Ultrasonix Medical Corp., Richmond, BC, Canada). Images were selected to cover a broad range of medical applications: brain imaging (MR, CT), carotid imaging (US), abdominal imaging, e.g., kidney, bladder, prostate (US, MR and CT) and chest and pelvic imaging (X-ray). All images were manually segmented to generate ground truth data. This dataset of 22 images was then divided into two subsets, dataset 1 (*DS1*) and dataset 2 (*DS2*), of 11 images each.

Each participant completed tasks under the four main conditions (*M+FBS*, *P+FBS*, *M+RCD*, *P+RCD*) whose order was counterbalanced according to a 4×4 Latin square (i.e., each quarter of the participants went through the conditions in an order given by one row of the Latin square). For each participant, two of the conditions were performed using *DS1*, and the other two were performed using *DS2*. Further counterbalancing ensured that half of the participants started with *DS1*, the other half started with *DS2*. In total, there were 16 participants $\times$ 2 Drawing conditions (FBS and RCD) $\times$ 2 Devices (Mouse and Pen) $\times$ 1 data set (*DS1* or *DS2*) $\times$ 11 images per data set = 704 interactive segmentation trials.

In each trial, participants were shown the ground truth segmented image and were asked to reproduce a similar result. The ground truth segmentation was provided because the purpose of the study was to assess segmentation performance, not the medical image interpretation

skills of the participants. Participants assessed the accuracy qualitatively by visualization of the ground truth image. No time limit was set for the segmentation task and the accuracy was left to user satisfaction. This reflects the trade-off between ease of use and the time required to segment the image. For each main condition (*M+FBS*, *P+FBS*, *M+RCD*, or *P+RCD*), participants were first introduced to the segmentation method through a training session using 13 images belonging to neither *DS1* nor *DS2*. No data were recorded during training. Then, during the recorded session, the participants were asked to perform the most accurate segmentation they could with respect to the ground truth in the shortest time possible for each image. A trial consists of segmenting one image. During each trial, editing (where the user positions the labels) and processing (where the segmentation results are computed) phases alternated.

With interactive segmentation, any additional user interface features, such as ability to zoom or undo, would affect performance, but these are not the focus of our study. We therefore controlled for such differences between user interfaces by keeping the user interface as simple as possible, restricting user actions to (i) drawing foreground, background and contour (for RCD) labels, (ii) resizing the drawing brush, and (iii) erasing seeds. Participants could not zoom or undo.

At the end of each trial, when a satisfactory segmentation result for the image was obtained, we recorded the time required to perform the segmentation and the accuracy of the final result using Equation (2.9). Because user performance varies substantially between images, the overall segmentation performance for the whole dataset was considered, rather than for each individual image. In other words, the time reported in our results section is the total time required for a participant to segment all 11 images, and the accuracy score is the average F_1-*Score* obtained over the 11 images.

2.5.2 Implementation

All the algorithms were implemented using *Python*. The core RW algorithm was taken from the *Scikit-image* open source library[3]. The user interface was developed using the *Qt4* library (Figure 2.7). With the mouse, the user draws foreground and background labels using left and right buttons, respectively, and holds a keyboard button "Contour" while drawing the rough contour (only for RCD) using right button. The user can activate an erase mode by pressing a keyboard button "Erase". When the erase mode is activated the user can delete the drawn labels using the left mouse button. The erase mode can be deactivated by pressing the "Erase" button again. With the tablet and the pen, the user switches between foreground and background labels (for FBS and RCD) and rough contour (for RCD) by pressing a button located on the side of the pen.

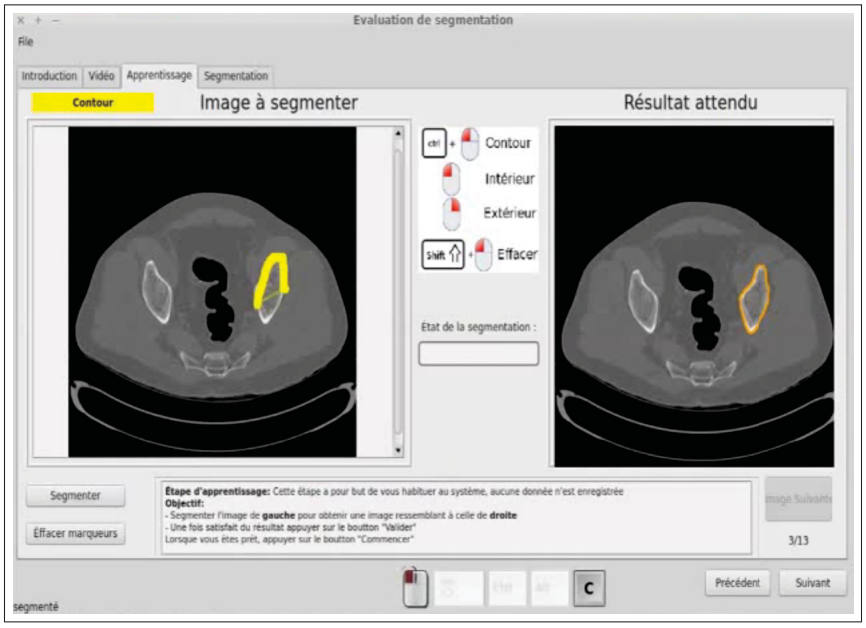

Figure 2.7 The user interface developed for the experimentation: the user draws on the left image and the results are displayed on the right image

[3] http://scikit-image.org/docs/stable/api/skimage.segmentation.html

For the super-pixel computation (see Section 2.6.1), the scikit-image library and Python were used to calculate the super-pixel clusters. Then, each cluster is converted to a graph vertex before passing it to the segmentation algorithm.

2.5.3 Results

Table 2.1 Key conclusions of the user study

Criterion	Results
Time	The fastest interactive segmentations are achieved using **our approach** (RCD), irrespective of the device (mouse or pen) that is used.
Accuracy	Our graph reduction approach does not sacrifice accuracy in any significant way, irrespective of the device (mouse or pen) that is used.
Human effort	Overall, interactive segmentation that is initiated using our RCD approach does not require significantly more manual labelling than segmentation initiated with the conventional FBS approach.

Table 2.1 summarizes our conclusions. The details of the analysis supporting these conclusions are provided in the next two subsections, with important results in bold.

2.5.3.1 Interaction

Figure 2.8 shows average time and accuracy for the main conditions. A Shapiro-Wilk normality test revealed that the time taken to perform segmentations was not normally distributed [$p < 0.01$]. Therefore, a non-parametric ANOVA-type statistic (ATS) test (Brunner *et al.*, 2002) was considered. Segmenting using the RCD ($mean_{RCD}(Time) = 65.09\,\text{s} \pm 3.98\,\text{s}$) was significantly faster than using FBS ($mean_{FBS}(Time) = 113.43\,\text{s} \pm 6.92\,\text{s}$) [$p < 0.01$]. Qualitatively, the time reduction was more substantial for larger images. These results support our initial hypothesis that **the time spent drawing the rough contour results in a significant gain in the overall segmentation time**. Segmentation using the pen ($mean_{pen}(Time) = 94.65\,\text{s} \pm 6.67\,\text{s}$)

was significantly slower than using the mouse ($mean_{Mouse}(Time) = 83.86\,\text{s} \pm 7.42\,\text{s}$) [$p < 0.01$]. Considering the intuitiveness of the tablet and pen for drawing, this was unexpected. This result could be explained by the low drawing accuracy required by RW segmentation to obtain satisfactory results. The F_1-*Score* results show that both devices provide sufficient control to perform a good segmentation. Indeed, no significant difference was found between the four conditions regarding the F_1-*Score* ($mean(F_1\text{-}Score) = 0.919 \pm 0.001$) with [$p = 0.70$] and [$p = 0.14$] for the *Drawing* and the *Device* factors, respectively ; meaning that **using the rough contour drawing (RCD) method, users achieved the same segmentation accuracy in shorter time with both devices**.

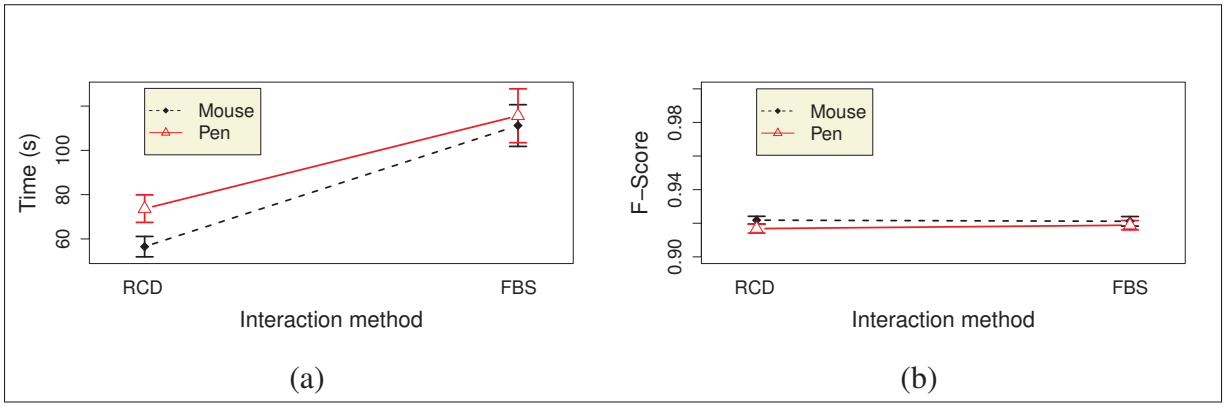

Figure 2.8 Results of the experiment: (a) The average time for each condition and (b) the average F_1-*Score* (a larger score means better accuracy) for each condition

To evaluate user effort, we analyzed the number of pixels that were labelled by the participants and the number of segmentation feedback exchanges required to achieve the final segmentation (i.e., the number of times the user pushed the segmentation button to view an intermediate result), shown in Figure 2.9. Tukey contrast tests (Munzel & Hothorn, 2001) on the normalized number of labelled pixels and the number of segmentation feedback exchanges reveal no significant differences between the four conditions, suggesting that **all the approaches required similar amounts of effort**, on average.

However, a Fligner-Killeen test (Conover *et al.*, 1981) reveals a significant inhomogeneity of variances in the number of labelled pixels [$\chi^2 = 21.95, df = 3, p < 0.01$]. This reflects the

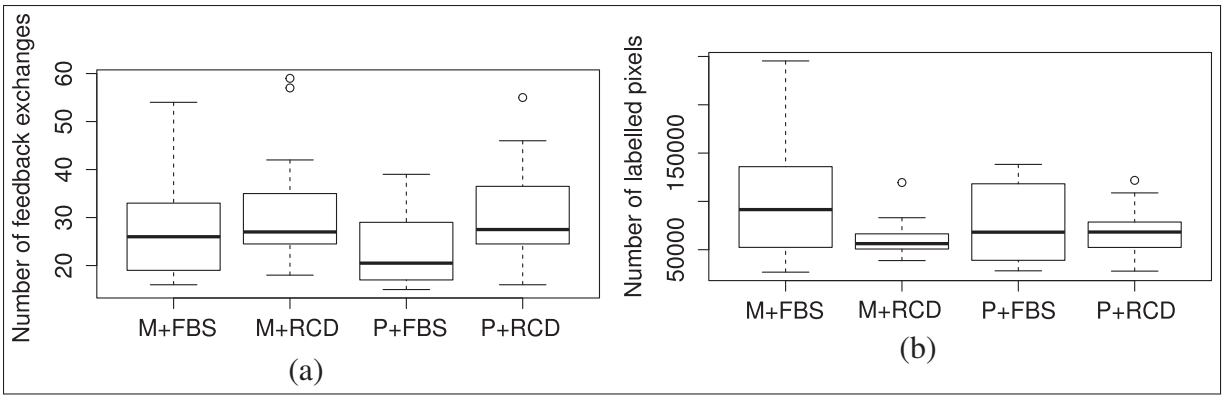

Figure 2.9 Box plot of (a) number of segmentation feedback exchanges and (b) number of labelled pixels. Note that in the latter, FBS shows larger variations than RCD

larger inter-quartile range displayed for the FBS approaches in Figure 2.9.b. In other words, **the conventional labelling approach allows larger variation in the drawings**. This result is related to the implicit label positioning constraints in the FBS approaches, discussed in Section 2.4.1. The FBS approach gives the user more freedom in drawing, but the positions of truly useful labels are actually constrained by the algorithm. Therefore, the number of labelled pixels varies significantly from one user to the next, depending on the usefulness of their drawings. In contrast, the RCD approach is explicitly constrained, thereby facilitating the seeding process.

2.5.3.2 Computation time

From the previous study, we extracted the final labels generated by the participants for each image. We also recorded the average time required to compute the segmentation on each image, ignoring the time required by participants for drawing. Figure 2.10.a shows that the average computation time grows linearly with image size. This is because the size of the linear system that produces the RW probabilities (Equation (1.12)) is proportional to the size of the image. The computation time trend with respect to image size is estimated with a linear regression giving $6.203\,s/Mpixel$ for the FBS and $1.577\,s/Mpixel$ for the RCD. Therefore, **our approach is on average 3.93 times faster than the conventional RW segmentation**.

Figure 2.10.b shows average overall segmentation time (including user interaction time) as function of image size. First, we note that the time required for the user to perform segmentation increases with image size. Second, the most significant benefits of the contour-based graph reduction approach occur for large images. For the largest image with size 1348×1101, the average participant performed the segmentation in $35.28\,\mathrm{s} \pm 8.42\,\mathrm{s}$ for *M+FBS*, $12.96\,\mathrm{s} \pm 4.56\,\mathrm{s}$ for *M+RCD*, $45.64\,\mathrm{s} \pm 20.22\,\mathrm{s}$ for *P+FBS* and $26.18\,\mathrm{s} \pm 8.26\,\mathrm{s}$ for *P+RCD*. In addition to reducing segmentation time, our graph reduction approach leads to the highest repeatability between participant performances. This is due to the explicit label positioning constraint that forces participants to focus the drawing while using our segmentation approach.

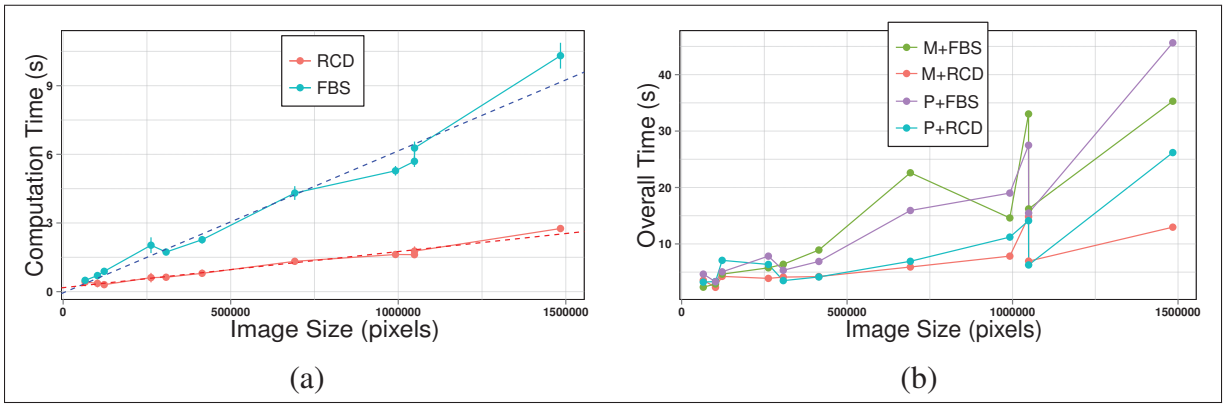

Figure 2.10 Results of the segmentation time according to the image size: (a) Computation time excluding user interaction time, dashed lines represent the linear regression of the data for both FBS (blue) and our approach RCD (red) and (b) overall segmentation time including user interaction time

2.6 Extension to other segmentation algorithms

Having demonstrated the benefits of our approach in the context of RW segmentation through a user study, we now show how these extend to other graph-based approaches and how they compare and can be combined with super-pixel clustering for further improvements in efficiency. A benchmark segmentation is experimented and three key features of our method are highlighted: (i) the benefits of combining our approach to super-pixel-based graph reduction, for example using simple linear iterative clustering (SLIC) (Achanta *et al.*, 2012), are shown

in Section 2.6.1, (ii) the independence of our graph reduction approach with respect to the choice of segmentation algorithm is shown in Section 2.6.2 by extending our approach to the GC (Boykov & Jolly, 2001) and Lazy Snapping (Li *et al.*, 2004) segmentation algorithms, and finally (iii) multi-resolution graph segmentation using multiple super-pixel resolutions. Table 2.2 summarizes the segmentation time obtained for each method. For all the experimented algorithms, using our graph reduction, the segmentation performs faster than the conventional super-pixel graph reduction. Furthermore, our approach achieves slightly better F_1-*Score*s, due to the preservation of full pixel resolution around the boundary (which is not possible with super-pixels). When combined with super-pixel reduction, both RW and GC are accelerated. However, this is not the case for the Lazy snapping segmentation approach, due to the on-line k-means clustering. The remainder of this section provides further details about the experiment results.

2.6.1 Combination with super-pixels

Our graph reduction approach is independent of the image graph. Therefore, super-pixel clustering approaches can be used alongside it to further reduce the graph. To illustrate this, we choose the simple linear iterative clustering (SLIC) super-pixel method, which provides satisfactory results in terms of under-segmentation error (Achanta *et al.*, 2012). A cryosectional image of human anatomy of size 2048×1216 (AnatQuest, 2004) was used for this experiment. The task was to segment the right hand biceps using identical input labels for RW with and without SLIC (resp. for our approach with and without SLIC) approaches (see Figure 2.11.e-f). Table 2.2 (column 1) shows the computation time results of the four segmentation approaches. The experiment was repeated 100 times using the same labels, to account for time lags from external factors. The time to generate 2000 super-pixels using SLIC was 39.391 s. Using our approach, the time required to build the layers from the RCD was 0.551 s. Note that the layers are computed immediately after the user draws the contour. These two computations are only carried out once. However, the time required for super-pixel clustering renders the segmentation inefficient for live applications. The computational bottleneck for RW segmentation is

Table 2.2 Computation time comparison (mean ± one standard deviation) of the conventional segmentation approaches to our graph reduction approach with and without super-pixel pre-segmentation. The *off-line* time indicates the required pre-processing time to build the graphs (i.e., layers and/or super-pixels). Columns represent the method used to solve the segmentation problem, i.e., Random Walker, Lazy Snapping and Graph cuts. Rows represent the graph reduction approaches used during the segmentation, i.e., (i) *conventional* approaches as described in (Grady, 2006), (Li *et al.*, 2004) and (Boykov & Jolly, 2001) respectively, with no graph reduction (ii) conventional approaches used with SLIC super-pixel graph reduction, (iii) *our approach* as described in Section 2.3, and (iv) *our approach* combined with SLIC super-pixel graph reduction. Shaded rows represent the total segmentation time. Best results are showed in bold characters

			Column 1	Column 2	Column 3
		Off-line	**Random Walker**	**Lazy Snapping**	**Graph Cuts**
Conventional		–	26.401 ± 0.364	13.235 ± 1.726	3.861 ± 0.167
(i)	- Graph building	–	0.188 ± 0.006	1.880 ± 0.069	1.693 ± 0.140
	- k-means	–	–	1.114 ± 0.033	–
	- solve RW/max-flow	–	25.813 ± 0.356	9.726 ± 1.726	1.595 ± 0.070
Conventional with super-pixels		39.391	1.789 ± 0.028	2.671 ± 0.091	1.068 ± 0.035
(ii)	- Super-pixels	39.391	–	–	–
	- Graph building	–	0.205 ± 0.004	0.604 ± 0.023	0.600 ± 0.020
	- k-means	–	–	1.573 ± 0.077	–
	- solve RW/max-flow	–	1.168 ± 0.029	0.003 ± 0.002	0.00071 ± 0.0
Our approach		0.551	1.352 ± 0.023	1.541 ± 0.068	1.004 ± 0.046
(iii)	- Layers	0.551	–	–	–
	- Graph building	–	0.704 ± 0.019	0.567 ± 0.049	0.530 ± 0.030
	- k-means	–	–	0.530 ± 0.039	–
	- solve RW/max-flow	–	0.038 ± 0.009	0.011 ± 0.0	0.024 ± 0.002
Our approach with super-pixels		39.942	0.822 ± 0.026	3.588 ± 0.189	0.901 ± 0.029
(iv)	- Layers	0.551	–	–	–
	- Super-pixels	39.391	–	–	–
	- Graph building	–	0.417 ± 0.021	0.512 ± 0.020	0.514 ± 0.023
	- k-means	–	–	2.683 ± 0.193	–
	- solve RW/max-flow	–	0.000447 ± 0.0	0.000039 ± 0.0	0.00004 ± 0.0

solving the linear system induced by the unlabelled pixels ($25.813\,\text{s} \pm 0.356\,\text{s}$). Using our approach, this computation is reduced to $1.352\,\text{s} \pm 0.023\,\text{s}$, outperforming RW with SLIC. More-

over, our approach can be combined with super-pixels to further reduce the computation time to $0.822\,\mathrm{s} \pm 0.026\,\mathrm{s}$. However, using super-pixels can lead to segmentation errors in the case of weak boundaries, as illustrated in Figure 2.11.d. Our approach, preserves high-resolution segmentation results near the object boundary (F_1-*Score* $= 0.968$ using our reduction approach vs. F_1-*Score* $= 0.938$ using SLIC reduction). This is important for interactive segmentation, since the user cannot possibly correct segmentation errors that originate in super-pixel creation.

