Semi-Automatic Foreground Extraction For Natural Images

Andreas Rishede Hyllested
Martin Wallengren Nilsson

IT University Technical Report Series TR-2005-64

ISSN 1600-6100 March 2005

ISSN 1600-6100

ISBN 87-7949-094-8

Copies may be obtained by contacting:

IT University of Copenhagen
Rued Langgaards Vej 7
DK – 2300 Copenhagen S
Denmark

Telephone: +45 72 18 50 00
Telefax: +45 72 18 50 01
Web: www.itu.dk

IT University of Copenhagen

Semi-Automatic Foreground Extraction For Natural Images

Andreas Rishede Hyllested (arh) 190578-2971
Martin Wallengren Nilsson (mwn) 150976-2321

March 2005

Supervisor: Ole Fogh Olsen
IT University of Copenhagen

Abstract

The work described in this thesis was carried out at the IT University of Copenhagen, Denmark from September 2004 to March 2005. The focus of the work has been defined together with the collaboration partner Laerdal-Sophus A/S who seeks to replace a current manual segmentation method with a semi-automatic segmentation method.

The thesis produces a method for and describes the framework of interactive segmentation of foreground objects in natural images. The basis technology is segmentation by graph cut. To increase interactivity speed the graph construction is preceded by a toboggan watershed segmentation of the input image. The watershed segmentation is implemented in a multi-scale framework where different normalisation methods are tested. Based on the watershed segmented image is built a graph, in which every watershed segment corresponds to one node. The minimum-cut is efficiently computed using the new augmenting path based Boykov-Kolmogorov algorithm. To increase performance of the algorithm an initial pre-augmentation of all terminal links is proposed.

For fine tuning of the segmentation we propose a new method that uses a local graph representation to perform a pixel-based minimum-cut in specified areas. The method has automatic input of seed points. For difficult parts of the object boundary the locally found minimum-cut improves the result by overriding the global segmentation.

Two new methods to enter seed points for the min-cut/max-flow algorithm are demonstrated. The first method is based on finding shortest paths through elongated image structures and converting the found paths into dense rows of seed points. The other method automatically enters background seed points by subtraction of a background image.

The advanced Bayesian framework for alpha matting proposed by Chuang et al. is tested. It shows to have the potential for producing good alpha mattes even for difficult segments, but it occurs to be too slow to match Laerdal-Sophus A/S's need for interactivity. Instead, by applying Gaussian low-pass filtering to the alpha channel we obtain decent alpha transitions for simple borders in a few seconds. This simple approach can be allowed, at least, in the specific case Laerdal-Sophus A/S, because their image segments usually have simple borders.

All implementation is done in C++ to integrate with Picture Factory, which is an image editing application currently used by Laerdal-Sophus A/S.

Acknowledgments

We would like to thank our academic supervisor Ole Fogh Olsen, Associate Professor at the IT University of Copenhagen, Denmark, for excellent supervision throughout the thesis period.
Thanks to Ken Friis Larsen and Asger Kunuk Ottar Alstrup, from Laerdal-Sophus A/S, for friendly cooperation and guidance.

Contents

Chapter 1

Introduction

The work for this thesis is carried out as a part of the fulfillment of a Master of Science degree in Multi Media Technology at the IT University of Copenhagen, Denmark. It has been carried out in the period September 2004 to March 2005. The thesis is done in cooperation with Laerdal-Sophus A/S.

Interactive foreground extraction in natural images describes the process of separating a foreground object in a digital image from the background. It is referred to as image segmentation but is a subpart of general image editing. The relevance for efficient interactive tools for image segmentation has increased as digital image compositing has become more and more commonly used both by the professional graphics industry and by private users. Therefore, segmentation methods for interactive use have been an area of significant research and interest for many years. The problem is to provide a tool that assists a user in obtaining a segmentation of varying types of objects, in less time and in higher quality than could be obtained by manual processing. Providing such a tool calls for combining efficient algorithms for image analysis with a graphical user interface.

Though having general relevance, the objective of this thesis rise from the specific need for an interactive segmentation tool from Laerdal-Sophus A/S. Laerdal-Sophus A/S seeks to replace their current manual segmentation method with a semi-automatic segmentation method. Most image material tested in this thesis will therefore be provided by Laerdal-Sophus A/S.

1.1 Problem Statement

We wish to describe the framework of and develop a prototype tool for semi-automatic segmentation of foreground objects in natural images. The tool is developed in collaboration with Laerdal-Sophus A/S to whom we wish to propose a more efficient alternative to their current manual segmentation method. Shadowing of the segmented objects, which is an important part of Laerdal-Sophus A/S's

production, will be addressed to some extend. To allow future integration with Laerdal-Sophus A/S's production and to encourage further development based on the prototype tool, it should be implemented in C++ as an add-on to their image editing tool Picture Factory.

1.2 Introduction to Laerdal-Sophus A/S

Laerdal-Sophus A/S (LS) has specialised in making computer simulators for training of medical personal such as paramedics, doctors and military doctors. The simulators display 2D scenes that are composed by a range of segments coming from real photographs of a staff of actors.

When a scene in the simulator is composed, different personal such as doctors and patients and various medical equipment is combined in numerous ways. It is possible to swap the background of the scene independently of the foreground contents. Figure 1.1 illustrates the concept of taking a segment from one image and putting it into a new background setting.

Figure 1.1: $_{b}^{a}\ _{c}^{}\ _{d}^{}$ a) Crop of an original studio image of size 3072x2048 pixels. b) Hand made greyscale mask used to extract the desired segment. c) The desired segment. d) The segment composed into a new background.

The use of real photos has a high priority at LS because they want to present their users with realistic situation in the simulators. Image segments are therefore extracted from original images of a high resolution ranging from 6 to 11 megapixel in 24 bit colour. LS shoots up to 10.000 images in the making of a simulator. From each image they extract one or more segments. All pictures are manually segmented and the segments are combined to form new scenes.
A simplified production line for an LS simulator is shown in Figure 1.2. LS has special interest in optimising the image manipulation step. This step consists of three main tasks:

1. Segmenting objects from the original images
2. Creating shadows for the segmented objects
3. Quality control

of which we will focus on the segmentation task. The reason for LS's special

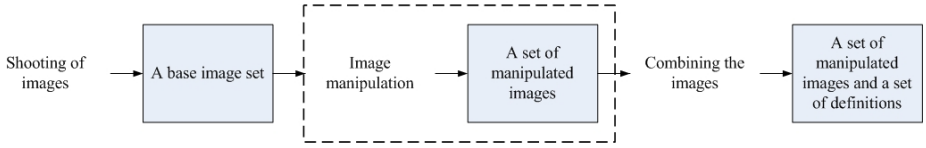


Figure 1.2: Simplified production line for an LS simulator. This thesis will be about the "Image manipulation" step.

interest in this particular step of the process is that manual segmentation and shadowing are very time consuming processes. On average it takes a trained operator 15 minutes to segment, shadow and perform quality control on a single image. More than half the time it takes for LS to create a new product is used for manual image manipulation. A reduction in the time spent on image manipulation would reduce the total production time and lower the production cost. This in turn would allow LS to enhance the quality of existing product types or present a wider selection of products to their costumers.

1.3 Current Image Manipulation

This section briefly describes the current image segmentation and shadowing tasks at LS. This is meant to provide the relevant background knowledge for the reader to understand the setting in which this thesis is worked out.

1.3.1 Image Segmentation

The first step is the image segmentation where the desired part of the image is identified and cut out.