2.6.2 Extensions to graph cut and lazy snapping segmentation algorithms

To show the generalizability of our approach, we applied our graph reduction approach to lazy snapping (Li *et al.*, 2004) and graph cut (Boykov & Jolly, 2001) segmentation algorithms (see Figure 2.12). Recall that lazy snapping is based on the minimization of the cost of the cut, defined by the following energy function

$$E = \sum_{v_i v_j} E_{binary}(v_i, v_j) + \lambda \sum_{v_i} E_{unary}(v_i), \tag{2.10}$$

where $E_{binary}(v_i, v_j)$ expresses the transitional cost between pairwise pixels and $E_{unary}(v_i)$ is determined using the k-means algorithm (see Section 1.5). We empirically set $\lambda = 0.01$ for our experiment. Similarly to Section 2.6.1, we experimented four algorithms based on the lazy snapping segmentation: (i) and (ii) using FBS respectively without and with SLIC super-pixel pre-segmentation, and (iii) and (iv) using our graph reduction respectively without and with SLIC super-pixels. Table 2.2 (column 2) shows the computation time results of the four approaches averaged over 100 repetitions using the same labels. Using our graph reduction approach we achieve a faster segmentation with a better F_1-*Score* (0.976 using our approach without SLIC) than the approaches using super-pixels (0.939 using the conventional approach with SLIC and 0.939 using our approach with SLIC). Although the segmentation performance depends on the labels that are used, we can observe in Figure 2.12.g-h that the segmentation errors originate from super-pixel clustering. Therefore, they cannot be corrected through user input. When combined to super-pixels, our approach achieved the fastest min-cut/max-flow computation (row iv-column 2 in Table 2.2). However, the total segmentation time is slower

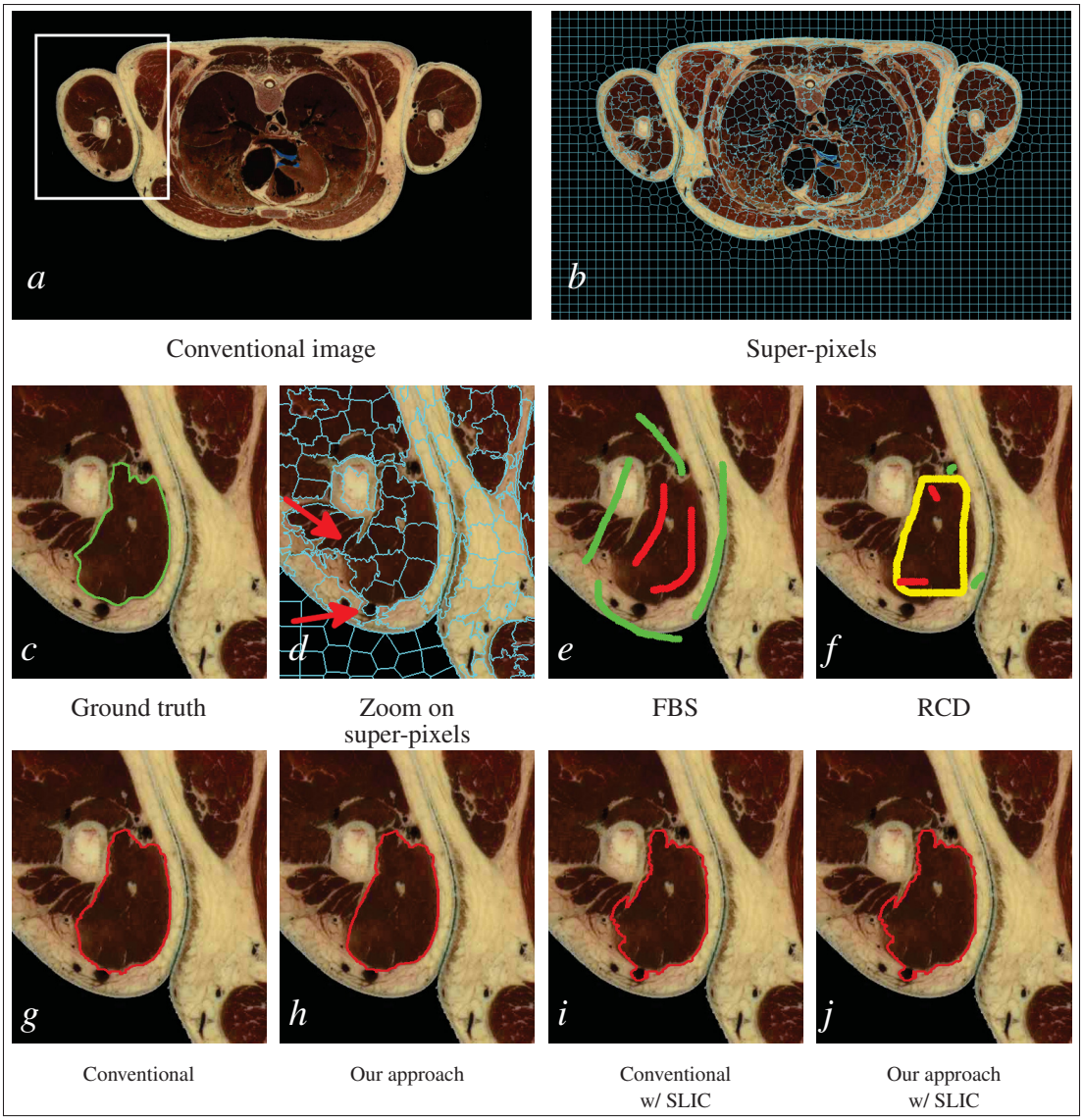

Figure 2.11 Random walker segmentation of the right biceps of a high-resolution cryosectional image: (a) Conventional image, the white rectangle is the region represented in figures (c-j), (b) super-pixel clustering using SLIC, (c) ground truth segmentation, (d) zoom on the super-pixel clustering, the red arrows indicates segmentation errors, (e) foreground and background drawings for RW approaches with and without SLIC super-pixels, (f) rough contour drawings approaches with and without SLIC super-pixels, (g-j) random walker segmentation results using conventional approach, our approach, conventional approach with super-pixels, and our approach with super-pixels, respectively. When using super-pixels, the final segmentation inherits the SLIC segmentation errors

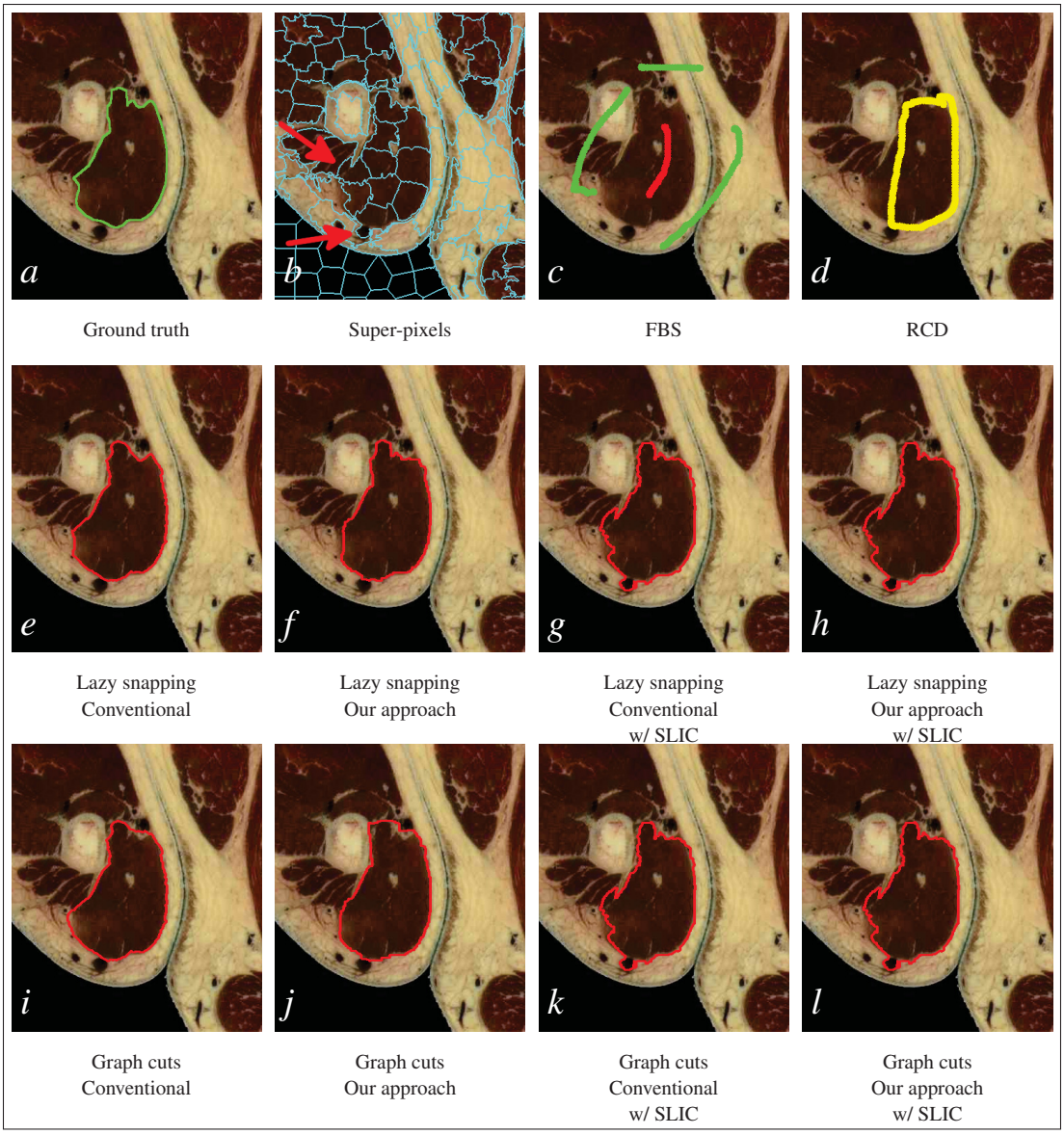

Figure 2.12 Graph cuts and Lazy snapping segmentation of the right biceps of a high-resolution cryosectional image (zoomed): (a) Ground truth image, (b) SLIC super-pixels, (c) Foreground and background drawings for conventional approaches with and without super-pixels, (d) Contour drawings and labels for our segmentation approach with and without super-pixels, (e-h) segmentation results for lazy snapping, (i-l) segmentation results for graph cut (GC). Note that using super-pixels the final segmentation inherits the SLIC segmentation errors

than the conventional approach with SLIC (row ii-column 2 in Table 2.2). This is due to the E_{unary} computation using k-means clustering. Indeed, the more vertices are labelled, the

more vertices are involved in k-means clustering, and the longer k-means takes to converge. Because our approach labels the super-pixels all around the object, more pixels are labelled than using a simple FBS rendering our approach less efficient when combined to super-pixels in this context.

To evaluate the our graph reduction regarding the min-cut/max-flow computation of the GC algorithm (Boykov & Jolly, 2001), we ignore the k-means computation. This is achieved by ignoring the unary energy term of the lazy snapping approach; i.e., this is equivalent to setting $\lambda = 0$ in Equation (2.10). Table 2.2 (column 3) shows the computation time results of the four approaches averaged over 100 repetitions using the same labels. A slightly better computation time is achieved using our approach at both the super-pixel ($0.901\,\mathrm{s} \pm 0.029\,\mathrm{s}$) and the pixel resolutions ($1.004\,\mathrm{s} \pm 0.046\,\mathrm{s}$) than the conventional segmentation using only super-pixels ($1.068\,\mathrm{s} \pm 0.035\,\mathrm{s}$). However, using pixel resolution achieves a better F_1-*Score* with both the conventional GC (0.968) and our graph reduction approach (0.970).

2.6.3 Adaptive multi-scale super-pixels

Rather than simply choosing a *DSL*, the layers described in Section 2.3.1 can be used to combine super-pixel image decompositions at multiple resolutions to build an adaptive multi-scale graph. Figure 2.13 shows an example of a multi-scale graph built from three SLIC super-pixel decompositions, R_1, R_2 and R_3 (containing 100, 1000 and 3000 blocks, respectively), and a full resolution image R_4, using the the algorithm described in Algorithm 2.1. This requires pre-computing each super-pixel image decomposition offline. For the image of size 1024×608 pixels shown in the example, $1.58s$, $7.15s$ and $20.45s$ were required to compute R_1, R_2 and R_3, respectively. The multi-scale graph was constructed in $0.773s$.

This representation provides: (i) a straightforward reduction in graph size, and (ii) an image resolution that gracefully adapts with distance from the boundary. Indeed, compared with conventional super-pixel clustering, the multi-scale graph prevents resolution loss near the object boundary, e.g., errors due to super-pixel segmentation discussed in Section 2.6.1, since the

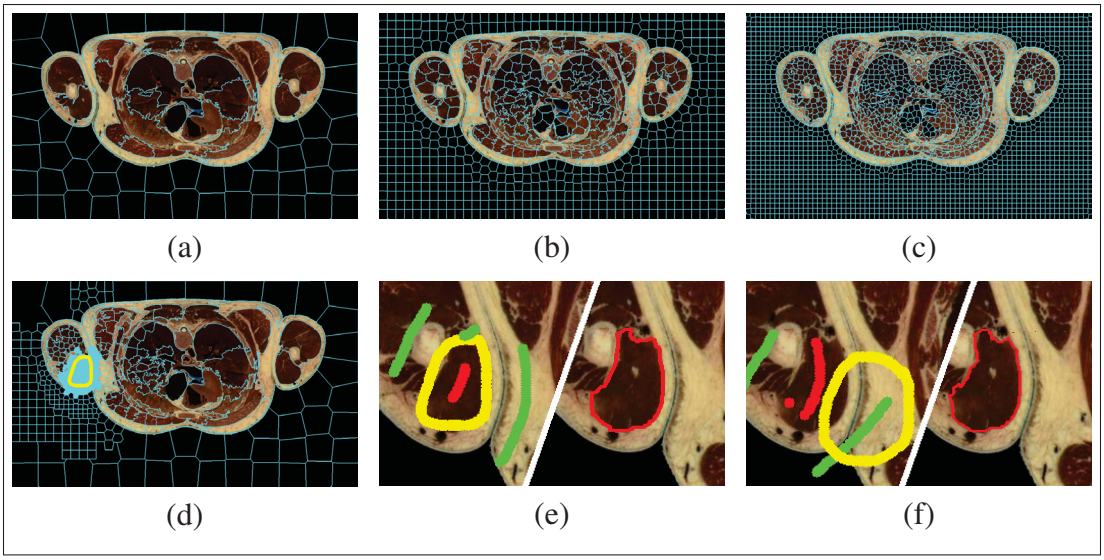

Figure 2.13 Multi-scale graph generation example using super-pixel images: (a-c) super-pixels generated using SLIC with 100, 1000 and 3000 blocks, respectively, (d) multi-scale graph image generated from the contour drawing (yellow) using combination of super-pixel images a-c, (e-f) examples of segmentation using the multi-scale graph

user can interactively adjust the segmentation with a pixel-resolution accuracy. Compared to our initial approach, where pixels beyond the *DSL* are completely ignored, the segmentation is slower, i.e., $0.88s$ using the multi-scale graph vs. $0.41s$ using our initial approach. However, because the information far from the drawings is not completely discarded (it is still available at a coarse resolution), the segmentation is less sensitive to contour drawing positioning (see Figure 2.13.f).

2.7 Conclusion

In this chapter, we proposed a novel graph reduction method based on user drawings for high-resolution interactive image segmentation. In this approach, the user draws a rough contour of the object. Based on a distance map with respect to the drawn contour, image layers are computed such that a pixel far from the contour is assigned to a thicker layer than a pixel near the contour. Then, foreground and background labels are automatically generated, ignoring pixels on the farthest layers. Our approach is based on a hybrid interaction approach combining

Algorithm 2.1 Multi-resolution super-pixels

Input : $R_1, \ldots, R_N$: N super-pixel images
L : Hierarchical layer map // see Fig. 2.1.b
Output: M : Multi-resolution image

```
Initialize matrix M with −1 ;
labelCount ← 0 ;
level ← DSL // from Eq. 2.8
for i ← 0 to N do
    for each label l ∈ R_i do
        // All pixels beyond DSL are assigned to the lowest resolution
        if (∃p ∈ l/L(p) ≥ level) and (i = 0) then
            ∀p ∈ l/M(p) ← labelCount ;
        end
        // Pixels lying on level l are assigned to the same resolution
        if (∃p ∈ l/L(p) = level) then
            ∀p ∈ l/M(p) ← labelCount ;
        end
        labelCount ← labelCount + 1 ;
    end
    level ← level − 1 ;
end
  // Assign the remainder pixels to pixel-resolution
for p ∈ M do
    if M(p) = −1 then
        M(p) ← labelCount ;
        labelCount ← labelCount + 1 ;
    end
end
```

rough contour drawing and foreground-background seeding, benefiting from fast computation and intuitive labelling. Finally, a graph-based segmentation method such as random walker segmentation, is applied on the reduced graph, thereby improving the computation time while preserving full resolution near the object boundary.

The user study reported in this chapter showed that the amount of time the user spends to draw a rough contour leads to a significant gain in overall segmentation time. This is due to the fact that after the initial segmentation, the user focuses his/her effort on small adjustments,

i.e., only few foreground-background labels are required to obtain a satisfactory segmentation. Moreover, the segmentation was experimented with two different devices: a mouse and a tablet with a stylus pen. Surprisingly, although drawing labels using a stylus pen should be more intuitive, segmentation using the mouse is faster. This is probably because labelling-based interactivity for graph-based segmentation requires little drawing accuracy, which is easily reached with the familiar mouse.

Further experiments showed the benefits of our approach both over and combined with graph reduction based on super-pixels and demonstrated its generalization to a variety of graph-based segmentation approaches. Using our graph reduction approach, segmentation is achieved in a time comparable to that achieved with super-pixel graph reduction. However, because our approach preserves full pixel resolution, the user can interactively correct segmentation errors that cannot be corrected using a super-pixel resolution. Moreover, our approach can be combined to super-pixels and achieve even faster segmentation. This is useful for applications where time is more critical than accuracy. Using our approach with super-pixel decompositions at multiple resolutions, we proposed a multi-scale graph construction that adapts to the user drawings. The multi-scale graph segmentation ensures a more flexible user interaction at the expense of slightly more computation time.

Finally, the experiment revealed that only pixels near the object boundary are needed to achieve a satisfactory segmentation, meaning that regions lying far from the contour of the object can be discarded during the computations. This point is exploited in the next chapter, in which we propose a method to automatically infer the relevance of the image pixels using only foreground and background scribbles provided by the user. This reduces the required amount of interaction, as the user is not asked to draw the rough contour at the beginning of the segmentation.

CHAPTER 3

TOWARDS REAL-TIME VISUAL FEEDBACK FOR INTERACTIVE IMAGE SEGMENTATION

3.1 Introduction

The approach proposed in Chapter 2 showed that it is possible to obtain satisfactory segmentation results that preserve the image resolution near the object boundary while reducing the research space. Yet, the proposed approach requires extra effort from the user to roughly draw an additional contour at the beginning of the segmentation process. This has two drawbacks. First, in most cases the rough contour overlaps with the object boundary, causing visual occlusions. This is visually inconvenient when performing corrections using foreground-background labelling. Second, even if the rough contour does not require a high tracing accuracy, it does not allow a dynamic adaptation of the drawings. In other words, in the case of user manipulation errors causing an unsatisfactory contour drawing, the user has to redraw the contour. This mitigates the effectiveness of the interaction mechanism. This chapter addresses these issues by proposing *FastDRaW*, an effective coarse-to-fine interactive segmentation technique designed to achieve fast segmentation. FastDRaW uses regular foreground and background scribbles to perform a fast segmentation on a down-sampled version of the image. The coarse contour resulting from this segmentation acts as the rough contour drawing, described in the previous chapter, to perform a second segmentation on the full resolution image. This prevents the use of any additional user interaction mechanism. We demonstrate the benefits of our approach using random walker (RW) segmentation applied to large images. In addition to computational acceleration provided by the coarse-to-fine approach, we introduce a region of interest selection technique that exploits the user interaction to (i) restrict the object boundary search space based on a relevance map, thereby further reducing the computation time, and (ii) *focus* the segmentation results to reduce the required number of labels to achieve the segmentation. We show how our approach improves the efficiency of the segmentation to achieve real-time

feedback ($\sim$ 100 ms) for images of size 512×512 pixels or less, and interactive response time ($\sim$ 1 s) for images of size 1500×1500 pixels.

The remainder of this chapter is organized as follows. Section 3.2 discusses the concept of interactive activities. Section 3.3 presents the details of our segmentation approach. User study design and results are presented in Section 3.4.

3.2 What is real-time segmentation feedback ?

According to Newell (Newell, 1994), user interactive activities can be classified into three cognitive categories corresponding to their completion time, namely: (i) *deliberate acts* requiring $\sim$ 100 ms, *e.g.,*recognition tasks (ii) *cognitive operations* requiring $\sim$ 1 s, *e.g.,*selecting objects and (iii) *unit task* requiring $\sim$ 10*s*, *e.g.,*editing a line of text. In the context of scribble-based segmentation, the user activity is a drawing task, which falls in the cognitive operation category. Note that the activity also involves a recognition phase, in which the user interprets the intermediate segmentation results before drawing new labels (this will be further discussed in Chapter 4). For now, we consider the drawing action as a single activity represented by the cognitive operation category. The RW approach satisfies the corresponding responsiveness criterion for reasonably sized images, offering convenient interaction between the user and the computer (see Section 2.5.3.2). However, the response time increases with image size, rendering this communication ineffective for large images.

In a more general interactive application context, Miller (1968) described a *threshold* of response time delay that should be satisfied within a communication (either a human-human or a human-computer communication). The author stated "*Conditioning experiments suggest an almost magical boundary of two-second limits in the effectiveness of feedback of "knowledge of results," with a peak of effectiveness for very simple responses at about half a second*". This threshold is subject to the nature of the communication (Card *et al.*, 1991; Verborgh *et al.*, 2016). In this work, we refer to the segmentation as *interactive*, if the response time is shorter than 2 s. On the other hand, movements lasting less than $\sim$ 200 ms are not controlled by vi-

sual feedback mechanisms (MacKenzie, 1992, p. 117–118). Thus, we consider feedback to be *real-time* if segmentation results can be refreshed every 200 ms or less.

3.3 FastDRaW segmentation

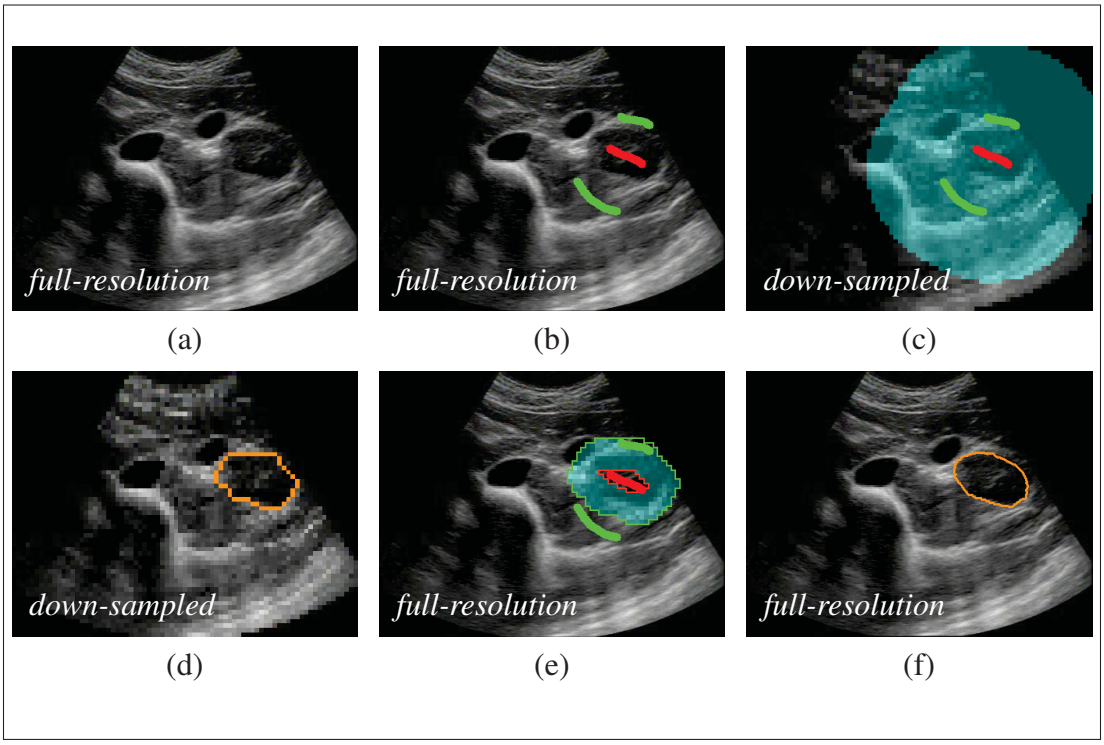

Figure 3.1 Example of segmentation using our multi-scale approach: (a) Original image, (b) image with foreground (red) and background (green) labels (c) label-based extraction of the region of interest (blue), (d) coarse segmentation result (e) refinement strip around the coarse segmentation result with additional foreground and background labels generated on its edges ; the segmentation is only computed on the strip (blue) region and (f) full-resolution segmentation result

Figure 3.1 shows an overview of the FastDRaW approach. Prior to the segmentation, the image is rescaled using a nearest-neighbour down-sampling method and the graph's Laplacian matrices for both the original and the down-sampled images are computed offline. During the segmentation, the user provides foreground and background labels (Figure 3.1b). The positions of these labels are used to extract a *relevance map*, indicating regions on the image where the

object boundary might be located. Then, a region of interest (ROI) is extracted by thresholding the relevance map (Figure 3.1c). The ROI is used to reduce the computation time. Additionally, implicit labels are generated outside the ROI to enhance the user's drawing efficiency. An initial coarse segmentation result is obtained from the down-sampled image (Figure 3.1d). To refine the segmentation, foreground and background labels are automatically placed along the edges of a narrow strip containing the coarse segmentation in the original image (Figure 3.1e). These additional labels are combined with the pre-existing labels to obtain a full resolution segmentation (Figure 3.1f). This section details the key steps of our approach.

3.3.1 Extracting the region of interest

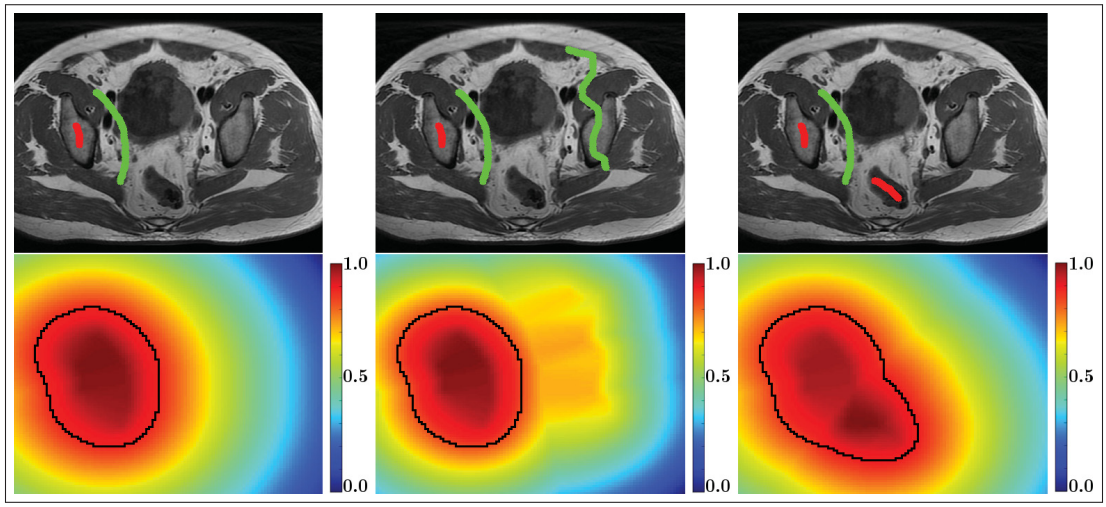

Figure 3.2 Extraction of the region of interest (ROI): relevance maps (bottom) extracted from labels (top), the ROIs are shown in black contour. (left) original labels, (middle) unnecessary background labels in green are ignored, (right) useful foreground labels in red extend the ROI

The ROI is not required to be highly accurate. Therefore, it is extracted on the down-sampled image to reduce the computation time. Our assumption is that the object boundary is more likely to be located somewhere between the foreground and background labels. To achieve this, a relevance map E is computed for every pixel p in the down-sampled image. First, for labels of each category $\ell \in \mathscr{L}$, an Euclidean distance map D_ℓ is computed and normalized such

that $D_\ell \in [0,1]$. Then, E is given by

$$E(p) = 1 - \frac{1}{|\mathscr{L}|} \sum_{\ell \in \mathscr{L}} D_\ell(p) \ \in [0,1], \tag{3.1}$$

for every pixel p of the image. Then, the relevance map is thresholded to obtain a ROI defining the search space. This can be done using a layering approach similar to the one used in Chapter 2. However, because we are operating at a coarse scale, i.e., an undersampled version of the image, preserving full initial resolution near the contour is no longer required and a simple distance-based thresholding is sufficient. Therefore, we consider that the pixel p belongs to the ROI if

$$E(p) \geq \bar{E} - k\sigma_E, \tag{3.2}$$

where $\bar{E}$ and σ_E are the mean and the standard deviation of the values in E, respectively, and $k \in \mathbb{R}$ is a constant parameter, such that increasing k reduces the number of pixels to be selected in the ROI. We empirically set $k = 1$.

Figure 3.2 shows examples of extracted ROIs for different label configurations. The distance maps are computed based on the categories of the labels. Therefore, redundant information provided by labels from the same category generates a low relevance. In contrast, pixels positioned between labels of different categories generate a high relevance, and thus are more likely to be selected in the ROI.

3.3.2 Properties of the ROI

The role of the ROI is to reduce the boundary search region, thereby reducing segmentation computations. Additionally, it implicitly provides supplementary labels for the segmentation algorithm. In fact, during a segmentation computation, pixels outside the ROI are assigned background labels, creating an implicit labelling effect. In the case of RW, it amounts to assigning these pixels zero probability of belonging to the segmented object. If the user updates the labels, the ROI is automatically updated and a new implicit label configuration is used. Figure 3.3 shows the effect of the implicit labelling of pixels outside the ROI.

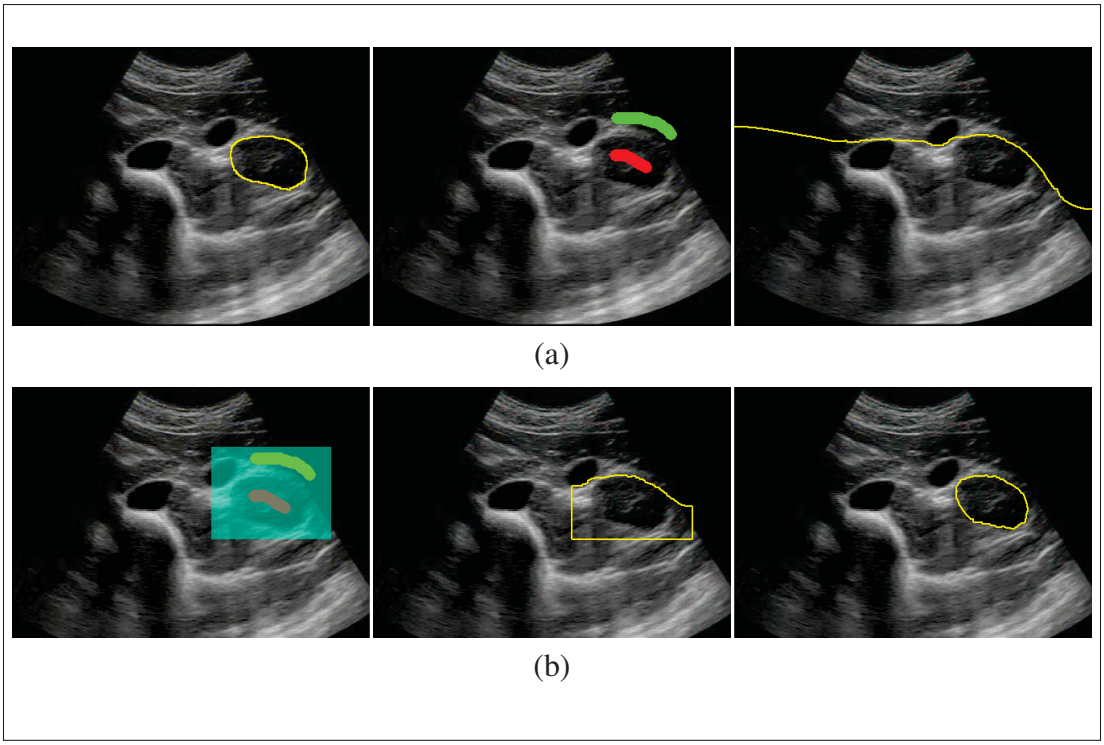

Figure 3.3 Effect of implicit labelling outside the ROI on segmentation results: (a) RW segmentation on the entire image: segmentation ground truth (left), labelled image (middle) and segmentation result (right), (b) RW segmentation on a ROI: (top) pixels on the edges of the image are ignored, (bottom) a small ROI around the object is defined, (middle) segmentation results using L^{Ign} from Equation (3.5) and (right) segmentation results using original L

This effect happens because the topology of the graph changes when considering the computations to be performed only inside the ROI. Precisely, the weights located at the boundary of the ROI are involved in the computations (see Figure 3.4). This induces a bias against the label category that we are solving the segmentation for. This effect can be canceled by removing the vertices that are not inside the ROI and the edges connecting a vertex inside the ROI $i \in \mathcal{V}_{ROI}$ to a vertex outside the ROI $j \notin \mathcal{V}_{ROI}$ (red edges in Figure 3.4).

In the case of RW it is not necessary to rebuild the graph. Recall that the RW solution is given by

$$\mathbf{L_U x_U} = -\mathbf{B^T x_S}, \tag{3.3}$$

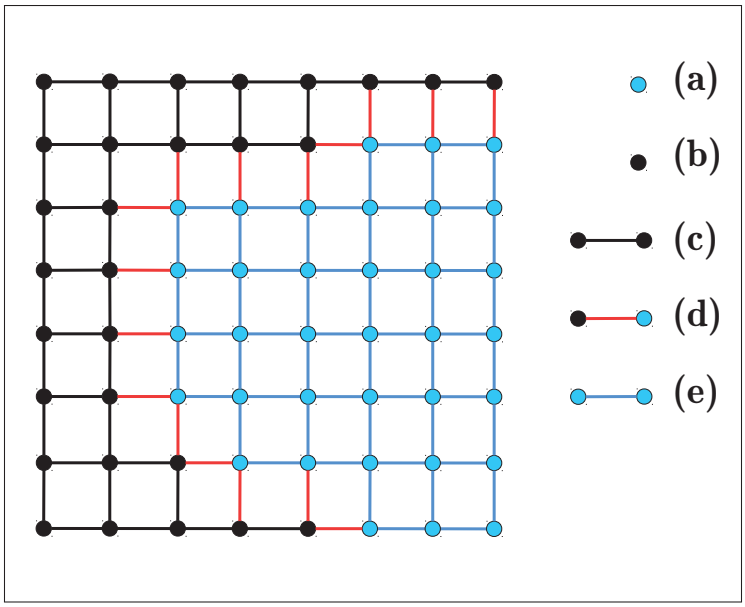

Figure 3.4 Graph topology for ROI selection: (a) vertices inside the ROI, (b) vertices outside the ROI, (c) edges outside the ROI, (d) edges connecting a vertex inside the ROI and a vertex outside the ROI, and (e) edges inside the ROI

where $\mathbf{x_U}$ is the unknown probability vector, $\mathbf{x_S}$ is the probability vector of the labelled vertices, and $\mathbf{L_U}$ and $\mathbf{B^T}$ are submatrices of the graph's Laplacian matrix $\mathbf{L}$ which is given by

$$L_{ij} = \begin{cases} d_i & \text{if } i = j \\ -w_{ij} & \text{if } v_i \text{ and } v_j \text{ are adjacent} \\ 0 & \text{otherwise} \end{cases}, \tag{3.4}$$

where w_{ij} is the weight between connected vertices i and j, and $d_i = \sum_j w_{ij}$ is the degree of the vertex i. Therefore, the effect of the ROI can be cancelled by adjusting the graph's Laplacian matrix $\mathbf{L}$, such that

$$L_{ij}^{Ign} = \begin{cases} L_{ij} & \forall i, j \in \mathcal{V}_{ROI},\ i \neq j \\ L_{ii} - \sum\limits_{e_{ik} \in \mathcal{E}} w_{ik} & \forall i \in \mathcal{V}_{ROI}, \forall k \notin \mathcal{V}_{ROI} \end{cases}, \tag{3.5}$$

where $\mathbf{L}^{Ign}$ is used instead of $\mathbf{L}$ in Equation (3.3). Empirically, we observed that better results are obtained without this adjustment. This is because the object of interest is likely located

inside the ROI, thereby limiting the risk of segmentation error. Thus, in our experiments, pixels lying outside the ROI are simply treated as background. Figure 3.5 shows segmentation examples using the additional implicit labels. Because the object of interest is often included inside the ROI, fewer explicit labels are required to perform the segmentation.