Figure 1.1a shows a crop from an original image containing the object to be segmented. An operator manually draws along the border of the object and uses the alpha channel to indicate what is part of the foreground object and what is background. An 8 bit alpha channel is used to control the opacity for each pixel. Alpha values range from 0 to 255 - where 0 is a fully opaque (for background pixels), and 255 is fully transparent (for foreground pixels).
The alpha channel shown in Figure 1.1b is then used as a ”cookie cutter” to cut out the relevant object as seen in Figure 1.1c.

Before a segment is ready to use, the borders are softened to give a better blend with the new background image. The softening is done manually by blurring the border in the alpha channel. For the blurring is used a Gaussian kernel. Figure 1.3 shows the improvement achieved by softening.

Figure 1.3: $a\ b$ a) Applying this non-softened mask to the original image would give a segment with hard edges. b) The softened mask is used to give a better blend with the background.

In the current procedure the operator locally varies the width of the smooth transition in the alpha channel according to the hardness of the edge in the original image. Soft edges require a wide transition whereas hard edges are almost preserved using a transition only a few pixels wide.

1.3.2 Shadowing of Segments

Shadowing of segments is the second step in the image manipulation procedure. The attentive reader may have noticed in Figure 1.1d that the inserted segment looks rather unnatural in its new surrounding. The segment needs a shadow to convince the eye that the man in the image is really present in a 3D world with natural light sources.

Similarly to the border softening the shadows are created in the alpha channel. As illustrated in Figure 1.4 the shadow effect is obtained by making a soft transition in the alpha values across the segment boundary.
If done correctly, this will be perceived as a shadow in the composed scene because it will graduate the amount of colour passing through from the background image.

Figure 1.4: $_{a\ b}$ a) Normally softened mask. b) Softened mask combined with shadow.

Figure 1.4 shows an example of a shadow in the alpha channel. In Figure 1.5 we see the difference the shadow makes to the realism of the segment.

Figure 1.5: $_{a\ b}$ a) The shadowed segment. b) The shadowed segment on a new background. It adds a lot to the impression of realness.

LS currently have four different methods for creating shadows, each of which are suitable for different situations. We will not go into details with the methods, but sum up by saying that all four methods are tedious manual work.

1.3.3 Quality Control

The third step of the image manipulation procedure is testing the quality. The quality control is purely visual to see whether both the segment and the associated shadow look natural in various backgrounds. Quality control is currently done in two different parts of the production. First by the employee that makes the segmentation and later by a superior staff member.

1.4 Objectives

Given an overview of the current production at LS we will now setup some objectives for this thesis. The aim for this thesis is to put forward a proposal for a new image manipulation tool for LS. For this reason LS's expectations and needs have been important for the priorities. Obviously, it would be preferable for LS if this thesis resulted in a tool that could be used directly in their production. This is, however, not a direct goal because we also focus on the academic relevance of each topic we investigate. We have sought to accomplish a suitable balance between commercial interests and academic novelty.

Based on the above the following objectives with respect to implementation should be obtained:

- We should implement a prototype of an interactive segmentation tool that enables us to test the possibility of replacing LS's current manual segmentation method
- In order to test the total segmentation time including user input and application run-time the implementation should include a graphical user interface
- The prototype tool should be implemented, tested and documented within the time span of the thesis
- The prototype tool should be implemented as an integrated part of LS's C++ based image editing application, Picture Factory

Our interactive segmentation tool should not lower the quality of the segments compared to the current segmentation quality at LS. Furthermore, the total segmentation time used to obtain the necessary quality should not surpass the current segmentation times at LS. This calls for the following restrictions:

- The implementation should allow fast user input.
- The implementation should provide fast user feedback.
- The implementation should provide possibilities for fine tuning.

1.5 Thesis Outline

The structure of the rest of the thesis is as follows. Chapter two discusses a selection of previous works in the field of interactive foreground extraction. An existing technique is chosen to be the starting point for our work. Chapter three presents the theoretical framework for solving the segmentation task, while chapter four goes into more detail on the specific method chosen. Chapter four also addresses essential aspects of interactivity in the segmentation work. Chapter five addresses the

problem of preparing image segments for compositing against new backgrounds. Chapter six comments on the choice of software platform and describes the structure of the made implementation. In Chapter seven various tests are presented and performance in terms of image segment quality and time usage is discussed. Finally, Chapter eight summarises the thesis and gives directions for possible further work.
We have reserved the test in Chapter seven for the final application. Therefore, throughout the thesis we will perform closed tests of parts that are not included in the final application and parts that are smaller subparts of the final application.

Throughout the thesis we present a large amount of image examples. For detailed study of the image material we have included a digital version of the thesis on the attached CD-Rom.

Chapter 2

Method Selection

We wish to develop a customised interactive segmentation method that matches the above list of objectives. For this purpose we are interested in finding an existing method to be the starting point for our work. In this chapter we first give a review of some existing interactive segmentation techniques. Then we present a list of specific selection criteria that a given segmentation method should meet in order to be a possible candidate. Finally, by grading each method in each of the categories of the selection criteria, we will find the most suited method.

2.1 Review of Some Existing Methods

Our literature study has led us to a range of appealing articles that describe various approaches to the segmentation problem. Before selecting a method we will make a small review of some existing techniques. The existing interactive segmentation methods can roughly be divided into two groups: boundary-based and region-based. This review is based on reviews made by [11], [19] and [14] supplemented by our own comments.

2.1.1 Boundary-Based Methods

Boundary-based methods are methods that assist the user in marking the boundary surrounding the object of interest. Typically, the methods assist the user either by providing a snapping effect of a line or by enabling the user to paint the boundary with a wide pencil hereby lowering the precision needed for the boundary input.

2.1.1.1 Intelligent Scissors or Magnetic Lasso

This group of methods assist the user to directly select the object boundary using the mouse. By clicking near the object boundary the user will activate a line that, while dragging the mouse cursor in the image, snaps to the object contour. The user anchors the computed line by placing seed points on the object contour. To make a snapping line the method relies on finding the minimum cost path from the

last seed point to the current cursor position. The method has been a part of the Photoshop toolkit for some years [22].
As a couple of later references on the subject we can mention Mortensen & Barret [15] and Schenk et al. [20]. In Mortensen's review on "Vision-Assisted Image Editing" [14] he states that with a few additions "..., Intelligent Scissors should stay on the cutting edge of vision-assisted image editing."
Blake et al. [19] and Li et al. [11], however, are both less enthusiastic about the Intelligent Scissors. The general opinion is that the method requires too much user input to get a satisfying result and therefore is to time consuming to use. [11], furthermore, complains that *"If a mistake is made, the user has to 'back up' the curve and try again.*" and *"The close control required interferes with the user's ability to get an overview of their progress. It is difficult to zoom in and out of the image while you are dragging the pixel-accurate boundary line"*. We have worked with the tool in Photoshop and must agree that it is hard to correct mistakes without starting all over.

2.1.1.2 Matting

For matting methods the user specify a trimap. A trimap is an image overlay that divides the image pixels into three areas: foreground, background and unknown. Based on the knowledge of definite foreground and background the idea of matting is to estimate an alpha value (the degree of transparency) for all pixels in unknown. This will result in a smooth transition between foreground and background. The concept is often referred to as *alpha matting* and can solve very difficult segmentation tasks e.g. boundaries that consist of thin straws of hair.
The most used newer method for alpha matting is Bayesian matting [3]. This method uses a Bayesian framework and solves the problem using a maximum a posteriori (MAP) technique to estimate the α-value for each pixel in the unknown area. There exist newer matting techniques such as [21] and [19] which both claim to outperform [3] in both quality and speed.
Because of the smooth transition on the object boundary, matting methods are often used when the aim is to composite the segment into a new background. It seems, however, that matting methods are very slow and require a lot of user interaction. Furthermore, the methods require a lot of tuning parameters to be set by the user, which makes them less ideal for unexperienced users. Therefore, alpha matting is rarely used as an interactive stand-alone segmentation method.
Matting methods are however often combined with other segmentation methods to make the boundary blend into a new background. This is done in [15],[11] and [19].