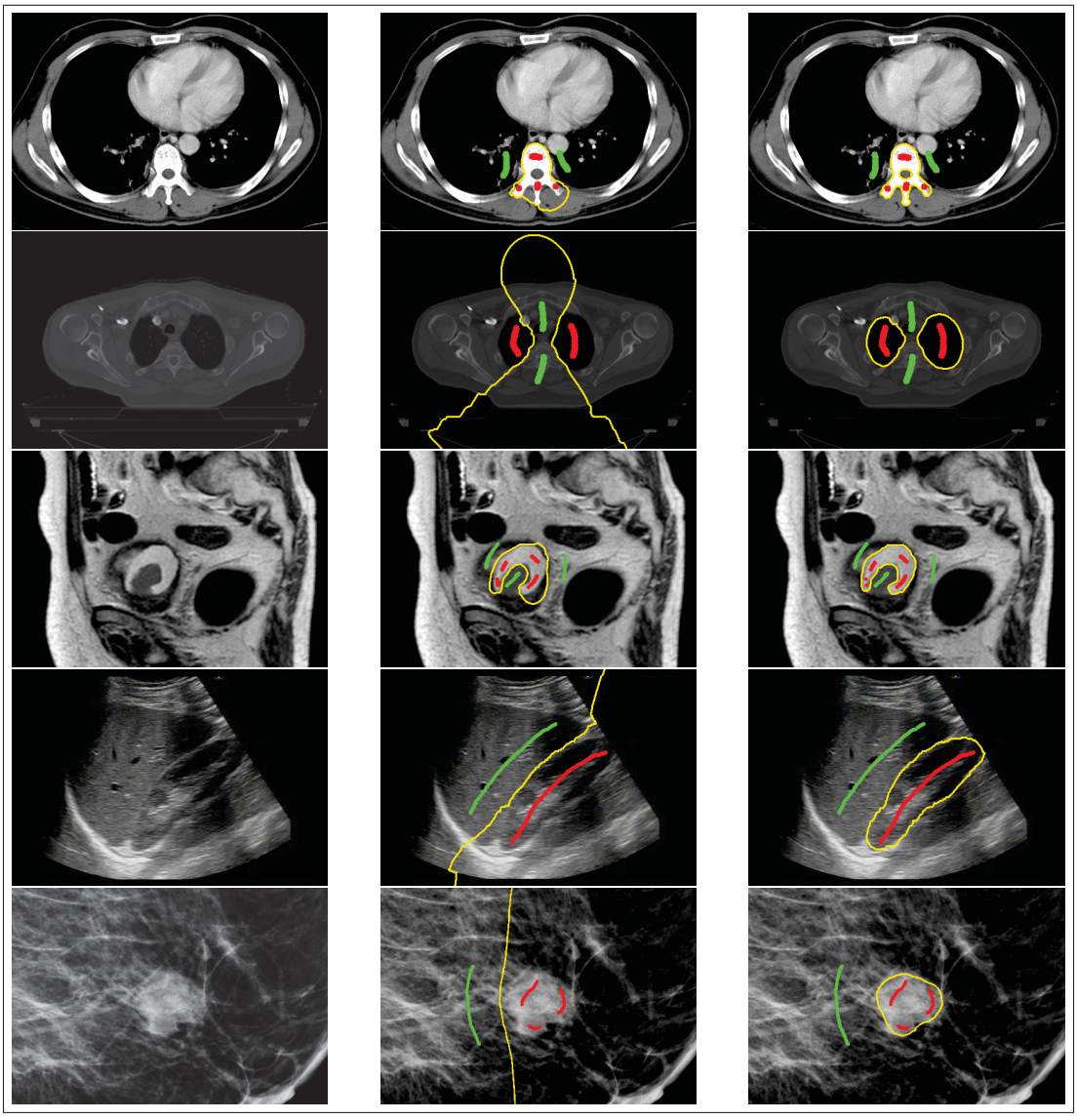

Figure 3.5 Effect of implicit labelling outside the ROI on segmentation results: (left column) the original images, (middle column) RW segmentation results without implicit labels, (right column) FastDRaW results using implicit labels

3.3.3 Segmentation refinement

Once an initial coarse segmentation result is obtained on the down-sampled image, a second RW segmentation is applied on the full-resolution image. This time, the segmentation is only computed on a narrow strip containing the contour (hereafter referred to as the *refinement region*), obtained by thresholding the distance map of the contour (Figure 3.1e). Here, the coarse contour obtained from the down-sampled segmentation result acts as the rough contour drawing used in Chapter 2. However, the contour is obtained without any additional user interaction mechanism. This contour is not displayed on the screen and the user is unaware of it.

By definition, the coarse contour resulting from the initial segmentation separates the image into two or more regions, $R_i, \forall i \geq 2$. Each region R_i contains at most one label category $\ell \in \mathscr{L}$. We assign to each region the label category contained therein, R_i^ℓ. Pixels lying along the edges of the refinement region are labelled according to the label category contained in their neighbouring region

$$\mathbf{x}^\ell(p_{edge}) = 1 \; \forall p_{edge} \in R_i^\ell,$$

where p_{edge} is the pixel lying on the edge of the refinement region and $\mathbf{x}^\ell$ is the probability vector associated to label ℓ. Finally, RW segmentation is performed within the refinement region.

3.4 Results

3.4.1 Implementation details

The Python programming language was used along with the RW algorithm implementation that belongs to the scikit-image open source library. In addition, we adapted the code to optimize the computations as follows: (i) the original RW implementation computes the probability map for every label category, i.e., for a foreground-background image segmentation, the RW core segmentation algorithm is run twice. The code was modified to provide the probability map

only for a selected label category. (ii) the code was optimized to segment 2D images, prohibiting unnecessary computation and assertion of 3D data structures. The FastDRaW implementation was inspired from the core segmentation algorithm of RW, i.e., the computation of the probability map. Additional processing was done in Python to downsample the image using the scikit-image library and compute distance maps using *scipy* library. The user interface was developed using *OpenCV version 2.0*[1] and *Qt4*[2] libraries.

3.4.2 Choice of down-sampling factor

Computing RW segmentation on the down-sampled image results in a decrease in segmentation quality and a major gain of time. To measure the effect of the image size on the segmentation performance and accuracy, we evaluated our approach with 14 images from The Cancer Imaging Archive (TCIA) database (Clark *et al.*, 2013). For each image, 100 label trials were performed using the semi-automatic random path generation method described in Appendix II. For each trial, two segmentations were performed: (i) using the ROI that induces the implicit labelling effect, and (ii) without using the ROI, i.e., the coarse segmentation is computed over the whole image. Figure 3.6.a shows the average computation time as a function of image size. The ROI keeps the segmentation time shorter, thereby affording the use of a higher resolution during the coarse segmentation step. We used the F_1-*Score* as a segmentation quality measure (see Equation (2.9)), using a manual segmentation as ground truth (Figure 3.6.b). Because a sufficient number of background labels were generated around the object, both approaches lead to satisfactory segmentation quality, with a slight improvement when using the ROI. This experiment suggests a down-sampled image of a size between 100×100 and 200×200 pixels to be a good trade-off between segmentation speed and quality. In our experiments, we choose an image size of $\simeq 100 \times 100$ pixels for the coarse segmentation step, while preserving the aspect ratio between the image *height* and the image *width*, such that the shortest dimension is

[1] `http://pythonhosted.org/pyopencv/2.1.0.wr1.2.0/index.html`

[2] `https://wiki.python.org/moin/PyQt4`

100 pixels long, i.e.,

$$size = \begin{cases} 100 \times \left\lfloor 100 \, \frac{width}{height} \right\rfloor & \text{if} \quad width > height \\ \left\lfloor 100 \, \frac{height}{width} \right\rfloor \times 100 & \text{if} \quad width \leq height \end{cases} . \tag{3.6}$$

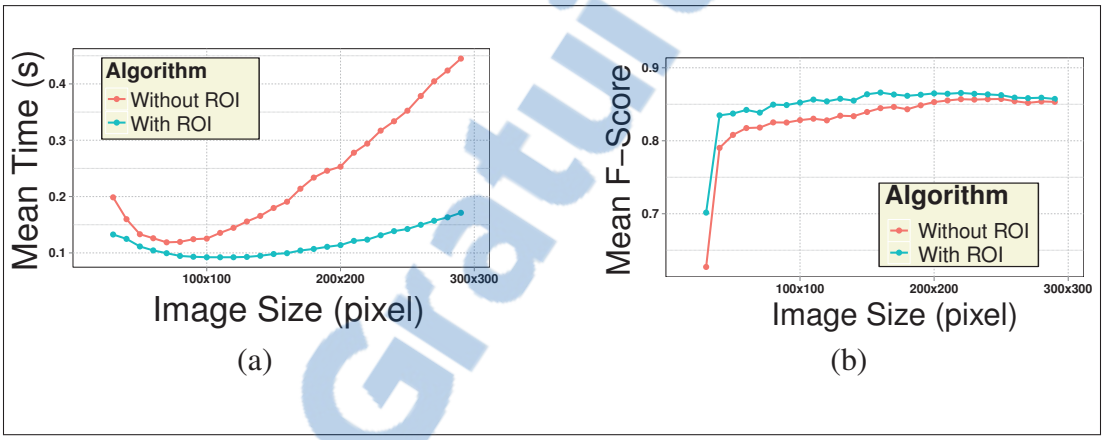

Figure 3.6 Effect of image size on the segmentation results: (a) Average computation time and (b) average F_1-*Score* as functions of image size

3.4.3 User study

Eight participants, selected from undergraduate and graduate engineering programs with no particular expertise in medical imaging, were asked to segment a total of 20 images using RW and FastDRaW approaches. Some of the participants had prior experience with scribble-based segmentation approaches, but none of them had any expertise in medical imaging. Images were sampled from TCIA (Clark *et al.*, 2013) and were organized into two datasets *DS1* and *DS2* of 10 images each. Each dataset contained 5 images of 512×512 pixels referred to as *Medium* images and 5 images of 1500×1500 pixels referred to as *Large* images. Each participant experimented a cross combination of $\{RW, FastDRaW\} \times \{DS1, DS2\}$ in a counterbalanced order with half the participants starting with *DS1* and the other half with *DS2*. During the segmentation, the ground truth was displayed in a separate window and participants were asked to provide similar segmentation results based on qualitative visual appreciation. Using FastDRaW,

the results were displayed as soon as they were available, providing immediate visual feedback to the user about the impact of new labels. However, for RW, the computational burden renders such immediate updates ineffective, as will be demonstrated in Chapter 4. The participants thus had to launch the segmentation algorithm by pressing a key with their non-dominant hand. We assume that this extra burden in the user interface naturally results in participants minimizing the number of segmentation iterations for the RW algorithm. An Intel© Core™ *i*5-2500 at $3.30\,GHz \times 4$ with $4\,Gb$ RAM was used for the processing.

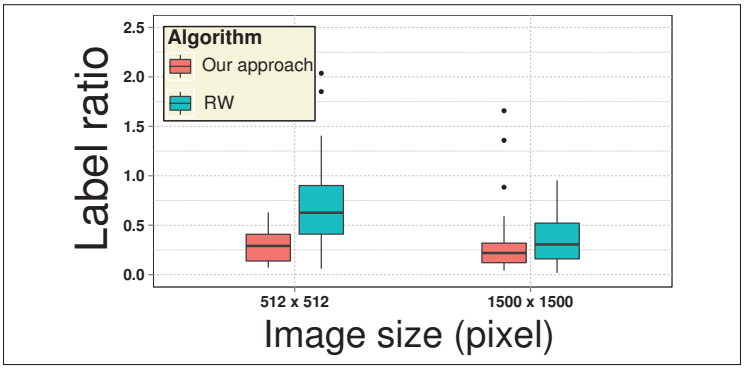

Figure 3.7 Number of labelled pixels per image normalized by the ground truth object size

Table 3.1 Results of the overall segmentation time, the computation time and the labelling time for RW and FastDRaW

		Algorithm	
	Size (pixels)	RW	FastDRaW
Overall time (s)	512×512	25.25 ± 2.28	$\mathbf{17.08 \pm 1.96}$
	1500×1500	79.98 ± 7.02	$\mathbf{43.58 \pm 3.31}$
Computation time (s)	512×512	1.23 ± 0.03	$\mathbf{0.13 \pm 0.003}$
	1500×1500	10.18 ± 0.25	$\mathbf{1.18 \pm 0.04}$
Labelling time (s)	512×512	$\mathbf{0.39 \pm 0.05}$	0.68 ± 0.19
	1500×1500	$\mathbf{3.60 \pm 0.58}$	5.82 ± 0.77

The results are summarized in Table 3.1. A repeated measures analysis of variance test revealed that the overall segmentation time is significantly improved using FastDRaW ($p < 0.01$) for

both Medium and Large images. For Medium images, the average computation time for FastDRaW to provide each intermediate segmentation result was $0.13 \pm 0.003\,s$, achieving real-time segmentation according to the definition in Section 3.2. This is sufficiently fast to allow visual updates of the segmentation results as the user is drawing labels. For the unmodified RW algorithm the segmentation was achieved at interactive speed and updating the segmentation every $1.23 \pm 0.03\,s$ would be feasible; however, this would affect user interaction. This can be observed when segmenting Large images using FastDRaW approach. Indeed, during the experiment, the time required by participants to draw labels was measured. For larger images, there was an increase in the time spent by participants for labelling, both for the RW and FastDRaW algorithms, with a significantly slower time for FastDRaW. This is due to participants' labelling movements slowing down while waiting for segmentation feedback, which is not the case for RW where labelling and computation were two separate actions. However, even if it is slower, a continuous update reduces the number of labels required to achieve an accurate segmentation (see Figure 3.7). In fact, participants stopped labelling as soon as a satisfactory result was reached (5981.20 ± 722.46 labelled pixels using FastDRaW), thereby avoiding unnecessary labels (7778.48 ± 990.42 labelled pixels using RW). On the other hand, the variance of the average number of labelled pixels is significantly reduced when using our approach ($p < 0.01$). This could be due to the additional implicit labels introduced by the ROI, meaning that all images required a similar amount of effort from the user to complete the segmentation task.

3.5 Conclusion

This chapter presented a multi-scale approach for interactive graph-based segmentation that allows real-time visual feedback during the segmentation. The method under-samples the original image and reduces the boundary search space based on the relevance of the labels to obtain an initial coarse segmentation. This segmentation is then refined within a small region of the full resolution image. This approach significantly reduces computation time, thereby allowing the real-time display of segmentation updates as the user labels parts of the image. A

preliminary user study was conducted to assess the effects of this on the interactive segmentation process. We compared our approach to conventional interactive RW segmentation where labelling and computations are separate actions controlled by the user. Computations on images of size 512×512 pixels, a conventional format in medical imaging, provided real-time segmentation updates, whereas interactive visual feedback was achieved for images of size 1500×1500 pixels.

Although the results suggest that the proposed approach drastically improves the computation time, the overall time required to complete the segmentation did not achieve commensurate reductions. In addition, real-time feedback yields a drawing with fewer labels, while slowing down the user's actions. This points to the influence of human factors in interactive segmentation tasks. In the next chapter, we investigate the effects of the response time and visual feedback on the user performance. This will help to understand how to best leverage the benefits of such computational improvements on the overall segmentation task.

CHAPTER 4

THE EFFECT OF LATENCY IN VISUAL FEEDBACK ON USER PERFORMANCE

4.1 Introduction

During an interactive segmentation task, the user modifies the inputs according to intermediate results, creating a *query-feedback loop* that ends when a satisfactory result is achieved. Because it is the *user* who judges what inputs to give and when the result is satisfactory, this process is strongly influenced by human factors. Previous evaluations of interactive segmentation have focused on algorithmic runtime (Udupa *et al.*, 2006; Singaraju *et al.*, 2009; Gulshan *et al.*, 2010), with few studies of human factors (Olabarriaga & Smeulders, 2001; West *et al.*, 2016; Ramkumar *et al.*, 2016). In our preliminary study conducted in Chapter 3, we found that improvements in computational performance did not yield a commensurate improvement in *overall* segmentation performance, i.e, the total time including user actions. Although computation time was improved by a factor $\sim$10 (from 1.23 s to 0.13 s), overall segmentation time was only improved by a factor $\sim$2 (from 25.25 s to 17.08 s). Still, the approaches tested led to results with similar accuracy. These results point to the importance of assessing the user's role in interactive segmentation processes (Olabarriaga & Smeulders, 2001; McGuinness & O'Connor, 2010). In turn, the impact of the user depends on his/her degree of involvement during the segmentation task. The goal of this chapter is to provide insight into the factors that affect the *user's performance* during an interactive segmentation task.

We report an experiment that manipulates the delay induced by computation time, i.e., the time elapsed between the query generated by the user's inputs and the response provided by the segmentation algorithm. This delay, referred to as the *feedback latency*, characterizes the computational efficiency of a segmentation method. Feedback latency is unavoidable in interactive applications and often has significant repercussions on the user's performance (Liu & Heer, 2014; Jota *et al.*, 2013; Beigbeder *et al.*, 2004). Previous work outside the context of image segmentation has studied the effect of latency on the user performance in interactive appli-

cations. For example, Ware & Balakrishnan (1994) studied the effect of lags in a Fish Tank virtual reality application (Ware *et al.*, 1993), where the task consists in reaching for a target in 3D using a head coupled stereo display and a hand tracking device. The authors investigated different types of latency, involving a blended latency between the head tracked stereo display and the hand device and a separate latency between these two devices. In addition to the decrease in user performance with the increase of the latency, the study revealed that a better performance could be achieved by separating head latency from hand latency. Liu & Heer (2014) investigated the effect of latency in the context of exploratory visual analysis, where the task is to interactively explore a database. The study concluded that a 500 ms latency could induce significant changes in the way the user explores the data, reducing his/her activity and observation performance. Jota *et al.* (2013) reported a study comparing direct and indirect pointing tasks in touch-based interactive systems. The authors showed that the latency is perceived at different levels depending on the task performed (dragging or tapping tasks) and the interaction mechanism involved (direct or indirect mechanisms). The user was more sensitive to the latency in a dragging task than in a tapping task; and similarly, he/she was more sensitive when using a direct mechanism than when using an indirect mechanism.

The latency affects the user in different ways depending on the interaction mechanism, the task to accomplish or the nature of the application. In an interactive image segmentation task, the effects of latency are unknown. For example, to what extent does the feedback latency decrease the user performance during a segmentation task? Is there an interaction mechanism design that reduces the effect of this latency on the user performance? What is the recommended latency, if it exists, below which the user interaction is no longer affected?

In this chapter, as in the rest of this thesis, we studied *scribble-based* segmentation. Recall that in this approach, the user drags the mouse using one of two buttons to “paint” either foreground or background labels on pixels. In response, the system recomputes and displays the segmentation. Scribble-based approaches allow a direct mapping between the image and the position of the labels, which has been successfully used within a wide range of segmentation algorithms and applications (Boykov & Jolly, 2001; Grady, 2006; Protiere & Sapiro, 2007; Rother *et al.*,

2004; Gulshan *et al.*, 2010). Further, while contour-based approaches constrain the user's actions to reproduce (albeit roughly) the object boundary drawing, the scribbling paradigm offers much more freedom by allowing a much broader variety of valid segmentation inputs, which motivates our study. Moreover, this freedom has significant impacts on the user's behaviour depending on the shape, the position and the order in which the labels are drawn. When the user observes the updated results during the segmentation task, he/she may adapt the drawings differently, leading to very diverse scenarios. Therefore, the conditions under which the user receives visual feedback influence the segmentation process.

This chapter investigates the effects of the visual feedback latency and timing on the user's performance during a scribble-based interactive segmentation task. A user study is carried out that consists of experimenting two techniques for refreshing visual feedback under different levels of latencies from 100 ms to 2 s. Flowcharts of the two refresh conditions are illustrated in Figure 4.1. The first technique is an *automatic* refresh method in which the user is continuously shown segmentation results as soon as they are made available during label drawing. In the second technique, the user manually initiates the segmentation each time he/she wants to visualize the results. Segmentation time, accuracy and drawing measurements are recorded during the segmentation. Our contributions are summarized in the following findings:

- The latency is perceived differently depending on the refresh method used during the segmentation task.
- Regarding the user performance:
  - The user-initiated refresh method allows to reduce effects of the latency on the user's performance;
  - The user is sensitive to the latency condition when the automatic refresh method is used. This sensitivity gradually decreases as latency increases;
  - Around ~ 2 s of latency time: the user performance seems to converge towards similar behaviours for both refresh conditions.

- Regarding the segmentation performance:
  - Overall, users performed better using the user-initiated refresh method;
  - Below ~ 350 ms of latency time: both refresh methods yield similar segmentation performance.

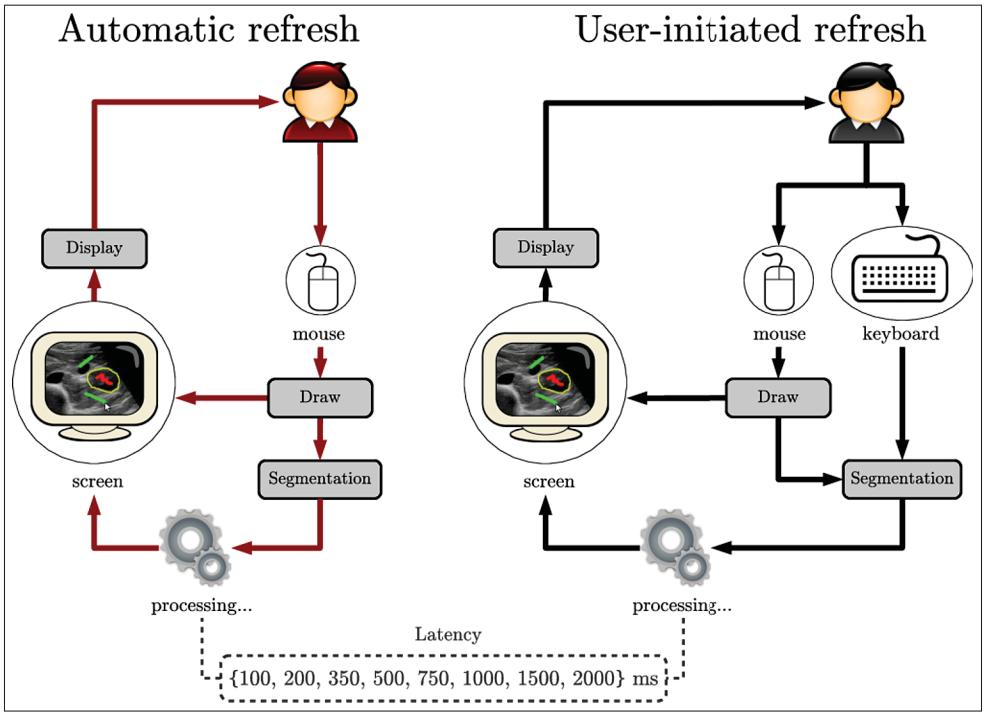

Figure 4.1 Illustration of the two scenarios of the refresh conditions

4.2 Background

4.2.1 Latency in interactive applications

The 2 s response time threshold suggested in the literature for efficient user-computer communication (Miller, 1968) is subject to change according to the nature of the task. The human visual system is highly efficient for cognitive tasks. It has been reported that the visual system can distinguish *comprehensive* content in images displayed for 13 ms (Potter *et al.*, 2014).