2.1.2 Region-Based Methods

In region-based methods the user selects a region inside the desired object. Using this input the methods present the user with a segmentation suggestion that, de-

pending on the method, can be adjusted by specifying additional regions inside the object.

2.1.2.1 Magic Wand

This method is based on colour statistics alone. The user selects a point or region inside the object, and the method computes a segment of connected pixels that have similar colour statistics to the selected. The technique has been referred to as *region growing* [6].
This method has been a part of the Photoshop toolkit for many years [22]. The method, however, receives a lot of criticism in the literature: *"Because the distribution in colour space of foreground and background pixels have a considerable overlap, a satisfactory segmentation is not achieved."* [19]. *"...in terms of algorithmic properties, it seems somewhat slow, unintelligent, and unpredictable. Further, there is no ability to interact after the mouse click and the override capability is limited to backing up or combining results from multiple mouse clicks."* [14].
Again, we have tried the tool in Photoshop and must agree with the above criticism.

2.1.2.2 Graph-Based Methods

This group of methods are based on pixel labeling by energy minimisation. The user loosely selects some foreground and background pixels (seed points) using the mouse. This information is then used in an energy equation which is minimised using a graph-based minimum-cut/maximum-flow algorithm. The produced segment can be adjusted by entering more foreground and background points.
Graph-based methods are among the newest methods. [1] was the first to introduce graph cut methods as an interactive image editing method. [19] and [11] have, however, recently introduced their segmentation tools "GrabCut" and "Lazy Snapping". These are both graph-based methods but include some additional tools to improve speed, user interaction and quality of the segments.
From earlier student projects at the IT-University of Copenhagen, [16] and [18], it is our impression that graph-based methods can be rather slow. However, [19] and [11] use a new optimised algorithm presented by Boykov & Kolmogorov [2], to speed up the graph cut segmentation. They claim that their methods can give almost instant user feedback.

2.2 Evaluation of Five Methods

Based on the list of objectives from Section 1.4 we have set up a more thorough description of the selection criteria. The criteria are highly related and therefore difficult to separate.
We have made a preliminary selection of articles and have chosen five of the most appealing. Most of these articles are brand new (from 2004) and represent state-of-the-art methods within the field of interactive foreground extraction for image

editing and compositing.
After the presentation of the selection criteria we will make a grading of these 5 articles. The grades will be given in the range of 1-5, where 5 is the best and 1 the worst. We emphasise that the evaluation is done from our own experience and general impression of the articles. However, it is still a useful tool to direct us in our work.

2.2.1 User Input

The amount of user input is of great importance for this thesis. For many segmentation purposes it is considered an advantage if the method works automatically, without any user input. This is, however, not the aim for this thesis. Since the segmentation tasks at LS have a high variation in both complexity and types of foreground objects, the segmentation tool has to rely on a trained operator to provide some assistance when segmenting. The tool should simply make it faster for the operator to segment a given image without lowering the quality of the segment. The amount of user input reflects how much work the operator has to do to get a segment. This often has a high correlation with the total segmentation time. Therefore, we will value methods where a low amount of user input is needed.

2.2.2 Possibilities of Fine Tuning

Since we expect an operator to assist in the segmentation procedure, it is important that he can fine tune the result to ensure a high quality of the segment. The possibility of fine tuning is very important for a segmentation method to become fully interactive. Methods that allow a high degree of flexible user control are preferable.

2.2.3 Segmentation Time

Here we estimate whether the method will reduce the total time used for segmentation at LS. This means that the criteria combines the program runtime with the time expected for user interaction. It both requires a low amount of user input and a fast algorithm to, ideally, provide instant user feedback. The fastest method will get the highest grade.

2.2.4 Quality of Results

An important property of a given method is the quality of the results it produces. But to discuss the quality issue in a meaningful way, we have to define what a good result is. In order for LS to make direct use of an implementation done in the course of this thesis, it must meet the same requirements as their current manual segmentation.
The quality estimation is done based on the examples shown in the articles. However, since we have already made a preceding sort out of articles, the remaining articles all show results that seem good enough for the task at hand.

2.2.5 Possibilities for Shadow Creation

Even though it is not within the scope of this thesis to make a tool for shadow creation we keep in mind that this task also has high priority for LS. Hence, we will try to assess each method's ability to combine with a future shadow creation or shadow segmentation method.

2.2.6 Difficulty of Implementation

Though hard to evaluate, it is relevant to consider the implementation difficulty for each method. We find it important to choose a method we expect that we will be able to implement, test and document within the time limits of the thesis.
In this category we look at how well documented the methods are. A high grade is given if an article leaves us with a good impression of how the method should be implemented. Low grades will be given to the articles that are quite superficial on the used algorithms and implementation details.

2.2.7 Method Grading

Table 2.1 shows the grade that were given to each of the selected methods

Article/Method	User Input	Fine Tuning	Speed	Quality	Shadows	Difficulty	Sum
Graph-Based							
Lazy Snapping [11]	4	4	5	4	3	4	24
GrabCut [19]	5	3	4	4	3	3	22
Matting							
Poisson-Matting [21]	1	5	1	5	4	2	18
Bayes-Matting [3]	2	2	1	4	4	3	16
Intelligent Scissors							
Toboggan-Based Intelligent Scissor [15]	3	3	3	4	3	5	21

Table 2.1: Method grading. Grades from 1-5 are given where 5 is best and 1 is worst.

We will here make a few additional comments on some of the given grades.

User Input In this category the GrabCut method [19] receives the highest grading because the initial user input is made very simple and intuitivly by dragging a rectangle around the object of interest. On the other extreme we find the matting methods that seem to require expert knowledge in order to set program parameters to obtain good results.
The Toboggan-based Intelligent Scissors gets a middle grading. It has an intuitive means of input, but, unfortunately, it is a tedious work to mark the entire border.

Fine Tuning Here Poisson matting receives the highest grade because the article describes the broadest selection of clever fine tuning tools. The Lazy Snapping gets the grade 4 for a range of interesting fine tuning features and usability study included in the article.

Speed In terms of speed we estimate that Lazy Snapping and GrabCut are best and second best in both speed of input and computation. The iterative scheme used in GrabCut and the fact that it uses a pixel-based graph may slow computation, thus it is only given the grade 4.
The matting methods are known to be rather time-consuming, so they are given the lowest grade in this category.

Quality All methods seem to be able provide the necessary quality. Poisson matting receives a higher grade because it shows impressive result for very difficult images.

Shadows In this category the matting techniques receive a higher mark because we believe that the unknown area in their trimaps can become useful in a future automatic computation of a shadow for the segment.

Difficulty Here the best grading goes to the Toboggan-based Intelligent Scissors for a very well-documented algorithm. Lazy Snapping is also quite well described, and conceptually it seems more intuitive than GrabCut.
The Poisson matting gets a low grading for lack of details on the fine tuning tools.

2.3 Summary

As can be seen from Table 2.1 the method Lazy Snapping [11] receives the highest total score. We think that the method will meet the requirements for LS's segmentation problem and furthermore we think that the method will be a good set off for our work in the field of interactive segmentation. Therefore, we decide to use Lazy Snapping as our primary focus for this thesis.
Lazy Snapping is a wide combination of different techniques such as watershed segmentation, graph cut, alpha matting and different GUI-techniques. In our further work with the method we may decide to omit or replace parts of Lazy Snapping. Nevertheless, we will still refer to the method as Lazy Snapping.

Chapter 3

Energy Minimisation and Graph Cut

In the previous chapter we decided on using a graph based method to approach the segmentation problem at LS. Before going into a more detailed description of Lazy Snapping, we will in this chapter make a more general description on how graph cut can be used to solve an energy minimisation problem in computer vision. Furthermore, this section contains a detailed description of the Boykov-Kolmogorov augmenting path algorithm that we used to find the minimum cut of a graph.

3.1 Energy Minimisation

A common approach to a range of image processing tasks has been to formulate the problem in terms of an energy function for which the global minimum give the optimal solution to the problem. One example is the deformable contours - or snakes which are driven by minimisation of an energy function consisting of the so-called internal and external forces, [27]. The internal term carries information about the curve surrounding the object and is, at no time, influenced by image data. Minimising this term alone would cause the curve to shrink. The external term carries information about local edges in the image. Minimising this part alone would make the snake settle close to nearby edges in the image.
Another example of the use of energy minimisation is for finding the optical flow in an image sequence. This can be used to facilitate segmentation of moving objects in the sequence. A method for computing the optic flow was first presented by Horn and Schunk [7]. Horn and Schunk propose a two-term energy function. The first term assumes that the pixel intensity of an image does not change over time. The second term is a smoothness term used to encode the assumption that neighboring pixels have approximately the same local displacement vectors. By minimising this energy function for all pixels in two successive images, a displacement vector for each pixel can be estimated.

3.1.1 Pixel Labeling by Energy Minimisation

Not least, energy minimisation has been used for pixel labeling to achieve a segmentation of an image. In this thesis we will address the segmentation problem as an image labeling problem by energy minimisation.
Markov random field (MRF) provides the basis theory for image labeling. MRF is a branch of probability theory for analysing contextual dependencies [10]. In the case of image labeling, a label takes a discrete value in a set of M labels:

$$\mathcal{L} = \{L_1, L_2, ..., L_M\} \tag{3.1.1}$$

An image labeling problem is to assign a label L to each of the pixels in the image. In our case we only need to divide pixels into two labels, foreground and background, $\mathcal{L} = \{F(=1), B(=0)\}$. The energy function used is therefore the special case of the Gibbs distribution where only two labels are needed.

$$E(L) = \sum_{p \in \mathcal{P}} D_p(L_p) + \lambda \sum_{(p,q) \in \mathcal{N}} V_{p,q}(L_p, L_q), \tag{3.1.2}$$

where $\mathcal{P}$ is the set of pixels we want to label and $\mathcal{N}$ is all sets of neighboring pixels. $D_p(\cdot)$ is a regional data function, $V(\cdot)$ encodes neighbour relations. The goal is to find a labeling L of all pixels in $\mathcal{P}$ that minimises E.

The energy term $D_p(\cdot)$ can be interpreted as the posterior probability that a pixel belongs to either F or B. This energy term is based on having some information about the foreground and background. In Section 3.2.4 we will come back to how this information is obtained.
The neighbourhood energy term, $V_{p,q}(\cdot)$, is a local energy that represents the prior probability of coherency between two neighbouring pixels p and q. This term is often referred to as the smoothness term, in that it penalises discontinuities in the pixel neighbourhood.

3.2 Energy Minimisation by Graphs Cut

For different minimisation problems there are different minimisation schemes. To some problems there exist direct solutions whereas others require some sort of iterative process (e.g. gradient descent). However, the problem of finding the global minimum for image labeling is often difficult and time consuming because the energy functions generally are non-convex and in many dimensions. The problem can be solved using simulated annealing. Simulated annealing is, however, very time consuming, and it is shown that a guaranteed global minimum can only be found in infinite time [23].

Within the last few years minimisation problems for pixel labeling have effectively been solved using graph-based minimisation schemes. The basic idea is to make a

graph based on the energy function such that the minimum cut/maximum flow of the graph minimises the energy function. For more details on which functions are well suited for this approach we refer to [9]. It is, however, not within the scope of this thesis to address which functions can be minimised, but it is shown in [9] that Equation 3.1.2 can be minimised using graph cut.

3.2.1 Building the Graph

In a graph, $\mathcal{G}$

$$\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle \tag{3.2.1}$$

where $\mathcal{V}$ and $\mathcal{E}$ represent nodes and edges respectively, each node corresponds to one pixel in the image, such that all non-boundary pixels have connecting edges (n-links) to 4 other pixels if we assume a 4-neighbourhood.
The graph also has a two terminal nodes which are connected to all ordinary nodes by edges called t-links. Each of the terminals correspond to the different labels that can be assigned to nodes during the segmentation. The two terminals are referred to as the *source* terminal s, that is associated with F, and the *sink* terminal t, that is associated with B.
Each edge $e \in \mathcal{G}$ has a capacity c. This capacity is directly associated with the energy terms of Equation 3.1.2. The capacities of all t-links ($c(p, terminal)$) correspond to $D_p(\cdot)$, and similarly the capacities of all n-links ($c(p, q)$) correspond to $V_{p,q}(\cdot)$. The graph structure and the belonging vocabulary can be viewed in Figure 3.1.

A graph can be either directed or undirected. In our implementations we have been working with both graph representations, so we provide a brief description of each.

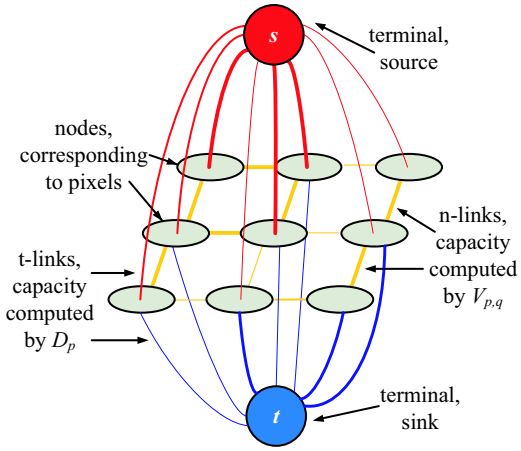


Figure 3.1: This figure illustrates the structure of the graph we use and the vocabulary used in connection with the graph.

3.2.1.1 Directed Graphs

In a directed graph each edge has a direction [4]. For our vision application a directed graph is constructed as described in [2]. Every pair of nodes (p, q) are doubly neighbour-linked (n-link) $p \rightleftarrows q$ by opposite directed edges (p, q) and (q, p).
All ordinary nodes are connected to both terminals by one edge. These t-links have the direction from the source to the ordinary node, $s \rightarrow p$ and from there further to the sink, $p \rightarrow t$. The directed graph is illustrated in Figure 3.2a.