However, even if the user is able to understand the visual content, the complexity of the mechanism involved during human interaction delays his/her reaction by 100 ms to 200 ms (MacKenzie, 1992, p. 117–118). Moreover, according to Miller (1968), human activities are naturally organized into groups of actions, named *closures*, that are determined by the achievement of subjective purposes. The user is more sensitive to a latency occurring within the same group of actions, referred to as *within-activity* latency, than to a latency occurring between two groups of actions, referred to as *between-activity* latency. For interactive image segmentation, we typically identify two groups of activities. (1) The *query* actions, in which the user provides the inputs. For example, this is represented by dragging the mouse to draw labels in a scribble-based segmentation approach. (2) The *feedback* actions, which consists in the cognitive activity where the user receives and interprets the results. Depending on when the latency occurs in the segmentation process, it may affect the user differently. Therefore, it is important to consider the user-interaction mechanism involved during the segmentation task. In our experiment, the two types of latency are exemplified by the refresh methods. In the first scenario, the segmentation updates are automatically displayed on the screen. The query and the feedback actions are confounded, leading to the occurrence of within-activity latencies during the segmentation task. In the second scenario, the segmentation updates are controlled by the user, dissociating the query and the feedback into two distinct groups corresponding to different actions. Hence, the delay caused by the computations is considered to be a between-activity latency.

4.2.2 Interactive segmentation assessment

The user plays a significant role in the interactive segmentation process. Yet, when it comes to the assessment of interactive segmentation methods, it is common in the literature to emphasize the computational aspects of the process without regards to the user's performance. Three factors are commonly used to assess segmentation methods (Olabarriaga & Smeulders, 2001): (i) accuracy assesses the similarity between the segmentation result and the ground truth, (ii) repeatability assesses the precision reached by the segmentation approach, and (iii) efficiency assesses the effort required to complete the segmentation task. However, to design

a segmentation assessment framework, it is important to consider the context of the segmentation task (Udupa *et al.*, 2006). The nature of the task, the type of images and the application domain influence the segmentation outcome and the user behaviour. In most cases, the goal is to compare different segmentation methods. Conducting a user study is the most common way to account for the user's performance in interactive application assessments (MacKenzie, 2013). User studies have been used in interactive segmentation contexts to evaluate the performance of different human-computer interactions (HCIs) (Top *et al.*, 2011; Lefohn *et al.*, 2003; Sadeghi *et al.*, 2009). McGuinness & O'Connor (2010) proposed a unified platform for interactive segmentation assessment. However, heterogeneous HCIs often involve significant changes in user interface, requiring a specific evaluation platform. Focusing on the use of HCI in the context of 3D image segmentation, Ramkumar *et al.* (2016) conducted a user study to compare contour-based and scribble-based approaches. The study concluded that the tested region-based segmentation approach yielded a slightly more effective segmentation time, while the contour-based approach was less frustrating. Objective and subjective metrics were recorded, allowing the comparisons in terms of computational performance and user appreciation, respectively.

Objective metrics, such as computation time and accuracy of the segmentation results, are reliable tools to assess the computational aspects of a segmentation method. In contrast, subjective metrics, often obtained through forms filled by the participants at the end of the experiment, are typically used to assess the cognitive aspects. However, some dimensions of the user's cognitive and behavioural performance during the segmentation can be measured objectively. Hebbalaguppe *et al.* (2013) attempted to measure the attention effort produced during a segmentation task by analyzing the user's electroencephalogram (EEG) signal, recorded during the task. They found that *the effort* (i.e., related to the EEG signal of the brain) produced by the user can be reduced by using an additional bounding box interaction mechanism. In order to assess the performance of a haptic interface in 3D segmentation, Harders & Szekely (2003) conducted experiments by adapting a model based on Fitts' law (Fitts, 1954) and steering law Accot & Zhai (1999), which describes a formal relationship between speed and accuracy in

aimed movements. The porposed haptic system was used to initialize a deformable model for the segmentation of 3D tubular structures. The study reported a better accuracy and segmentation time when the user was provided haptic feedback.

While the aforementioned works focus on comparing the performance of segmentation methods in terms of user interaction, this chapter aims at investigating user behaviour under different response times using a standard user interface. Based on the suggestions documented in (Udupa *et al.*, 2006; Olabarriaga & Smeulders, 2001), we designed a user study to characterize user behaviour during the completion of a scribble-based segmentation task.

4.3 Experiment

To evaluate the effect of latency on a scribble-based interactive segmentation task, we carried out a controlled user study. This section details the study design for the experiment.

4.3.1 Preparing the image dataset

A dataset of 80 images (250×250 pixels) was prepared. The images were carefully selected from *the cancer imaging archive* public database (Clark *et al.*, 2013), to which we added samples from our own collection. Similarly to Chapter 2, the database includes samples from computed tomography (CT), magnetic resonance (MR), X-ray and ultrasound (US) images, representing different anatomical structures. All the ground truth data were obtained by manually segmenting the images. The dataset was then partitioned into 8 non-overlapping subsets $D_{i=1\ldots8}$ of 10 images each, to avoid the carryover effect caused by segmenting the same image multiple times. The order in which the images within a dataset appear to the user is randomly shuffled. An additional dataset, $D_{training}$, of 21 images was similarly prepared to serve as a training dataset, such that $D_{training} \cap D_i = \emptyset, \forall i = 1 \ldots 8$.

4.3.2 Study design

Two factors were investigated. The first is the *Latency* factor, which is the time elapsed between the drawing query and the segmentation response, i.e., the delay before displaying the update on the screen. Eight Latency conditions were tested, $\mathscr{L} = \{100, 200, 350, 500, 750, 1000, 1500, 2000\}$ (ms). The second factor is the *Refresh* method and was designed to capture between- and within-activity latency types. Two conditions were tested $\mathscr{R} = \{Automatic, User\text{-}initiated\}$. In the automatic refresh condition, the updates were automatically displayed on the screen after the latency time elapsed. Here, the latency condition acted as within-activity latency. In the user-initiated refresh condition, the updates were displayed on the participant's demand by pressing a button on the keyboard. Once initiated by the user, the results are displayed after the latency time elapsed. Here, the latency condition acted as between-activity latency.

A total of eight participants were recruited from engineering undergraduate and graduate programs. Some of the participants had prior experience with scribble-based segmentation approaches. Every participant tested all the latency conditions and all the refresh conditions. The order in which the latency conditions were tested was counterbalanced according to a 8×8 Latin square design, i.e., one latency condition, $l_i \in \mathscr{L}, \forall i = 1 \ldots 8$, was tested on a single dataset $D_{j=1\ldots8}$, such that all combinations $\{l_i, D_j\}$ were tested by the eight participants. Therefore, each participant tested all the latency conditions with the automatic refresh method in a first experiment. Then, he/she tested all the latency conditions once again with the user-initiated refresh method in a second experiment. However, since the same image datasets are used for both experiments, such an ordered design could bias the participant's performances. To reduce the risks that a participant remembers the images and their associated labels, the second experiment was conducted at least two weeks after the user's participation in the first experiment.

Each experiment of a given refresh condition involved eight rounds of two successive steps: a training step followed by an evaluation step. The participants performed the training and

the evaluation with the same latency condition on the first dataset, then the training and the evaluation with the next latency condition on the second dataset, and so on. In total, there were 8 participants $\times$ 8 latencies $\times$ 2 update refresh methods $\times$ 10 images per dataset $=$ 1280 segmentation trials.

4.3.3 Experiment progress

4.3.3.1 Training step

During the training step, no data were recorded. The goal of the training was two-fold. First, it aimed at preparing the participant to understand how the scribble-based segmentation approach works. Second, it acted as a buffer between two successive conditions to accustom the participant for the next latency condition. During the training step, 10 images were randomly selected from the training dataset $D_{training}$. The participant had to segment all the 10 images under a given latency condition before proceeding to the evaluation step under the same condition. For each segmentation trial, the ground truth was provided and displayed beside the original image. In order to instruct the participant on the amount of accuracy required for the segmentation, the accuracy score of the current segmentation result was displayed on the top right of the screen, during the training step. An acceptable score was considered to be of 0.90 or above, before ending the trial and moving on to the next image.

4.3.3.2 Evaluation step

During the evaluation step, the time, the accuracy and efficiency of the segmentation were measured according to the parameters described in Section 4.4. Similarly to the training step, the ground truth was provided to the participant, indicating the anatomical structure to segment. This information compensates for the lack of medical image interpretation skills of the participants. However, during the evaluation step, the segmentation accuracy score was not shown. The participant was asked to perform the most accurate segmentation according to his/her ap-

preciation with respect to the ground truth in the minimum possible time. The evaluation step ended when the 10 images of the dataset D_i were segmented.

4.3.4 Interaction mechanism

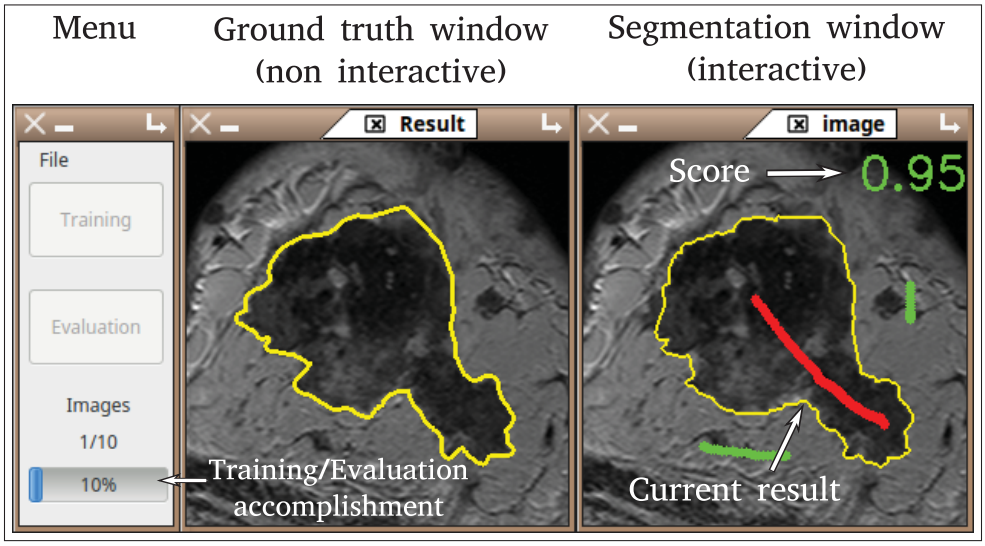

Figure 4.2 The user interface in our study

The user interface was designed in Python using the OpenCV and Qt4 libraries. It was based on a standard 2D window-icon-menu-pointer (WIMP) paradigm for minimal user-computer interaction (see Figure 4.2). It involved two menu buttons to switch between training and evaluation steps, and two windows to display the image and its ground truth. The user's actions were restricted to: (i) clicking and dragging the mouse to draw foreground and background labels; and (ii) undoing the last drawings, using a keyboard button. For the user-initiated refresh condition, the user had to press a keyboard button, with his/her non-dominant hand, for every desired segmentation update. The automatic refresh condition did not require any additional interaction. Once the results were satisfactory, the user ended the segmentation using a dedicated keyboard button. The next image and its ground truth were then automatically loaded on the screen. Any additional user interaction, such as zooming/panning, loading images and

resizing the thickness of the drawing brush, were prohibited, due to the unnecessary cognitive load they impose on the user.

During the segmentation, the user drew foreground and background labels and the result was updated on the screen. The segmentation computations were indicated using the mouse's waiting cursor. However, the participants were allowed to draw new labels while the computations took place. Hence, for both automatic and user-initiated refresh conditions, the drawing and the computations were separate processes. For example, for a 2000 ms latency, participants were able to anticipate the segmentation result by drawing additional labels before the computations were completed. In addition, the mouse acceleration was disabled to prevent any effect caused by speed variability of the mouse.

4.3.5 Segmentation method and computations

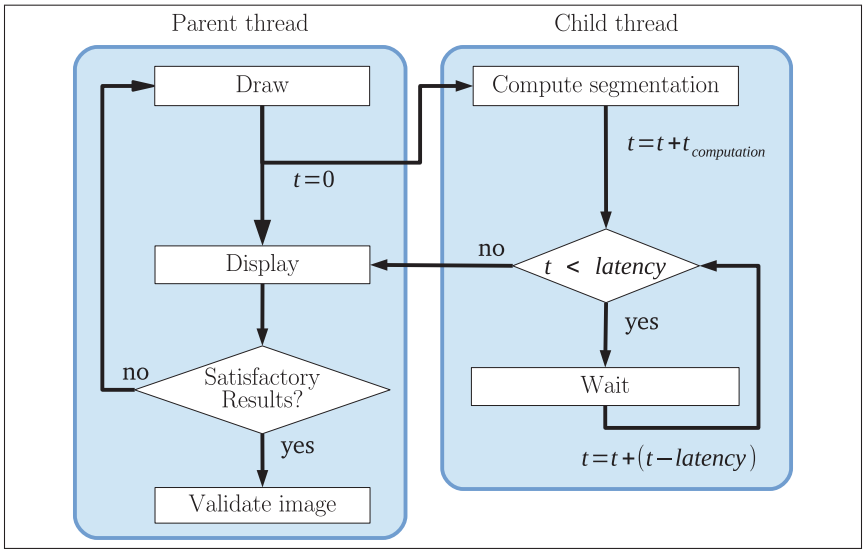

Figure 4.3 Workflow of the interactive segmentation software used for experiments. The drawing and segmentation computation are processed in different threads, thereby allowing user interaction during computations. The elapsed time t is used to control the latency of the segmentation

In this study, all processing was done on an Intel© Core i7-2630QM 2GHz×4 machine with 4Gb of RAM. To avoid holding the drawing resources during computations, all image processing operations were run in a separate thread (see Figure 4.3). The FastDRaW segmentation method, developed in Chapter 3, provides segmentation results in ~ 90 ms for images of size 250×250. Therefore, we set the lower latency limit tested in our experiments to 100 ms. Larger latencies were simulated by constraining the processing thread to wait the remaining amount of time before displaying the result on the screen. We considered a binary segmentation task of a single object per image, i.e., the user was allowed to draw only foreground and/or background labels.

4.4 Measures

During the evaluation step of the segmentation task, six metrics were recorded to capture the user's performance in terms of: (1) the time required to perform a segmentation, (2) the time required to label an image, (3) the speed of the drawings, (4) the accuracy of the segmentation, (5) the continuity of the drawings, and (6) the number of labels drawn. This section describes these measures.

4.4.1 Overall time - t_Ω

The overall *time* elapsed during the segmentation of one image was measured for each trial. Because of the diversity of the images, the time required to achieve the segmentation could vary significantly from one image to the next, regardless of the participant. Therefore, we used the average time required to segment a dataset D_i using a given latency condition, noted t_Ω, where $i = 1 \dots 8$ is the dataset index.

4.4.2 Labelling time - t_Λ

During the segmentation, the time taken to draw the labels was recorded. This measure involves the sum of elapsed times between the moment the user presses then releases the mouse buttons

to draw a foreground or background label, for a single image. The labelling time, noted t_Λ, is the average time recorded per image over each dataset D_i under a given latency condition.

4.4.3 Drawing speed - υ

The user labels the image by drawing scribbles using the mouse. A scribble is drawn by pressing and holding the mouse's button down while moving the mouse through the image. The speed at which the user moves the mouse between the moment the button is pressed and released indicates how fast the scribbles are drawn. This drawing speed was computed by dividing the distance travelled by the time elapsed between these two moments, and is given in $pixel/s$. The goal of this metric is to observe how the user draws the scribbles. In fact, we hypothesize that a large cognitive load slows down the user's drawings. The drawing speed υ is given by the average speed recorded for a dataset D_i using a given latency condition.

4.4.4 Accuracy - $\mathscr{A}$

Similarly to Chapter 2, we use the F_1-*Score* metric to assess the accuracy of the segmentation results. The F_1-*Score* is given by Equation (2.9) and measures the agreement between two samples of binary data. Here, $\mathscr{A}$ denotes the average accuracy achieved in a dataset D_i using a given latency condition.

4.4.5 Continuity of the strokes - ζ

The *continuity* ζ is the number of labelled pixels per mouse click in one segmentation trial. Because the drawing is performed by maintaining the button of the mouse pressed, this measure expresses the average number of pixels labelled in one mouse stroke. Therefore, a large value indicates that the labels were drawn continuously, i.e., in the form of long strokes. A discontinuous drawing can be caused by two events. Either the user draws multiple strokes of the same label category, or the user alternates between drawing foreground and background labels. We hypothesize that a continuous drawing is performed during the same action, and

may be associated to a single thinking process on the part of the user. The continuity measure characterizes the way the drawings were accomplished.

4.4.6 Number of labels - $\mathscr{N}$

When a segmentation trial was completed, the total number of labelled pixels was computed. Complex segmentation tasks would require more labels than simple ones. Therefore, the number of pixels labelled is an indicator of the amount of effort produced by the user during the segmentation task. However, the segmentation of large objects also requires more labels than the segmentation of smaller objects. Calculating the average number of labels per dataset would produce a high variability. Instead, we used the sum of the labels required to segment an entire dataset, noted $\mathscr{N}$.

4.5 Results

Figure 4.4 summarizes the results obtained by all the participants for both the automatic refresh experiment and user-initiated refresh experiment.

4.5.1 Overall time

The overall segmentation time results, including the latencies, are shown in Figure 4.4.a. For the automatic refresh method, the overall segmentation time increases non-linearly with the latency. The impact of the latency on the segmentation time becomes less significant as the latency increases. In fact, for latencies between 100 ms and 500 ms, the segmentation performance slows down quickly, from an overall segmentation time of $t_{\Omega}^{100} = 23.86\,\text{s} \pm 1.37\,\text{s}$ to $t_{\Omega}^{500} = 29.69\,\text{s} \pm 1.89\,\text{s}$ per image, respectively, which represents a slope of 12.46 ± 0.48. This impact is attenuated for latencies between 750 ms and 2000 ms to reach a slope of 2.79 ± 0.19.