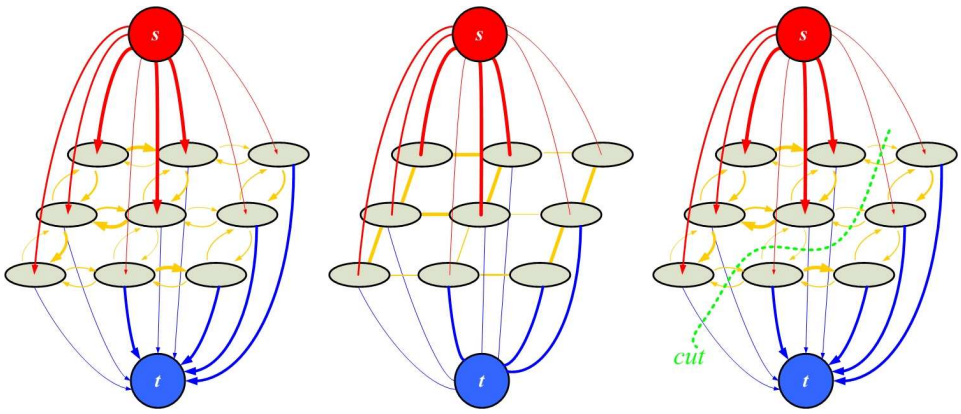


Figure 3.2: $_{a\ b\ c}$ a) A directed graph with two terminals. The thickness of the edges correspond to the edge capacities. b) An undirected graph with two terminals. c) A cut in a graph.

3.2.1.2 Undirected Graphs

In an undirected graph the edges do not have a direction. The undirected graph that we use in our work has a similar structure to the directed. Every pair of nodes (p, q) are connected by only one edge $(p-q)$. The ordinary nodes in the undirected graph also have t-links to both terminals. The undirected graph can be seen in Figure 3.2b.

3.2.2 Graph Cut

To obtain a segmentation of an image we seek to cut the graph for instance as shown in Figure 3.2c. A cut set, C is defined as all the edges that lie on the segmentation boundary, $C \subset \mathcal{E}$. The cut results in the graph

$$\mathcal{G}(\mathcal{C}) = \langle \mathcal{V}, \mathcal{E} \backslash C \rangle \tag{3.2.2}$$

in which s and t have been separated.
We are interested in finding a suitable graph cut algorithm that is capable of dividing the nodes of the graph into two disjoint subsets, F and B. In the end all nodes,

in the F will correspond the segmented object and all nodes in B will correspond to the background. An example of this is shown in Figure 3.3.

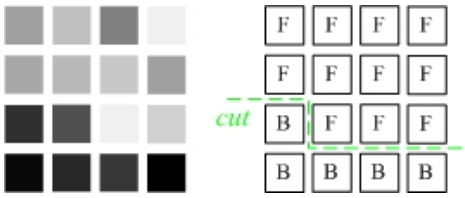


Figure 3.3: $_{a\ b}$ a) Unlabeled grey-scale image. b) Final labeling of the image after a min-cut has been made.

There exist two prevailing groups of algorithms for minimising via graph cuts. The type of algorithms that we exclusively deal with in this thesis is known as *augmenting paths*. The other group of algorithms that we shall barely mention is known as *push-relabel* methods. We have chosen only to work with the new augmenting path algorithm presented by [2]. The authors show that though is has a worse worst case complexity their algorithm in practice outperforms existing methods, including various push-relabel methods, that used to be the fastest for many years.

3.2.3 The Ford-Fulkerson Algorithm, Augmenting Paths

The augmenting path method was presented by Ford-Fulkersen [4] to find the max-flow in a graph $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$. Each edge $e \in \mathcal{E}$ has a capacity c. Besides the graph $\mathcal{G}$ we maintain a residual graph $\mathcal{G}_f$ which holds the remaining capacities c_f. $\mathcal{G}_f$ limits the possible flow that is allowed to pass through $\mathcal{G}$. Initially $\mathcal{G}_f = \mathcal{G}$.

The Ford-Fulkersen algorithm is given below:

Ford-Fulkersen Algorithm

```
while (path found s → t)
   find the bottleneck capacity Δf of P
   subtract Δf from all edges in P going from s → t
   add Δf to all edges in P going from t → s
end while
```

In each iteration of the algorithm we find a path, P from $s \rightarrow t$ and augment this path. Augmenting the path means finding the minimum remaining capacity (or bottleneck capacity) Δf along P and subtract Δf from all capacities in P.

$$for\, all\, c_f \in P\ from\ s \rightarrow t\ do: \ c_f = c_f - \Delta f \tag{3.2.3}$$

The rule of skew symmetry for directed graphs implies that Δf is added to all edges in the opposite direction [4], such that

$$for\,all\,c_f \in P\ from\ t \rightarrow s\ do:\ c_f = c_f + \Delta f \tag{3.2.4}$$

A saturated edge (an edge for which $c_f = 0$) cannot be a part of the path P. The principle is illustrated in Figure 3.4. As can be seen from Figure 3.4c each

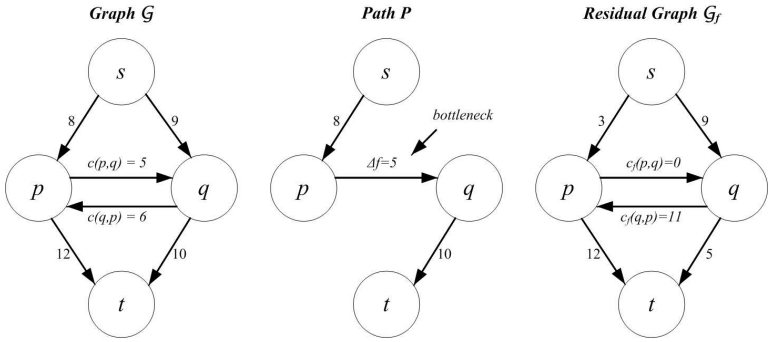


Figure 3.4: $a\ b\ c$ a) A directed graph with initial capacities. b) A path from sink to source with indication of the bottleneck capacity. c) The residual graph resulting from pushing Δf through the path, P.

augmented path results in at least one saturated edge. Each saturated edge is a part of the cut C. The algorithm terminates when no more augmenting paths can be found from $s \rightarrow t$. At this point we have obtained the min-cut or equivalently the max-flow

$$maxflow = |C| = \sum_{e \in C} c_e \tag{3.2.5}$$

The equivalence between the min-cut and max-flow is described in [4].

3.2.4 Seed Points

As mentioned in Section 3.1 we need some colour information from the image to compute the first energy term $D_p(\cdot)$. This information is obtained by choosing some pixels from the image. These pixels are referred to as seed points or seeds and correspond directly to nodes in the graph. There are two kinds of seeds, one related to the foreground $p \in \mathcal{F}$, and one related to the background $p \in \mathcal{B}$. Foreground seeds should be placed inside the object, to be segmented and background seeds outside the object.
Apart from providing the relevant colour information about the foreground and background, the seeds also impose a hard constraint on the final segmentation. This means that pixels $p \in \mathcal{F}$ will definitely end up inside the foreground object and pixels $p \in \mathcal{B}$ will become background.

3.2.5 The Boykov-Kolmogorov Algorithm

In the following we will consider the new min-cut/max-flow algorithm proposed by Boykov and Kolmogorov [2]. The algorithm is based on the Ford-Fulkerson augmenting paths, but contains some new features that makes it faster than existing algorithms. One new feature is that it simultaneously performs two breadth-first searches from the two terminals, s and t. This results in two search trees, S and T, with their roots in s and t, respectively. The other new feature that dramatically increases the speed is that the algorithm reuses the search trees and never rebuilds them from scratch.
In [2] the authors compare the new algorithm with the older Ford-Fulkerson based Dinic algorithm and two push-relabel algorithms. They mention that the new algorithm has a higher worst case complexity than these existing algorithms (see Table 3.2.5), because it does not necessarily find the shortest paths. However, they proceed to argue that this is not a major concern since their algorithm in practice is significantly faster than the others in various vision applications.

Method	**Worst Case Complexity**
Dinic	$O(mn^2)$
Push-Relabel-1 (†)	$O(n^3)$
Push-Relabel-2 (†)	$O(n^2\sqrt{m})$
Boykov-Kolmogorov	$O(mn^2\|C\|)$ (‡)

Table 3.1: Theoretical worst case run time complexities for some min-cut/max-flow algorithms, where n is the number of nodes and m is the number of edges. (†): Two different push-relabel methods by Goldberg & Tarjan. (‡): The of the Boykov-Kolmogorov algorithm depends on cost of the min-cut, $|C|$. Run times are taken from [2].

The Boykov-Kolmogorov algorithm runs a repeated three-stage procedure of so-called ”growth”, ”augmentation” and ”adoption”. In the growth stage the two search trees S and T are grown until an augmenting path, P $(s \rightarrow t)$ is found. The augmentation stage is where P is augmented to saturate at least one edge in P. Finally, the adoption stage ensures that branches of the search trees, that have been cut off by saturated edges, are replaced in the search tree or set free. The over all algorithm is given bellow. A is a list of active nodes, O is a list of orphan nodes. In the following subsections we will give more details about the individual stages.

Boykov-Kolmogorov Algorithm

Initialise: $S = \{s\}$, $T = \{t\}$, $A = \{s, t\}$, $O = \emptyset$
while true
  grow S or T to find an augmenting path P from s to t
  if $P = \emptyset$ terminate
  augment on P
  adopt orphans
end while

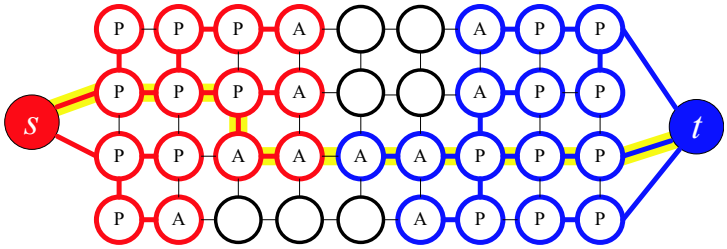


Figure 3.5: Two searches trees S (red) and T (blue) are grown from s and t. Active nodes ('A'), passive nodes ('P') and free nodes with black border. The yellow line indicates an augmenting path caused by the two trees touching each other.

3.2.5.1 Growth

Initially, all nodes are free in the sense that they do not belong to either of the search trees S and T. In the growth stage the two search trees expand by annexing free neighbouring nodes until the trees meet each other. When a node is annexed, it becomes active and is put in the active list A. Initially only the terminal nodes s and t are in the active list. An active node p can only annex a neighbour q if q is free and the connecting edge has $c_f(p \rightarrow q) > 0$. When an active node has no more neighbouring nodes to annex, it becomes passive and is removed from the active list. The concept is illustrated in Figure 3.5. This figure show a planar graph which is not the graph we are working with. The planar graph, however, illustrates the concept better.
If the algorithm runs in a directed graph, the residual capacity is $c_f(p \rightarrow q)$ when S grows and $c_f(p \leftarrow q)$ when T is grows.
When the search trees S and T meet, it results in a new augmenting path, P, $s \rightarrow t$ and the algorithm proceeds to the augmentation stage. Eventually, the algorithm stops when the whole graph has been searched, i.e. when A is empty. When this happens, all the nodes in the search tree, S makes up the resulting segmented foreground object.

3.2.5.2 Augmentation

In the augmentation stage the path P found in the growth stage is augmented as described in Section 3.2.3. Here the bottleneck capacity Δf is found and pushed through the path. At least one edge in the path is saturate. As a result of this saturation some nodes in S and T may become separated from their search trees because they no longer have valid parents, i.e. the edge from its parent q is zero, $c_f(q \rightarrow p) = 0$. A node severed from the tree is called an *orphan* node and is pushed into the orphan list O. When the whole path is augmented, the algorithm proceeds to the adoption stage.

3.2.5.3 Adoption

The orphan list, O is used to keep track of orphans nodes. During the adoption stage the nodes in O look for new parents. A neighbour q is a valid new parent to the orphan node p if the following criteria are met:

- p and q belong to same search tree
- The residual capacity, $c_f(q \rightarrow p) > 0$
- q has root in either of the terminals
- q is not a child of p

If an orphan cannot find a valid parent it becomes *free*. A freed node is disconnected from its search tree and behaves as if it never belonged to any tree. As such, free nodes are open to annexation in the next growth stage. All children of a freed node become orphans and are added to O. These new orphans must also look for new parents within the current adoption stage.

The adoption stage ends when there are no more nodes in O.

3.2.5.4 Choosing the Shortest Path

Choosing the shortest path from $s \rightarrow t$ increases the speed of the augmenting path algorithm. Boykov & Kolmogorov [2] argues that their algorithm only finds the shortest path the first time a path is found, which leads to a worse worst case complexity than the Dinic algorithm (Table 3.2.5). For a graph like the one we use, Figure 3.2, it is obvious that the shortest path from $s \rightarrow t$ is through the t-links, resulting in the path $s \rightarrow p \rightarrow t$. Since all pixels are connected to both s and t by t-links, it should be possible to find as many *shortest paths* as the number of nodes n in the graph. We therefore argue that it is possible to augment n shortest paths in the start of the algorithm. This will initially result in the saturation of half of the t-links in the graph. After augmenting these n shortest paths, it is not possible to predict the length of the paths.
The above observation will show to have significant influence on the runtime of the

search algorithm. Since we now know that half the t-links can be saturated in the first n paths, we argue that this saturation can be done already when calculating the t-link capacities of a node which would result in a pre-augmentation of t-links. Doing this will be faster than letting the algorithm search the graph to saturate these edges. To illustrate the concept we have in Figure 3.6 made an example showing the graph for which half the t-links have been saturated before running the Boykov-Kolmogorov search algorithm. The graph is made from the image in Figure 3.3. The test in Section 7 will show that this approach reduces the segmentation time but still finds the min-cut of the graph.

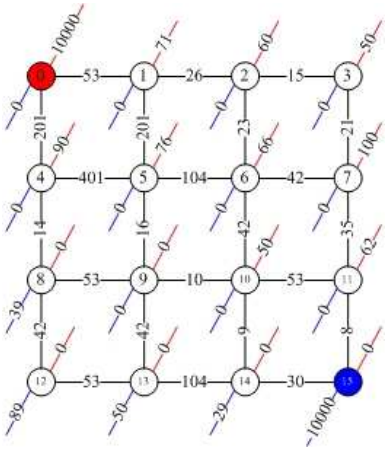


Figure 3.6: By pre-augmentation half of the t-link are initially saturated.

Another means of reducing the length of paths is used in the adoption stage. If an orphan node has more than one valid parent, it chooses the parent closest to s or t. This will reduce the average path length, but it will still not ensure that we find the shortest path.

Chapter 4

Lazy Snapping

This chapter presents the chosen method, Lazy Snapping. The method is presented in the article by Li et al. [11], but our implementation will differ from the article at some points. We will come back to this throughout the chapter. First, we will give an overview of the method.

4.1 Overview

Lazy Snapping is based on energy minimisation via graph cut. However, it differs from the more traditional graph segmentation method described in [1] by carrying out an initial pre-segmentation of the image. Instead of building a graph where each node is a pixel, we now build a graph where each node represent a small segment. The purpose of the pre-segmentation is to reduce the size of the graph which will increase the segmentation speed.
An important part of the Lazy Snapping method is the possibility to fine tune the segmentation result. [11] describes three fine tuning methods. The first is based on inserting more background and foreground seeds which is a standard fine tuning method in the framework of graph cut segmentation.
For the second and third method is made a curve representation in which the segment boundary is represented by vertices connected by line segments. Both methods are based on changing the position of the vertices which in turn changes the segment. Due to the time constraint of this thesis we will not include the curve based fine tuning method.
We will, however, present a new fine tuning method based on a local graph in the image. This method allows fine tuning by applying graph cut based locally on pixels instead of small segments.