Compared to the automatic refresh method, the segmentation task using the user-initiated refresh method was completed in similar time for latencies between 100 ms and 350 ms, and shows faster performances for larger values of latency. We draw the reader's attention to the

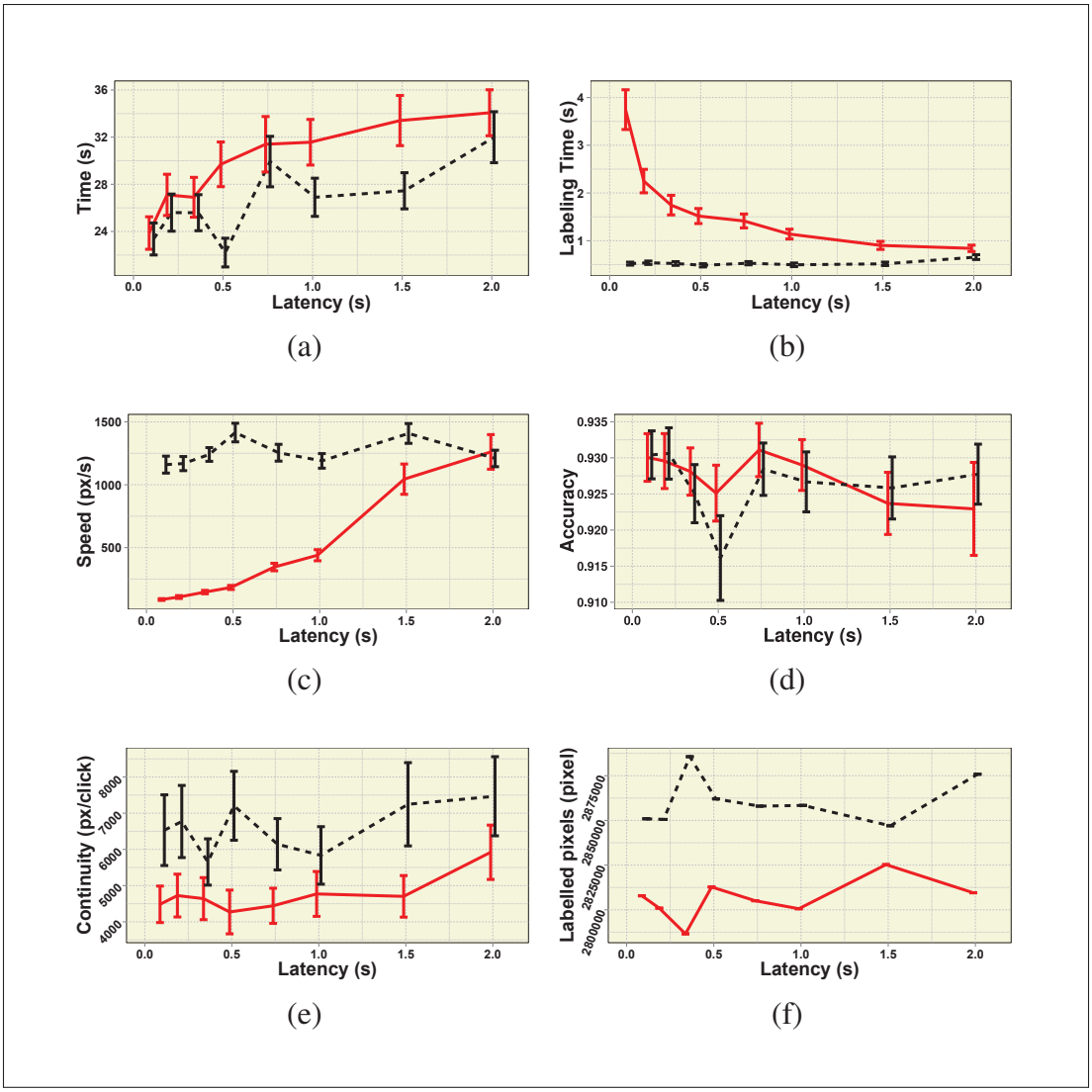

Figure 4.4 Summary of the results obtained with the automatic (solid red line) and user-initiated (dashed black) refresh methods: (a) average time to complete one image segmentation trial, (b) average time to label an image (c) average drawing speed per image, (d) average F_1-*Score* per segmented image (bigger is better), (e) average strokes continuity per image (bigger means longer strokes) and (f) sum of labelled pixels for all the images

shorter segmentation time obtained using the user-initiated refresh method at 500 ms latency. This score is similar to the one obtained with 100 ms latency for the same refresh condition. However, the corresponding accuracy score shows one of the worst performances with an average F_1-*Score* index equal to $\mathscr{A} = 0.916 \pm 0.005$ (see Figure 4.4.d). Unfortunately, we are

unable to explain the reason of this *outlier* value. This can be due to a lack of attention on the part of some users, which happened to be (for some reason) at 500 ms latency. Figure 4.5 shows the histogram of the experiments having a F_1-*Score* index below 0.9 for user-initiated refresh. Most of the segmentation errors occur at 500 ms latency, which may explain the low overall time obtained.

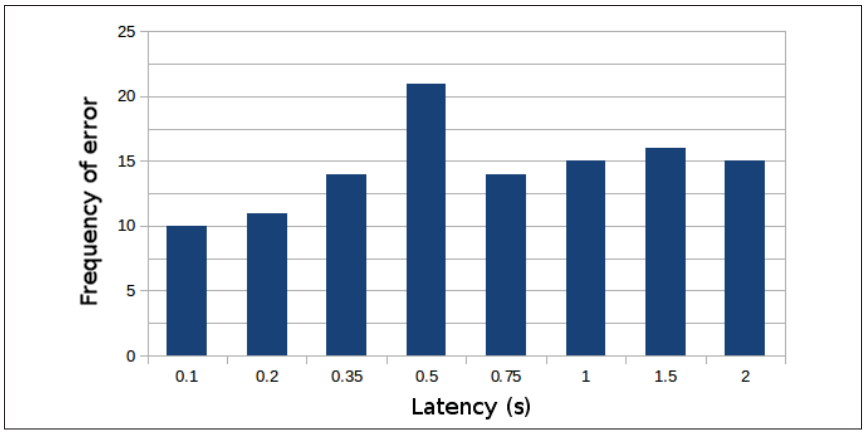

Figure 4.5 Frequency of error (F_1-*Score* < 0.9) obtained with the user-initiated refresh method

4.5.2 Labelling time and drawing speed

The average time required to label an image is shown in Figure 4.4.b. The user's drawing performances are clearly affected by latency when the automatic refresh method is used. For a latency of 100 ms, $t_\Lambda^{100} = 3.74\,\text{s} \pm 0.41\,\text{s}$, representing $16.19\% \pm 1.22\%$ of the overall segmentation time. Then, a rapid decrease in the labelling time occurs between 100 ms and 500 ms latency, before reaching a stable value around $t_\Lambda = 1.16\,\text{s} \pm 0.11\,\text{s}$ in the 500 ms to 2000 ms latency interval. Here, the average labelling time represents $3.89\% \pm 0.25\%$ of the overall segmentation time. Using the user-initiated refresh method, the latency condition seems to have little effect on the labelling time. The average performance for all the participants was $0.53\,\text{s} \pm 0.03\,\text{s}$, i.e., $2.18\% \pm 0.10\%$ of the overall segmentation time.

The average drawing speed is shown in Figure 4.4.c. The results are consistent with the labelling time. Using the automatic refresh method, the drawing speed increases with latency to reach its highest value of $\upsilon = 1261.15\,\text{pixel/s} \pm 137.73\,\text{pixel/s}$ at 2 s latency. Using the user-initiated refresh method, the drawing speed does not seem to be affected by latency, with an average speed of $\upsilon = 1255.80\,\text{pixel/s} \pm 101.57\,\text{pixel/s}$. This value is most likely the upper speed limit achievable while maintaining reasonably careful drawings in the tested scribble-based segmentation approach.

4.5.3 Segmentation accuracy

The average accuracy obtained per image is plotted in Figure 4.4.d. In this study, the participants were allowed to adjust the drawings until a satisfactory segmentation result was obtained, without a time limit. Having been shown the F_1-*Score* during the training step, the participants were visually habituated to match the segmentation result and the ground truth with sufficient similarity (F_1-*Score* > 0.90). Therefore, it is not surprising to observe high F_1-*Score* ($\mathscr{A} = 0.927 \pm 0.036$ for automatic refresh and $\mathscr{A} = 0.926 \pm 0.037$ for the user-initiated refresh methods) with small variations between participants. To gain more insight into the accuracy performance, we recorded the evolution of the F_1-*Score* index during the segmentation task for all latency conditions. Figure 4.6 shows the cumulative fraction of all the trials that reached a F_1-*Score* of 0.90 or above over time. Using the user-initiated refresh method, users rapidly achieved satisfactory segmentation results. In contrast, using the automatic refresh method, satisfactory results took longer to achieve under long latencies.

4.5.4 Continuity of the strokes

The stroke continuity results are shown in Figure 4.4.e. Recall that the continuity measure indicates the average length of the strokes per click and gives insight into how the labels were drawn. Using the user-initiated refresh method, the strokes produced by the participants appear to be longer, i.e., with larger values of ζ. However, an analysis of variance (ANOVA) (Chambers & Hastie, 1992) reveals no statistically significant effects of the *latency* on the con-

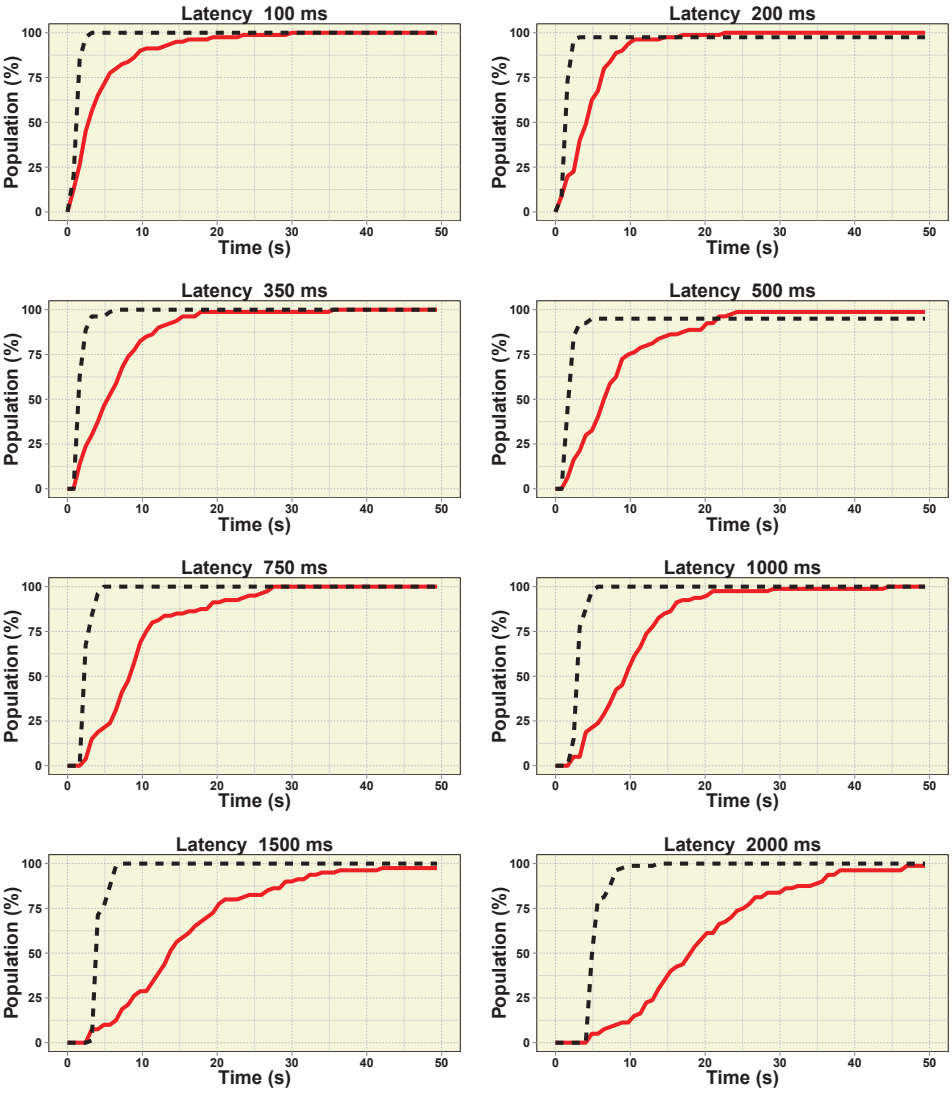

Figure 4.6 The cumulative fraction of trials having a F_1-*Score* of 0.9 or above as a function of time, using the automatic refresh (solid red line) and user-initiated refresh (dashed black) methods

tinuity of the labels, with $p = 0.074$ for the automatic refresh and $p = 0.29$ for the user-initiated refresh methods.

4.5.5 Number of labels

The total number of labels required to segment all the images for each latency condition is shown in Figure 4.4.f. The difference in number of labelled pixels is not statistically significant ($p = 0.710$). However, because each participant segmented the exact same dataset using the same latency condition, we can directly compare the performance of the participants with respect to the refresh methods. Assuming that a given participant would behave similarly given twice the same image, this compensates for the inter-user variability. The results show that using the automatic refresh method, fewer labels were required to complete the segmentation task than when using the user-initiated refresh method. This is mostly due to the fact that users stop labelling as soon as a satisfactory segmentation result appears on the screen.

4.6 Discussion

Table 4.1 shows a qualitative summary of the experimental results. In the first part of this section, we discuss the results from the user performance perspective. In the second part, our analysis focuses on the segmentation performance.

4.6.1 User performance

The latency condition seems to have more influence on the user's performance under the automatic refresh condition. However, the effects differ depending on the magnitude of the latency. In our discussion we analyze the results in terms of two distinct latency ranges: a Rapid Update Rate (RUR) between $[100\,\text{ms}, 500\,\text{ms}]$ latency and a Slow Update Rate (SUR) between $[750\,\text{ms}, 2000\,\text{ms}]$ latency.

4.6.1.1 Automatic vs. user-initiated refresh method

Automatically refreshing the results makes the user subject to a within-activity latency condition. This is because the cognitive block is combined with the interactive block in a single action: *drawing labels*. Any latency occurring during this action would be experienced as

Table 4.1 Qualitative summary of the experimental results

Refresh method		Latencies (ms): Rapid Update Rate (RUR) 100 to 500 ms	Latencies (ms): Slow Update Rate (SUR) 750 to 2000 ms
	Automatic	t_Ω: fast (low)	t_Ω: slow (high)
		t_Λ: slow (high)	t_Λ: fast (low)
		υ: slow	υ: fast
		ζ: discontinuous	ζ: discontinuous
		$\mathscr{N}$: fewer labels	$\mathscr{N}$: fewer labels
	User-initiated	t_Ω: fast (low)	t_Ω: slow (high)
		t_Λ: fast (low)	t_Λ: fast (low)
		υ: fast	υ: fast
		ζ: continuous	ζ: continuous
		$\mathscr{N}$: more labels	$\mathscr{N}$: more labels

within-activity latency by the user. On the other hand, a user-initiated refresh of the results allows the user to explicitly separate the interactive block and the cognitive block using two different actions. The user is then subject to a between-activity latency condition. In our experimental results, the user's performance under the user-initiated refresh condition was less sensitive to latency than under the automatic refresh condition. In fact, under the user-initiated refresh condition, participants behaved similarly in terms of drawing (i.e., labelling time, drawing speed, number of labels and stroke continuity), regardless of the latency. Moreover, when the latency was large, participants displayed similar behaviour regardless of the refresh method used.

4.6.1.2 Relationship between latency and drawing efficiency

The user's performance varies differently depending on the latency. When using the automatic refresh method, the user's performance is highly sensitive to latency. In the RUR range, segmentation updates are sufficiently fast, making it possible to instantly adapt to the visual feedback while drawing. The cognitive process involved during the segmentation, hereafter referred to as the *thinking process*, i.e., evaluating the current segmentation result and decid-

ing where to draw the next labels, slows down the labelling process. This is reflected in the longer time elapsed while drawing labels and the slower drawing speed under short latency conditions. The labelling time decreases and the drawing speed increases as the user is interrupted less frequently by the updates, in the SUR range. This is reflected in the long latency condition results. The attention allocated to the updates decreases if the response is too slow. The results obtained with the user-initiated refresh method suggest that between-activity latency requires less attention than within-activity latency. In fact, allowing the user to control the refresh reduces the risk of interrupting the thinking process. Therefore, the drawing can be performed more *efficiently*. We also observe that for 2000 ms latency, the drawing performance measures obtained using the automatic refresh method are similar to those obtained using the user-initiated refresh method. This suggests that, close to $\sim$ 2000 ms latency, the user attention devoted to segmentation updates during the drawing becomes insignificant. This is in accordance with the recommended 2 s threshold for interactive applications discussed in (Miller, 1968).

4.6.1.3 Participant feedback

After they completed each experiment, we collected individual participants' feedback during an informal discussion. Using the automatic refresh method, all participants agreed that performing under long latency conditions was a bit frustrating, while it was more convenient under short latency conditions. Participants were more aware of the latency when using the automatic refresh method. This is probably because they had no control over the refresh rate and the segmentation was part of the labelling process. In contrast, most participants did not notice the latency variations when they completed the segmentation using the user-initiated refresh method.

4.6.2 Segmentation performance

4.6.2.1 Relationship between latency and segmentation time

The overall segmentation time is commonly used to characterize segmentation efficiency. Despite the extra effort required to manually refresh the segmentation, the user-initiated method yielded better overall performances. However, under latencies between $[100\,\text{ms}, 350\,\text{ms}]$ the segmentation task was completed in similar amounts of time using both refresh methods. In this latency range, the user's drawing performance is optimal for the user-initiated refresh method, which is not the case for the automatic refresh method. This suggests that under $\sim 350\,\text{ms}$ of computation time the type of latency (i.e., between- or within-activity latency) is irrelevant regarding the overall time. This is consistent with the human physical reaction, which has been reported to be located between $[100\,\text{ms}, 200\,\text{ms}]$ (MacKenzie, 1992). We believe that the extra delay could be due to the mouse manipulation.

4.6.2.2 Segmentation accuracy

The study revealed that given sufficient time, the latency does not affect segmentation accuracy, regardless of the refresh method. All participants were able to achieve satisfactory segmentation results. However, results were achieved faster with the user-initiated refresh.

4.7 Conclusions

Our study investigated the user's performance during an interactive scribble-based segmentation task under different latency conditions. The computations were performed with latencies of 100 ms, 200 ms, 350 ms, 500 ms, 750 ms, 1000 ms, 1500 ms and 2000 ms. Two refresh conditions were tested. First, we tested the automatic refresh method, wherein the results are updated automatically as soon as available. Then, we tested the user-initiated refresh method, wherein the user explicitly asks for the updates by pressing a key. For each latency condition, we measured the user's performance in terms of the overall time needed to complete the seg-

mentation task, the time required to draw the labels, the speed of the drawings, the accuracy of the segmentation results, the continuity of the scribbles and the number of labels drawn.

Obviously, increasing the latency has negative impacts on the overall performance. However, this affects the user's performance differently depending on the latency and the refresh method. For the automatic refresh method, we observed two modes in the user's behaviour. The first mode occurs for latencies between 100 ms and 500 ms. In this short latency mode, overall segmentation time and drawing time are highly sensitive to changes in latency. The user is attentive to the segmentation updates. The time required to complete the drawing decreases rapidly to reach its minimum around 500 ms latency. The second mode occurs for latencies between 750 ms and 2000 ms. In this long latency mode, the overall segmentation time increases slowly with the latency. The user's behaviour is less sensitive to the latency changes. The time required to draw the labels is stable across latencies. The user is less attentive to the updates, as observed through the increase in drawing speed. The transition between the two modes occurs somewhere between 500 ms and 750 ms latency, depending on the user.

For the user-initiated refresh method, increased latency has less significant impacts on the user's behaviour. In this case, the drawing and the interpretation of the result updates are dissociated into two separate processes. The attention of the user is focused on a single task at a time, which improves his/her performance. In terms of the overall segmentation time and the time required to label the image, the user-initiated refresh method showed better performances. Given a sufficient amount of time, the accuracy of the chosen segmentation method is not affected by the latency nor the refresh method. However, satisfactory results were obtained earlier using the user-initiated refresh method. Finally, using the automatic refresh method, participants completed the segmentation task by labelling fewer pixels.

The automatic and user-initiated refresh methods tested in our study exemplify the within- and between-activity types of latency, respectively. Results obtained using both methods are consistent with the theoretical characterization of latency discussed by Miller (1968). We ob-

serve a convergence of the user behaviour as the latency approaches 2 s, which is the suggested threshold for an effective response time in interactive applications.

It is notable that high performance systems are becoming commonly available, with sufficient computational power to provide real-time segmentation response (Delong & Boykov, 2008; Grady *et al.*, 2005). However, the user interaction time represents an important part of the overall segmentation time. It would be relevant to investigate how to improve the user interaction mechanism during the segmentation task. For example, future work would involve to visually help identifying useful labels while drawing the scribbles. In fact, the label drawn during the segmentation task affects the segmentation results at different levels, resulting in some scenarios in unavailing actions that produce none or little change in the segmentation results. Therefore, we believe that focusing future research on how to improve the user performance could result in much more gain in segmentation effectiveness.

CONCLUSION AND RECOMMENDATIONS

The purpose of the presented research was to improve the effectiveness of both the computation and the user performances during an interactive image segmentation task. Specifically, we investigated the scribble-based interaction mechanism, in which the user draws foreground and background labels to perform the segmentation. Three objectives were achieved throughout this work. The first objective was to propose a segmentation approach that reduces the computation time without altering the segmentation quality. The concept was to reduce the image size by discarding pixels that are of low relevance to the segmentation process. These pixels were manually selected using an additional contour roughly drawn by the user. The second objective directly results from the findings of the first contribution, as we propose a framework to automatically extract relevant pixels on the image in order to further speed up the computations. In this case, no additional contour drawing was required. This led to a real-time segmentation response while preserving a high quality of the segmentation results. However, preliminary results showed that the user performance was sensitive to the visual feedback and the delay of the response provided by the segmentation method. The third objective was then to assess the user performance under different response time conditions. The conducted experiment helped to understand how the user behaves according to the segmentation visual feedback. From this study, we derived guidelines to design effective interaction for such image segmentation tasks.