4.2 Pre-Segmentation

For the pre-segmentation step we perform an oversegmentation of the image into small regions. For this purpose watershed segmentation methods have good properties because they provide a fast segmentation of the whole image. Furthermore, watershed methods do not necessarily need any user input, which enables pre-segmentation to be done automatically without user interaction. Lazy Snapping uses the watershed method proposed by Vincent & Soille [25]. We have, however, chosen to use the toboggan based watershed method described in [15]. The reason for this choice is that we operate with floating point gradient magnitude images. The Vincent-Soille [25] algorithm requires a sorting of all pixels in the gradient magnitude image, which is very costly when the image consists of floating points. The toboggan based method is not slowed down by a floating point gradient magnitude images, and is therefore more efficient than [25]. Furthermore, [15] states that: *"The regions produced by tobogganing are effectively identical to the catchment basins produced by applying Vincent-Soille [25]."*, which tells us that the quality of the pre-segmentation is kept intact.

4.2.1 Toboggan Watershed Segmentation

The toboggan based watershed method is quite intuitive to understand. Having a landscape one can imagine a toboggan sliding downhill until a bottom of a valley is reached. If the landscape is represented by a gradient magnitude image, G, then the bottom of the valley represent a local minimum in G. Every local minimum is labeled with a unique label. For each local minimum there will be a new segment. All pixels that were passed on the way down hill are labeled with the same label as the local minimum. If we slide into an already labeled pixel on the way down, all pixels in the toboggan path are given this label without sliding all the way to the local minimum.
Before going into further details with the toboggan algorithm, we will describe how the gradient magnitude image is obtained in a multi-scale framework.

4.2.1.1 Multi-Scale Gradient Magnitude

To compute the gradient magnitude we differentiate the two-dimensional Gaussian kernel

$$N_\sigma(x,y) = \frac{1}{2\sqrt{2}\pi\sigma} e^{-\frac{1}{2} \frac{x^2+y^2}{\sigma^2}} \tag{4.2.1}$$

with respect to both x and y

$$\frac{\partial N_\sigma}{\partial x} = N_\sigma(x,y) \cdot \frac{-x}{\sigma^2} \quad , \quad \frac{\partial N_\sigma}{\partial y} = N_\sigma(x,y) \cdot \frac{-y}{\sigma^2} \tag{4.2.2}$$

Since we are working with colour images, we have to decide on how to compute the gradient magnitude for an image with three channels. [15] suggests to sum the

squared gradient magnitude over each colour channel:

$$G_\sigma = \sum_b \left[I_b * \frac{\partial N_\sigma}{\partial x} \right]^2 + \left[I_b * \frac{\partial N_\sigma}{\partial y} \right]^2 \tag{4.2.3}$$

where I_b is a colour channel of the original image I.

Another method is, however, to use the max-norm where the maximum gradient of each of the colour channels is selected

$$G_\sigma = \max_b \left(\left[I_b * \frac{\partial N_\sigma}{\partial x} \right]^2 + \left[I_b * \frac{\partial N_\sigma}{\partial y} \right]^2 \right) \tag{4.2.4}$$

Which of these two methods produces the best segmentation will be tested in Section 4.2.1.4.

To consider both large and small structures in the image the gradient magnitude is found in a multi-scale fashion. The scale selection is done by selecting the maximum of all G_σ values in every point in the image

$$G(x, y) = \sqrt{\max_\sigma (G_\sigma(x, y))} \tag{4.2.5}$$

4.2.1.2 Scale Normalisation and Scale Selection

In [15] it is described that the gradient magnitude is selected among the largest $G_\sigma(x, y)$. However, according to Lindeberg [13]: *"...the amplitude of the variations in a signal will always decrease with scale"*. This means that high values of σ, G_σ is less likely to be chosen in Equation 4.2.5. This leads [13] to suggest a normalisation of G_σ over scale. The normalised gradient magnitude, $G_{\sigma normal}$, is found by multiplying with σ^2:

$$G_{\sigma^2 - normal} = \sigma^2 G_\sigma(x, y) \tag{4.2.6}$$

Furthermore, [12] introduces γ-normalisation where γ is a tuning parameter which depends on what sort of feature is extracted (edges, ridges, corners or blobs).

$$G_{\gamma - normal} = (\sigma^2)^{\gamma/2} G_\sigma(x, y) \tag{4.2.7}$$

For edge detection, as in our case, [12] suggests to set $\gamma = \frac{1}{2}$. In order for us to decide which normalisation scheme to use, we have made a comparison of the three different approaches. For each normalisation method Figure 4.1 illustrates which scales were used to obtain $G(x, y)$. Black pixels show where the lowest scale was used and white pixels where the highest scales were used. Grey pixels show the in-between scales. When evaluating this figure we take the account that:

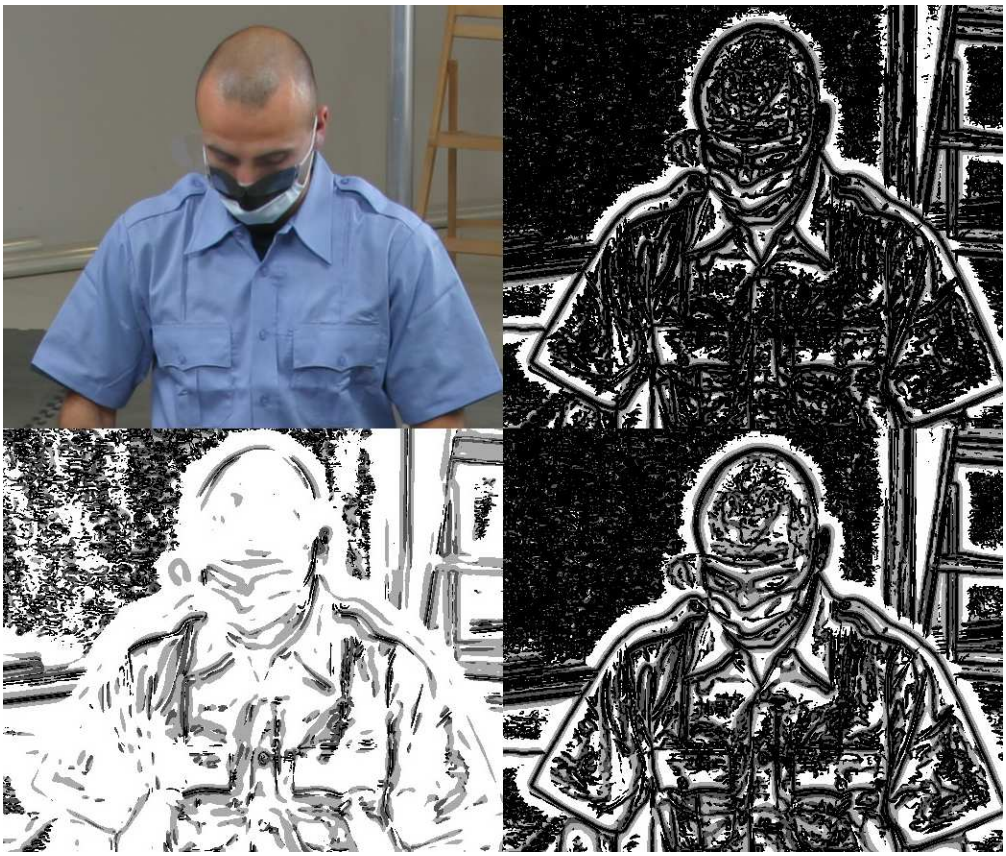


Figure 4.1: $\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}$ All images shown here is subpart of a 1024x1024 image. Black pixels represents the lowest scale, white highest scale, gray the in-between scales. σ=$\{$1, 2, 4, 8$\}$. a) Original image. b) No normalisation. c) Normalised as in Equation 4.2.6. d) γ-normalised as in Equation 4.2.7, $\gamma = \frac{1}{2}$.