Contributions

Contribution 1: We proposed a new interaction mechanism that uses a contour drawn by the user to dynamically reduce the graph size. The main benefits of this approach are twofold: 1) the approach is generalizable to all graph-based segmentation approaches that rely on a scribble-like interaction mechanisms; 2) the experimental study showed that only information near the object boundary is required to achieve a satisfactory segmentation results. The proposed approach significantly improves the computational performance of the segmentation, de-

spite the time required to draw the additional contour. Moreover, we showed how our approach can be combined with existing graph reduction approaches, such as super-pixel clustering, for further improvement of the computation time.

Contribution 2 : The *FastDRaW* method, a fast adaptation of the random walker segmentation algorithm (Grady, 2006), was proposed and used in the context of high resolution image segmentation. The method uses a multi-scale framework and a region of interest search space to reduce the computation time and focus the segmentation results. FastDRaW allows a real-time computation for standard medical image sizes of 512×512 pixels, and an interactive computation (below 2 s) for high resolution images of size 1500×1500 pixels.

Contribution 3: We investigated the effects of the visual feedback latency and timing on the user's performance in scribble-based segmentation methods. We compared two techniques for refreshing visual feedback under different levels of latencies from 100 ms to 2 s: (i) automatic refresh, and (ii) user-initiated refresh. The user-initiated refresh yielded better overall segmentation performance than automatic refresh, despite the extra effort required to activate the refresh. For short latencies, the user's attention is focused on the automatic visual feedback, slowing down his/her labelling performance. This effect is attenuated as the latency grows larger, and the two refresh techniques yield similar user performance at the largest latencies.

Recommendations

In light of our experiments, it becomes clear that the user has a significant influence on the outcomes of the interactive image segmentation process. In this section, we first give some general guidelines that one might consider while designing an interactive image segmentation method. Then, we discuss potential directions that might be worthy of future investigation.

From our own experience designing and testing interactive segmentation algorithms, we provide the following recommendations:

- Before starting the design of a segmentation approach, it is important to consider the context and the purpose of the application. A general purpose segmentation approach is suitable because it can solve multiple segmentation problems. But often, it provides less accurate results than a dedicated algorithm.

- Simple mechanisms are recommended. Scribble-based segmentation approaches do not require a high drawing accuracy. It is possible to achieve satisfactory results with a rough drawing.

- If the segmentation response is fast enough (below $\sim 350\,\text{ms}$), consider an automatic refresh. Otherwise, provide the user the ability to refresh the results manually.

- Separate the tasks involved in the segmentation method into different processing threads, e.g., drawing labels on the image, calculating the object boundary, displaying the results, etc.

The user experiments conducted in this thesis were performed on 2D medical images. However, the computational processing time increases drastically when dealing with 3D images. More importantly, the user interaction requires more complex mechanisms. Most interaction mechanisms available in 3D medical image segmentation software (e.g., Slicer 4[1], MITK[2], ITK Snap[3]) propose the use of three orthogonal planes to navigate through the 3D image. Hence, labels are provided by drawing on 2D slices. This is not a trivial task. It often requires medical (or anatomical) background knowledge from the user to be able to understand where to look at. In future work, it might be interesting to investigate the effect of the 3D navigation on the user performance, particularly in label drawing tasks. This involves zooming in and out, panning the view window across the image and scrolling through a set of images, which require signif-

[1] https://www.slicer.org/

[2] http://www.mitk.net

[3] http://www.itksnap.org

icant efforts. In other words, given a 3D segmentation process completed in a certain amount of time, it is interesting to understand in which task the user spent most of his/her time, e.g., labelling, zooming/panning or navigating through slices.

Other publications indirectly related to this work

During my Ph. D., I worked on related topics in medical image processing, which are not directly connected with the content of this thesis. This work resulted in two articles published in peer reviewed conference proceedings:

- Houssem-Eddine Gueziri, Sébastien Tremblay, Catherine Laporte and Rupert Brooks, *Graph-based 3D-Ultrasound reconstruction of the liver in the presence of respiratory motion*, LNCS Vol. 10129, pp.48-57, Reconstruction, Segmentation, and Analysis of Medical Images: MICCAI 2016 Workshops, Athens, Greece.
- Houssem-Eddine Gueziri, Michael J. McGuffin and Catherine Laporte, *Visualizing Positional Uncertainty in Freehand 3D Ultrasound*, Proc. SPIE 90361H, Medical Imaging: Image-Guided Procedures, Robotic Interventions, and Modeling, San Diego, USA. (2014)

APPENDIX I

COMPUTATIONAL COMPLEXITY

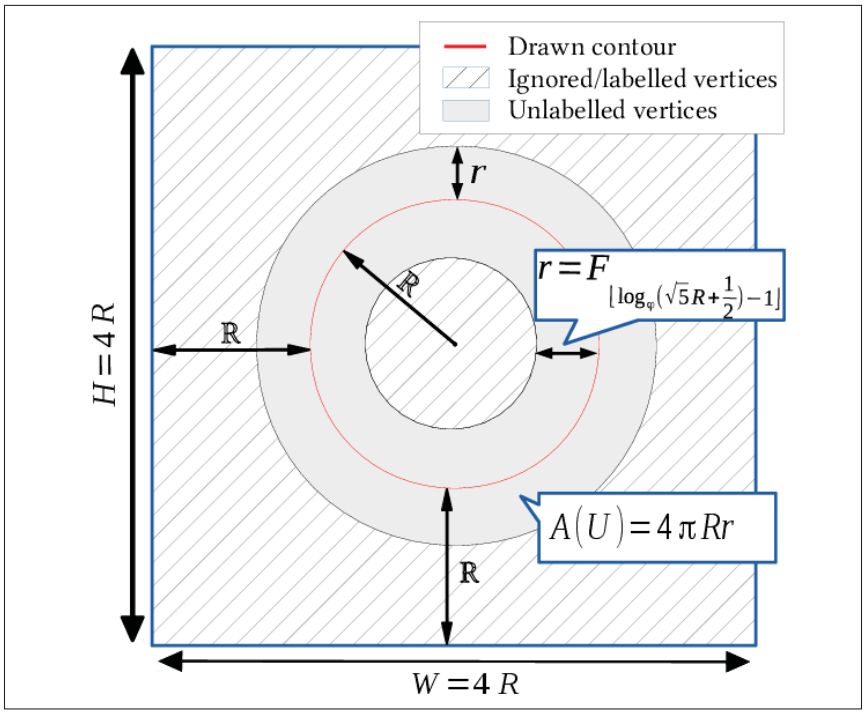

Figure-A I-1 Representation of the unlabeled pixels given a circle drawing with a radius of R

Suppose that Eq. 2.7 is used to generate the layers and Eq. 2.8 to select the *DSL*. We consider the segmentation of a circular object in the middle of the image, and assume that the user draws a circle with radius R in the center of an image of size $(4R)^2$. This is the worst case circular contour for this image size, as any other size would increase the number of labeled vertices. The complexity of RW segmentation is $O(|U|)$, and $|U|$, the number of the unlabeled pixels, is given by

$$\mathscr{A}(U) = 4\pi Rr, \tag{A I-1}$$

where

$$r = t\left(\left\lfloor \log_\phi(\sqrt{5}(R-1)+\frac{1}{2})\right\rfloor\right) \tag{A I-2}$$

is the minimum distance of the layer generated from the circular drawing (see Figure I-1). Using Binet's formula,

$$\mathscr{A}(U) = 4\pi R \frac{\phi^{\left\lfloor \log_\phi(\sqrt{5}(R-1)+\frac{1}{2})\right\rfloor} - (-\phi)^{-\left\lfloor \log_\phi(\sqrt{5}(R-1)+\frac{1}{2})\right\rfloor}}{\sqrt{5}}. \tag{A I-3}$$

Noting that $(-1)^{-\left\lfloor \log_\phi(\sqrt{5}(R-1)+\frac{1}{2})\right\rfloor} = \pm 1$,

$$\mathscr{A}(U) \simeq 4\pi R^2 + R(\frac{2\pi}{\sqrt{5}} - 4\pi) \pm \frac{4\pi R}{5(R-1)+\frac{\sqrt{5}}{2}}. \tag{A I-4}$$

Then, under our simplifying assumptions, the complexity of our approach is $O(R^2)$.

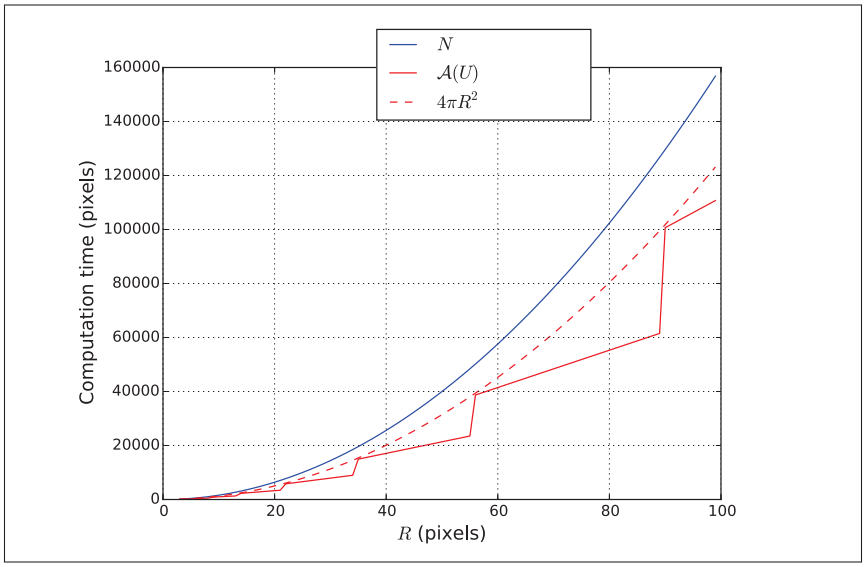

Figure-A I-2 Plot of N, $A(U)$, and $4\pi R^2$ as a function of the radius R in pixels. Note that $4\pi R^2 = O(A(U))$

This result assumes that a distance R separates the drawing from the image boundaries (see Figure I-1). Hence, by construction, we have $R = \frac{1}{4}\sqrt{N}$. The complexity is $O(4\pi R^2) = O(N)$, with a constant factor reduction of $\frac{\pi}{4} \simeq 0.785$. Figure I-2 shows N, $\mathscr{A}(U)$ and $4\pi R^2$ as a

function of R. As R grows towards N,

$$r = \min\left(\frac{H}{2} - R, \frac{W}{2} - R\right) < R.$$

Thus, the *DSL* is selected based on the outer region, R_{out}, and $O(Rr) < O(R^2)$. This represents the worst scenario, where the object size nearly equals the image size. This is rare in practice. The main advantage of our approach is that it reduces the order of complexity to the size of the area enclosed by the user drawing $R^2 \ll N$. Thus, for a fixed foreground object size and a growing image size, the complexity is constant.

APPENDIX II

RANDOM PATH GENERATION

To avoid introducing variability due to human factors, we use a simulation to compare our approach to the classical foreground-background seeding (FBS) approach, purely in terms of computation time. We simulate user interaction by generating random paths as illustrated in Figure II-1. These paths are used to (i) generate an approximation of the object boundary, for our approach and (ii) generate foreground and background seeds, for classical FBS. Random paths are generated from ground truth segmented images as follows.

The model : Let Pos_{seq} be the ordered sequence of pixel positions representing a subset of the object boundary (Figure II-1.a). Using Pos_{seq}, we generate the *nearest neighbourhood map* (NNM) of the image, i.e. considering each element of Pos_{seq} as a label, the NNM maps each pixel of the image to the nearest element of Pos_{seq} (Figure II-1.b). Pos_{seq} is also used to generate a sequence of *directional vectors*, (*DV*). Each element of *DV* represents a vector from $Pos_{seq}(i)$ to $Pos_{seq}(i+1)$ stored in polar coordinates with a r_i radius and φ_i angle (Figure II-1.c).

Random path generation: Starting from the first element of Pos_{seq}, we add Gaussian noise, i.e. $x = Pos_{seq}(0) + \varepsilon$ and $\varepsilon \sim \mathcal{N}(0,\sigma)$. A large σ will initialize the first point far from the object boundary. Then, using $i = NNM(x)$, the next position, x', is generated according to $DV(i)$. To capture the randomness of a freehand drawing, we add noise to the distance and angle from x to x', i.e. $\tilde{r} = r + \varepsilon_r$ and $\tilde{\varphi} = \varphi + \varepsilon_\varphi$ where $\varepsilon_\varphi \sim \mathcal{N}(0,\sigma_\varphi)$ and $\varepsilon_r \sim \mathcal{N}(0,\sigma_r)$. Thus, we can control the variation of the path with σ_r and σ_φ (Figure II-1.c). A large σ_r tends to smooth the path. In contrast a large σ_φ will lead to more jittery paths.

For FBS interaction, there is more freedom for seed generation. In this case, the image is manually marked with background and foreground areas where seeds can be generated. In our simulation we choose 4 background areas and 1 foreground area reasonably placed and

sufficiently wide to represent where the user might mark seeds. Then, we generate random paths using the same strategy described above.

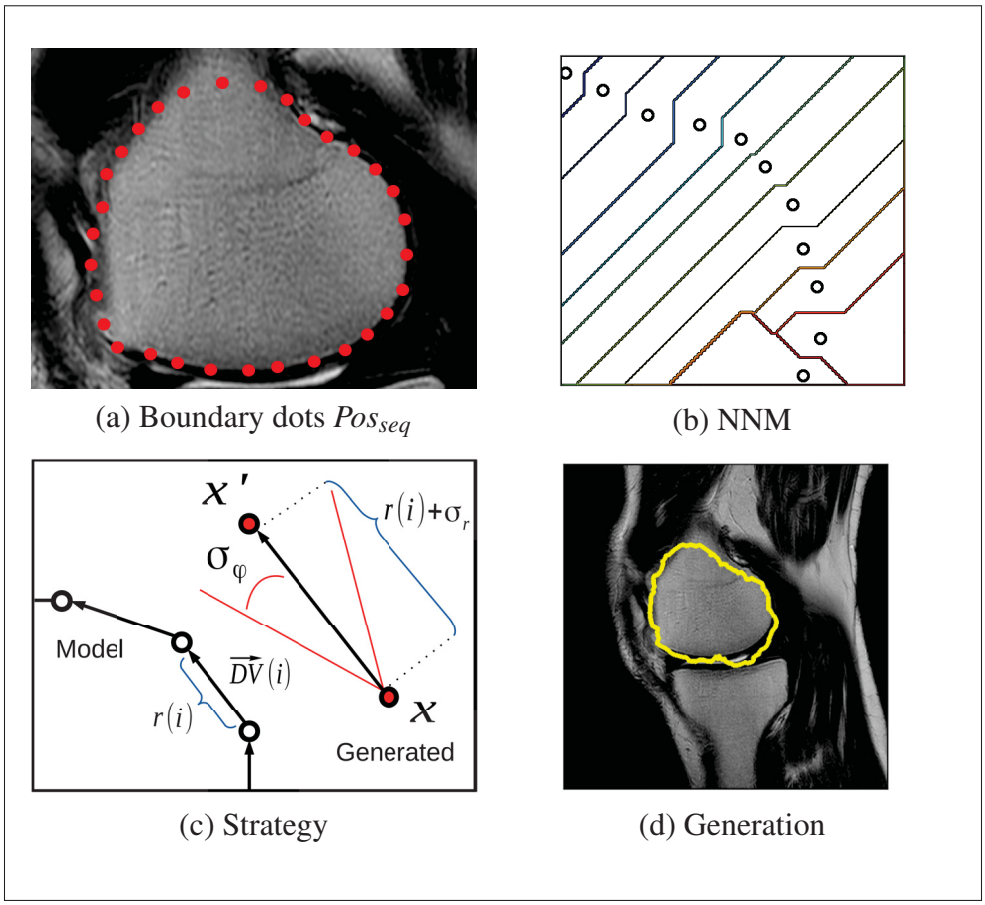

Figure-A II-1 Automatic marker generation strategy: (a) the sub-sampled ground truth segmentation, (b) the nearest neighbourhood map, (c) one step of the path generation strategy, and (d) example of the automatic marker generation

APPENDIX III

IMAGES USED FOR THE USER EXPERIMENT

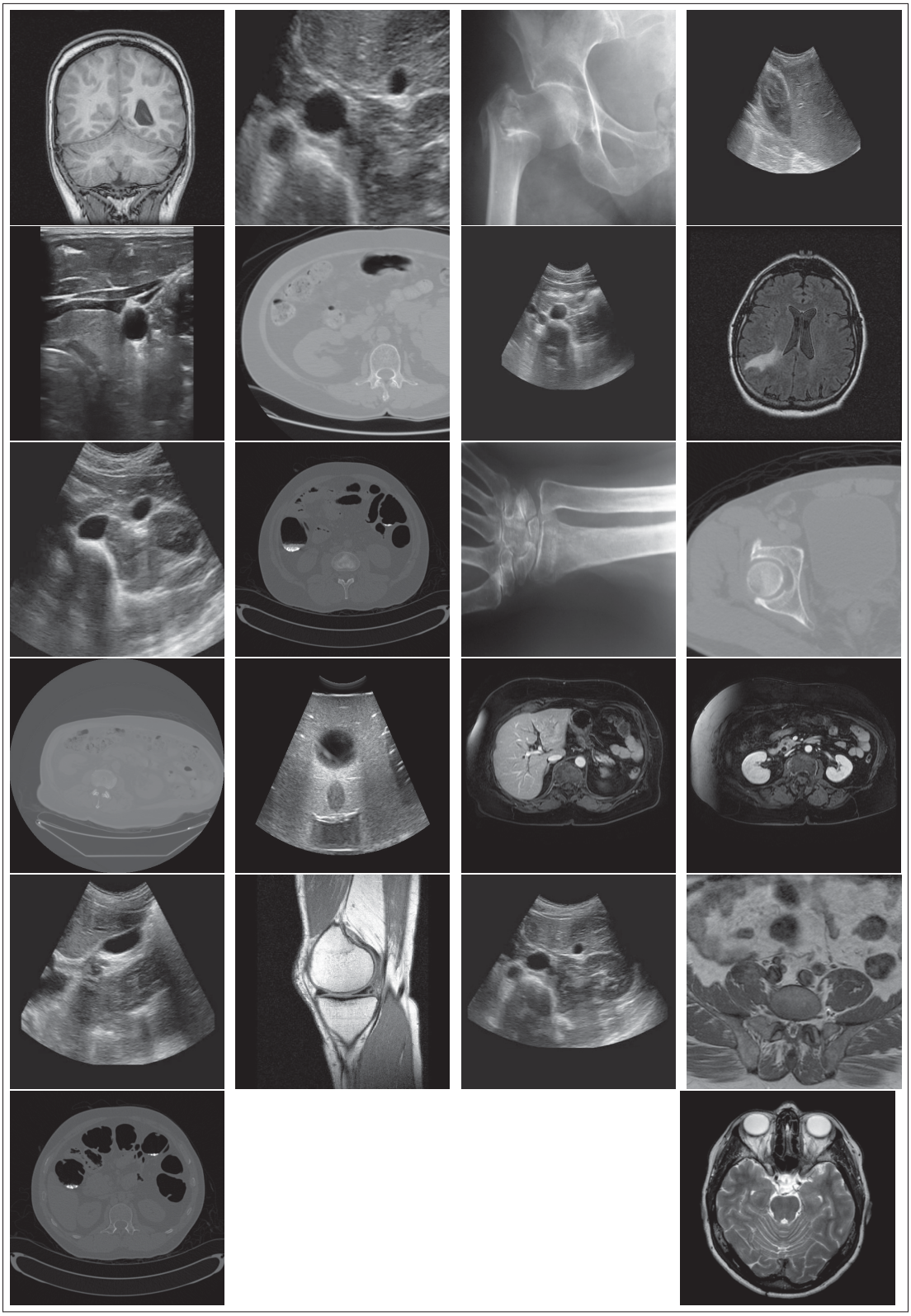

Figure-A III-1 Dataset of images used for user experiment

BIBLIOGRAPHY

Accot, J. & Zhai, S. (1999). Performance evaluation of input devices in trajectory-based tasks: An application of the steering law. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, (CHI '99), 466–472.

Achanta, R., Shaji, A., Smith, K., Lucchi, A., Fua, P. & Süsstrunk, S. (2012). SLIC superpixels compared to state-of-the-art superpixel methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(11), 2274–2282.

Adams, R. & Bischof, L. (1994). Seeded region growing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(6), 641–647.

Ahuja, R. K., Magnanti, T. L. & Orlin, J. B. (1993). *Network Flows: Theory, Algorithms, and Applications* (ed. 1). Prentice Hall.

AnatQuest. [http://anatquest.nlm.nih.gov/, Accessed: May 2015]. (2004). Anatomic images online [Online Dataset].

Andrews, S., Hamarneh, G. & Saad, A. (2010). Fast random walker with priors using pre-computation for interactive medical image segmentation. In *International Conference on Medical Image Computing and Computer Assisted Intervention* (vol. 6363, pp. 9-16).

Beigbeder, T., Coughlan, R., Lusher, C., Plunkett, J., Agu, E. & Claypool, M. (2004). The effects of loss and latency on user performance in unreal tournament 2003®. *Proceedings of 3rd ACM SIGCOMM Workshop on Network and System Support for Games*, (NetGames '04), 144–151.

Boykov, Y. & Jolly, M.-P. (2001). Interactive graph cuts for optimal boundary amp; region segmentation of objects in n-d images. *IEEE International Conference on Computer Vision*, 1, 105-112.

Boykov, Y. & Kolmogorov, V. (2004). An experimental comparison of min-cut/max- flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(9), 1124-1137.

Brunner, E., Domhof, S. & Langer, F. (2002). *Nonparametric analysis of longitudinal data in factorial experiments*. New York, NY, USA: Wiley series in probability and statistics.

Card, S. K., Robertson, G. G. & Mackinlay, J. D. (1991). The information visualizer, an information workspace. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, (CHI '91), 181–186.

Caselles, V., Kimmel, R. & Sapiro, G. (1997). Geodesic active contours. *International Journal of Computer Vision*, 22(1), 61–79.

Cevidanes, L., Bailey, L., GR Tucker, J., Styner, M., Mol, A., Phillips, C., Proffit, W. & Turvey, T. (2005). Superimposition of 3d cone-beam ct models of orthognathic surgery patients. *Dentomaxillofacial Radiology*, 34(6), 369-375.

Chambers, J. M. & Hastie, T. J. (1992). *Statistical models in S*. Wadsworth & Brooks/Cole Advanced Books & Software.

Chan, T. F. & Vese, L. A. (2001). Active contours without edges. *IEEE Transactions on Image Processing*, 10(2), 266-277.

Chen, L., Huang, X. & Tian, J. (2015). Retinal image registration using topological vascular tree segmentation and bifurcation structures. *Biomedical Signal Processing and Control*, 16, 22 - 31.

Clark, K., Vendt, B., Smith, K., Freymann, J., Kirby, J., Koppel, P., Moore, S., Phillips, S., Maffitt, D., Pringle, M., Tarbox, L. & Prior, F. (2013). The cancer imaging archive (TCIA): maintaining and operating a public information repository. *Journal of digital imaging*, 26(6), 1045–1057.

Comaniciu, D., Meer, P. & Foran, D. J. (1999). Image-guided decision support system for pathology. *Machine Vision and Applications*, 11(4), 213–224.

Conover, W. J., Johnson, M. E. & Johnson, M. M. (1981). A Comparative Study of Tests for Homogeneity of Variances, with Applications to the Outer Continental Shelf Bidding Data. *Technometrics*, 23(4), 351–361.

Cootes, T. F., Edwards, G. J. & Taylor, C. J. (2001). Active appearance models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(6), 681-685.

Cootes, T., Taylor, C., Cooper, D. & Graham, J. (1995). Active shape models-their training and application. *Computer Vision and Image Understanding*, 61(1), 38 - 59.

Couprie, C., Grady, L., Najman, L. & Talbot, H. (2009, Sept). Power watersheds: A new image segmentation framework extending graph cuts, random walker and optimal spanning forest. *IEEE International Conference on Computer Vision*, pp. 731-738.

Delong, A. & Boykov, Y. (2008). A scalable graph-cut algorithm for n-d grids. *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1-8.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1), 269–271.

Duda, R. O., Hart, P. E. & Stork, D. G. (2000). *Pattern classification (second edition)*. John Wiley & Sons.

Faisal, A., Ng, S. C., Goh, S. L., George, J., Supriyanto, E. & Lai, K. W. (2015). Multiple lrek active contours for knee meniscus ultrasound image segmentation. *IEEE Transactions on Medical Imaging*, 34(10), 2162-2171.

Falcão, A. X., Udupa, J. K. & Miyazawa, F. K. (2000). An ultra-fast user-steered image segmentation paradigm: live wire on the fly. *IEEE Transactions on Medical Imaging*, 19(1), 55-62.

Falcao, A. X., Stolfi, J. & de Alencar-Lotufo, R. (2004). The image foresting transform: theory, algorithms, and applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(1), 19-29.

Falcão, A. X., Udupa, J. K., Samarasekera, S., Sharma, S., Hirsch, B. E. & de A. Lotufo, R. (1998). User-steered image segmentation paradigms: Live wire and live lane. *Graphical Models and Image Processing*, 60(4), 233 - 260.

Fitts, P. M. (1954). *The information capacity of the human motor system in controlling the amplitude of movement*. Journal of Experimental Psychology. American Psychological Association.

Forbus, K. D. & Usher, J. (2002). Sketching for knowledge capture: A progress report. *Proceedings of the 7th International Conference on Intelligent User Interfaces*, (IUI '02), 71–77.

Ford Jr, L. R. & Fulkerson, D. R. (1962). *Flows in networks*. Princeton university press.

Galasso, F., Keuper, M., Brox, T. & Schiele, B. (2014). Spectral graph reduction for efficient image and streaming video segmentation. *IEEE Conference on Computer Vision and Pattern Recognition*.

Gocławski, J., Węgliński, T. & Fabijańska, A. (2015). Accelerating the 3D random walker image segmentation algorithm by image graph reduction and gpu computing. In *Image Processing & Communications Challenges 6* (vol. 313, pp. 45-52).

Goldberg, A. V. & Tarjan, R. E. (1988). A new approach to the maximum-flow problem. *J. acm*, 35(4), 921–940.

Grady, L. (2006). Random walks for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(11), 1768-1783.

Grady, L. & Sinop, A. (2008). Fast approximate random walker segmentation using eigenvector precomputation. *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1-8.

Grady, L., Schiwietz, T., Aharon, S. & Westermann, R. (2005). Random walks for interactive organ segmentation in two and three dimensions: Implementation and validation. In *International Conference on Medical Image Computing and Computer Assisted Intervention* (pp. 773-780).

Greig, D. M., Porteous, B. T. & Seheult, A. H. (1989). Exact maximum a posteriori estimation for binary images. *Journal of the Royal Statistical Society. Series B (Methodological)*, 51(2), 271-279.

Gulshan, V., Rother, C., Criminisi, A., Blake, A. & Zisserman, A. (2010, June). Geodesic star convexity for interactive image segmentation. *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3129-3136.

Harders, M. & Szekely, G. (2003). Enhancing human-computer interaction in medical segmentation. *Proceedings of the IEEE*, 91(9), 1430-1442.

Hebbalaguppe, R., McGuinness, K., Kuklyte, J., Healy, G., O'Connor, N. & Smeaton, A. (2013, Jan). How interaction methods affect image segmentation: User experience in the task. *1st IEEE Workshop on User-Centered Computer Vision*, pp. 19–24.

Honnorat, N., Eavani, H., Satterthwaite, T., Gur, R., Gur, R. & Davatzikos, C. (2015). Grasp: Geodesic graph-based segmentation with shape priors for the functional parcellation of the cortex. *NeuroImage*, 106, 207 - 221.

Jota, R., Ng, A., Dietz, P. & Wigdor, D. (2013). How fast is fast enough?: A study of the effects of latency in direct-touch pointing tasks. *Proceedings of the sigchi conference on human factors in computing systems*, (CHI '13), 2291–2300.

Kass, M., Witkin, A. & Terzopoulos, D. (1988). Snakes: Active contour models. *International Journal of Computer Vision*, 1(4), 321-331.

Kolmogorov, V. & Zabih, R. (2001). Computing visual correspondence with occlusions using graph cuts. *IEEE International Conference on Computer Vision*, 2, 508-515.

Krs, V., Yumer, E., Carr, N., Benes, B. & Měch, R. (2017). Skippy: Single view 3d curve interactive modeling. *ACM Transactions on Graphics*, 36(4), 1–12.

Landay, J. A. & Myers, B. A. (1995). Interactive sketching for the early stages of user interface design. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, (CHI '95), 43–50.

Lefohn, A. E., Cates, J. E. & Whitaker, R. T. (2003). Interactive, gpu-based level sets for 3d segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention* (pp. 564–572).

Lermé, N., Malgouyres, F. & Letocart, L. (2010, Sept). Reducing graphs in graph cut segmentation. *IEEE International Conference on Image Processing*, pp. 3045-3048.

Levinshtein, A., Stere, A., Kutulakos, K., Fleet, D., Dickinson, S. & Siddiqi, K. (2009). TurboPixels: Fast Superpixels Using Geometric Flows. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(12), 2290-2297.

Li, Y., Sun, J., Tang, C.-K. & Shum, H.-Y. (2004). Lazy snapping. *ACM Transactions on Graphics*, 23(3), 303–308.

Litjens, G., Toth, R., van de Ven, W., Hoeks, C., Kerkstra, S., van Ginneken, B., Vincent, G., Guillard, G., Birbeck, N., Zhang, J., Strand, R., Malmberg, F., Ou, Y., Davatzikos, C., Kirschner, M., Jung, F., Yuan, J., Qiu, W., Gao, Q., Edwards, P. E., Maan, B., van der Heijden, F., Ghose, S., Mitra, J., Dowling, J., Barratt, D., Huisman, H. & Madabhushi, A. (2014). Evaluation of prostate segmentation algorithms for mri: The promise12 challenge. *Medical Image Analysis*, 18(2), 359 - 373.

Liu, Z. & Heer, J. (2014). The effects of interactive latency on exploratory visual analysis. *IEEE Transactions on Visualization and Computer Graphics*, 20(12), 2122-2131.

Long, J., Shelhamer, E. & Darrell, T. (2015). Fully convolutional networks for semantic segmentation. *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3431–3440.

Luccheseyz, L. & Mitray, S. (2001). Color image segmentation: A state-of-the-art survey. *Proceedings of the Indian National Science Academy (INSA-A)*, 67(2), 207–221.

MacKenzie, I. S. (1992). Fitts' law as a research and design tool in human-computer interaction. *Human-Computer Interaction*, 7, 91–139.

MacKenzie, I. S. (2013). *Human-computer interaction, an empirical research perspective*. Burlington, MA, USA: Elsevier Morgan Kaufmann.

Maksimovic, R., Stankovic, S. & Milovanovic, D. (2000). Computed tomography image analyzer: 3d reconstruction and segmentation applying active contour models — 'snakes'. *International Journal of Medical Informatics*, 58–59, 29 - 37.

Maurer,Jr., C. R., Rensheng, Q. & Raghavan, V. (2003). A linear time algorithm for computing exact euclidean distance transforms of binary images in arbitrary dimensions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(2), 265–270.

McGuinness, K. & O'Connor, N. E. (2010). A comparative evaluation of interactive segmentation algorithms. *Pattern Recognition*, 43, 434–444.

McInerney, T. & Terzopoulos, D. (1996). Deformable models in medical image analysis: a survey. *Medical image analysis*, 1(2), 91–108.

Merhav, N. & Bhaskaran, V. (1997). Fast algorithms for dct-domain image downsampling and for inverse motion compensation. *IEEE Transactions on Circuits and Systems for Video Technology*, 7(3), 468-476.

Mi, H., Petitjean, C., Vera, P. & Ruan, S. (2015). Joint tumor growth prediction and tumor segmentation on therapeutic follow-up {PET} images. *Medical Image Analysis*, 23(1), 84 - 91.

Miller, R. B. (1968). Response time in man-computer conversational transactions. *Proceedings of the December 9-11, 1968, fall joint computer conference, part I*, pp. 267–277.

Miranda, P. A. V., Falcao, A. X. & Spina, T. V. (2012). Riverbed: A novel user-steered image segmentation method based on optimum boundary tracking. *IEEE Transactions on Image Processing*, 21(6), 3042-3052.

Mishra, A., Wong, A., Zhang, W., Clausi, D. & Fieguth, P. (2008, Aug). Improved interactive medical image segmentation using enhanced intelligent scissors (EIS). *IEEE International Conference on Engineering in Medicine and Biology Society*, pp. 3083-3086.

Moltz, J. H., Braunewell, S., Rühaak, J., Heckel, F., Barbieri, S., Tautz, L., Hahn, H. K. & Peitgen, H. O. (2011). Analysis of variability in manual liver tumor delineation in ct scans. *IEEE International Symposium on Biomedical Imaging: From Nano to Macro*, pp. 1974-1977.

Mori, G. (2005). Guiding model search using segmentation. *IEEE International Conference on Computer Vision*, 2, 1417–1423.

Mortensen, E. N. & Barrett, W. A. (1998). Interactive segmentation with intelligent scissors. *Graphical models and image processing*, 60(5), 349–384.

Munzel, U. & Hothorn, L. A. (2001). A unified approach to simultaneous rank test procedures in the unbalanced one-way layout. *Biometrical Journal*, 43(5), 553–569.

Newell, A. (1994). *Unified theories of cognition*. Harvard University Press.

Noble, J. A. & Boukerroui, D. (2006). Ultrasound image segmentation: a survey. *IEEE Transactions on medical imaging*, 25(8), 987–1010.

Olabarriaga, S. & Smeulders, A. (2001). Interaction in the segmentation of medical images: A survey. *Medical image analysis*, 5(2), 127–142.

Orlin, J. B. (2013). Max flows in o(nm) time, or better. *Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing*, (STOC '13), 765–774.

Peng, B., Zhang, L. & Zhang, D. (2013). A survey of graph theoretical approaches to image segmentation. *Pattern Recognition*, 46(3), 1020 - 1038.

Piqueras, S., Krafft, C., Beleites, C., Egodage, K., von Eggeling, F., Guntinas-Lichius, O., Popp, J., Tauler, R. & de Juan, A. (2015). Combining multiset resolution and segmentation for hyperspectral image analysis of biological tissues. *Analytica Chimica Acta*, 881, 24 - 36.

Potter, M. C., Wyble, B., Hagmann, C. E. & McCourt, E. S. (2014). Detecting meaning in rsvp at 13 ms per picture. *Attention, Perception, & Psychophysics*, 76(2), 270–279.

Protiere, A. & Sapiro, G. (2007). Interactive image segmentation via adaptive weighted distances. *IEEE Transactions on Image Processing*, 16(4), 1046–1057.

Ramkumar, A., Dolz, J., Kirisli, H. A., Adebahr, S., Schimek-Jasch, T., Nestle, U., Massoptier, L., Varga, E., Stappers, P. J., Niessen, W. J. & Song, Y. (2016). User interaction in semi-automatic segmentation of organs at risk: a case study in radiotherapy. *Journal of digital imaging*, 29(2), 264–277.

Rother, C., Kolmogorov, V. & Blake, A. (2004). Grabcut: Interactive foreground extraction using iterated graph cuts. *ACM Transactions on Graphics*, 23(3), 309–314.

Sadeghi, M., Tien, G., Hamarneh, G. & Atkins, M. S. (2009). Hands-free interactive image segmentation using eyegaze. *Proceedings of SPIE Medical Imaging: Computer-Aided Diagnosis*, 72601H.

Saraswathi, S. & Allirani, A. (2013). Survey on image segmentation via clustering. *2013 International Conference on Information Communication and Embedded Systems (ICICES)*, pp. 331-335.

Schneider, J., Kraus, M. & Westermann, R. (2009). GPU-based real-time discrete euclidean distance transforms with precise error bounds. *International Conference on Computer Vision Theory and Applications (VISAPP)*, pp. 435–442.

Shen, J., Du, Y., Wang, W. & Li, X. (2014). Lazy random walks for superpixel segmentation. *IEEE Transactions on Image Processing*, 23(4), 1451-1462.

Simhon, S. & Dudek, G. (2004). Sketch Interpretation and Refinement Using Statistical Models. *Eurographics Workshop on Rendering*.

Singaraju, D., Grady, L. & Vidal, R. (2009). P-brush: Continuous valued mrfs with normed pairwise distributions for image segmentation. *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1303-1310.

Spina, T. V., de Miranda, P. A. V. & Falcão, A. X. (2014). Hybrid approaches for interactive image segmentation using the live markers paradigm. *IEEE Transactions on Image Processing*, 23(12), 5756-5769.

Stakhov, A. P. (2009). *The mathematics of harmony: from euclid to contemporary mathematics and computer science*. Hackensack, NJ, USA: World Scientific.

Tang, T. W. & Chung, A. C. (2007). Non-rigid image registration using graph-cuts. *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pp. 916–924.

Top, A., Hamarneh, G. & Abugharbieh, R. (2011). Active learning for interactive 3d image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention* (pp. 603–610).

Trentacoste, M., Mantiuk, R. & Heidrich, W. (2011). Blur-aware image downsampling. *Computer Graphics Forum*, 30(2), 573–582.

Udupa, J. K., LeBlanc, V. R., Zhuge, Y., Imielinska, C., Schmidt, H., Currie, L. M., Hirsch, B. E. & Woodburn, J. (2006). A framework for evaluating image segmentation algorithms. *Computerized Medical Imaging and Graphics*, 30(2), 75 - 87.

Verborgh, R., Sande, M. V., Hartig, O., Herwegen, J. V., Vocht, L. D., Meester, B. D., Haesendonck, G. & Colpaert, P. (2016). Triple pattern fragments: A low-cost knowledge graph interface for the web. *Web Semantics: Science, Services and Agents on the World Wide Web*, 37, 184 - 206.

Vineet, V. & Narayanan, P. J. (2008). CUDA cuts: Fast graph cuts on the gpu. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pp. 1-8.

Wang, L., Shi, F., Li, G., Gao, Y., Lin, W., Gilmore, J. H. & Shen, D. (2014). Segmentation of neonatal brain {MR} images using patch-driven level sets. *NeuroImage*, 84, 141 - 158. doi: http://dx.doi.org/10.1016/j.neuroimage.2013.08.008.

Ware, C. & Balakrishnan, R. (1994). Target acquisition in fish tank VR: The effects of lag and frame rate. *Proceedings of Graphics Interface*, (GI '94), 1–7.

Ware, C., Arthur, K. & Booth, K. S. (1993). Fish tank virtual reality. *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems*, (CHI '93), 37–42.

West, R., Kajihara, M., Parola, M., Hays, K., Hillard, L., Carlew, A., Deutsch, J., Lane, B., Holloway, M., John, B., Sanandaji, A. & Grimm, C. (2016). Eliciting tacit expertise in 3d volume segmentation. *Proceedings of the 9th International Symposium on Visual Information Communication and Interaction*, (VINCI '16), 59–66.

Wyszecki, G. & Stiles, W. S. (1982). *Color science*. Wiley, New York, NY.

Xie, J., Hertzmann, A., Li, W. & Winnemöller, H. (2014). Portraitsketch: Face sketching assistance for novices. *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*, (UIST '14), 407–417.

Yang, W., Cai, J., Zheng, J. & Luo, J. (2010). User-friendly interactive image segmentation through unified combinatorial user inputs. *IEEE Transactions on Image Processing*, 19(9), 2470–2479.

Yu, L., Yang, X., Chen, H., Qin, J. & Heng, P.-A. (2017). Volumetric convnets with mixed residual connections for automated prostate segmentation from 3d mr images. *Association for the Advancement of Artificial Intelligence*, pp. 66–72.

Yu, S. X. & Shi, J. (2003). Multiclass spectral clustering. *Proceedings Ninth IEEE International Conference on Computer Vision*, pp. 313-319 vol.1.

Yuan, J., Bae, E., Tai, X.-C. & Boykov, Y. (2014). A spatially continuous max-flow and min-cut framework for binary labeling problems. *Numerische Mathematik*, 126(3), 559–587.

Zhang, H., Fritts, J. E. & Goldman, S. A. (2008). Image segmentation evaluation: A survey of unsupervised methods. *Computer Vision and Image Understanding*, 110(2), 260–280.

Zhang, L. & Wu, X. (2006). An edge-guided image interpolation algorithm via directional filtering and data fusion. *IEEE Transactions on Image Processing*, 15(8), 2226-2238.