- Sharp edges should be kept intact and should not be blurred. This means that if a sharp edge occur, we will prefer that a low scale is selected because this will blur the edge the least.
- In equally colored areas we prefer that a higher scale is selected because this, when applying the watershed segmentation, will result in larger segments (and therefore fewer segments). This is preferable in later steps of the Lazy Snapping method because fewer segments will result in reduced computation time.
- If the same scale is always selected, the multi-scale framework is useless and a waste of computation time. In this case we might as well only calculate for only one scale.

Figure 4.1 shows that it is important which normalisation method is used. In Figure 4.1c the contour of the man is selected from the highest scale. This would result in a blurring of the edges and consequently a less accurate pre-segmentation.
We would choose not to normalise as in Figure 4.1b, or to use $\gamma-$normalisation as in Figure 4.1d. Both of these cases select the lowest scale at sharp edges which would preserve the edges well. However, $\gamma-$normalisation selects the highest scale more often, which gives the least amount of segments. In Section 4.2.1.4 we will look at pros and cons of the types of normalisation.

Since we are not interested in blurring edges but still interested in using the advantages of multi-scale edge detection, an additional proposal could be to test the method proposed by Perona & Malik [17]. This method preserves sharp regional boundaries in a scale-space framework. However, because of the time limits for this thesis, we will not explore this area any further.

4.2.1.3 The Toboggan Algorithm

When we have the gradient magnitude image we can proceed to compute the toboggan watershed segmentation as described in Section 4.2.1. The image is scanned in row major order and we keep sliding until all pixels are labeled.
Below is given the toboggan algorithm taken from [15]:

Toboggan Algorithm

```
Input:  G(p) (Gradient magnitude at pixel position vector p.)
Data Structures:  N(p) (4 connected neighborhood of p.)
Output:
   T(p) (Toboggan direction vector at p.)
   L(p) (Region label at p (initialized to nil for all p).)
Algorithm:
   regions←0; (Initialize # of regions.)
   for each p do begin (Scan image in row major order.)
      q←p;
      repeat (Slide to labeled pixel or local minimum.)
         min←G(q); q´←q;
         for each r∈(q) do begin (Find lowest gradient neighbor.)
            if G(r)≤min then begin
               min←G(r); q´←r;
            end if
         end for
         T(q)←q´- q; q←q´; Set slide direction and slide.
      until L(q)≠nil or T(q)=0
      if L(q)=nil then begin (If local minimum is unlabeled,)
         L(q)←regions;  (assign a unique label.)
         regions←regions+1;
      end if
      r←p;
      repeat (Repeat slide to label unlabeled pixels.)
         L(r)←L(q); r←r+T(r);
      until L(r)≠nil
   end for
```

In Figure 4.2 is given an example of the use of the toboggan algorithm. To con-

Figure 4.2: $\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}$ Pre-segmented image using the toboggan watershed algorithm. The colour of each small segment is calculated as the average colour of the pixels in the segment. a) Original image. b) Segmented without normalisation, NR=15.4. c) Segmented with normalisation as in Equation 4.2.6, NR=41.5. d) Segmented with γ-normalisation as in Equation 4.2.7, $\gamma = \frac{1}{2}$, NR=17.8.

tinue the discussion from last section on scale normalisation, we run the watershed segmentation on all three gradient magnitude images, i.e. the non-normalised, the scale normalised and the γ-normalised. Since the purpose of the pre-segmentation is to reduce the size of the graph, we want to set up a measure for how much it is reduced. This is done by calculating the pixel-to-segment ratio (referred to as the Node Ratio (NR $= \frac{number\ of\ pixels}{number\ of\ segments}$)). A high NR value indicates a high reduction in size. For comparison [11] has NR values in the range of 10-24 in their examples.

Figure 4.2c shows, as expected in Section 4.2.1.2, that normalisation by σ^2 blurs the edges too much which results in rather unprecise segments and very high NR value (NR=41.5). Figure 4.2b and 4.2d show very good pre-segmentations, with

4.2d having a slightly higher NR value as expected (NR=17.8).

4.2.1.4 Examination of Small Segments

When finding the gradient magnitude for colour images, we discussed whether to use the sum over each colour channel (Equation 4.2.3) or the maximum over each channel (Equation 4.2.4). In Figure 4.3 we have made a small test of the two cases. This test clearly shows that choosing the maximum over each colour channel gives the better result. Figure 4.3b shows that the man's ear is cut of when using the sum over each colour channel while, when using the max over each colour channel, Figure 4.3c, the ear is well preserved.

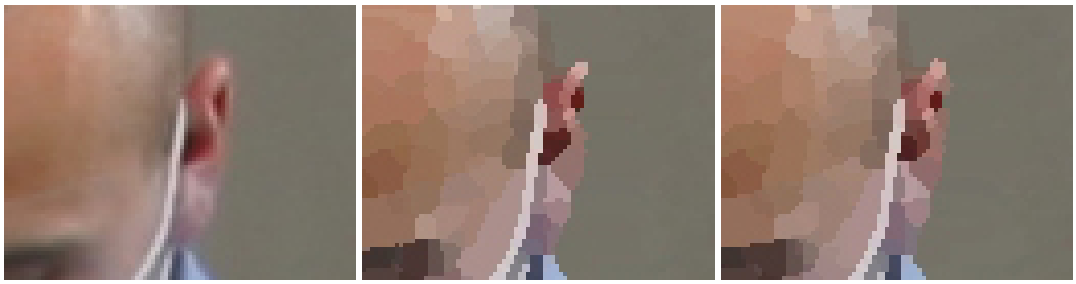


Figure 4.3: $_{a\ b\ c}$ σ=$\{1, 2, 4, 8\}$ a) Original image. b) Pre-segmentation using the sum over colour channels Equation 4.2.3. c) Pre-segmentation using maximum over channels Equation 4.2.4.

To show that normalisation gives larger segments we have in Figure 4.4 given an example that shows a zoom of the toboggan watershed segmented image from before. This example shows that using γ-normalisation gathers some of the smaller segments into one bigger. However, other examples show that using γ-normalisation sometimes blurs some edges too much which results in a segment being created across edges. Figure 4.5 shows two examples of this. Since the pre-segmentation is a preliminary step for the graph cut segmentation, errors like these will effect the quality of the graph cut segmentation. Thus, we find it more safe not to use γ-normalisation though it may result in a larger graph.

Figure 4.4: $\begin{smallmatrix} a & b & c \\ d & e & f \end{smallmatrix}$ a) Crop of original image. b) No scale normalisation σ={1, 2, 4, 8}. c) No scale normalisation σ={1, 2, 4, 8, 16, 32}. d) σ=1. e) γ-normalisation σ={1, 2, 4, 8}. f) γ-normalisation σ={1, 2, 4, 8, 16, 32}.

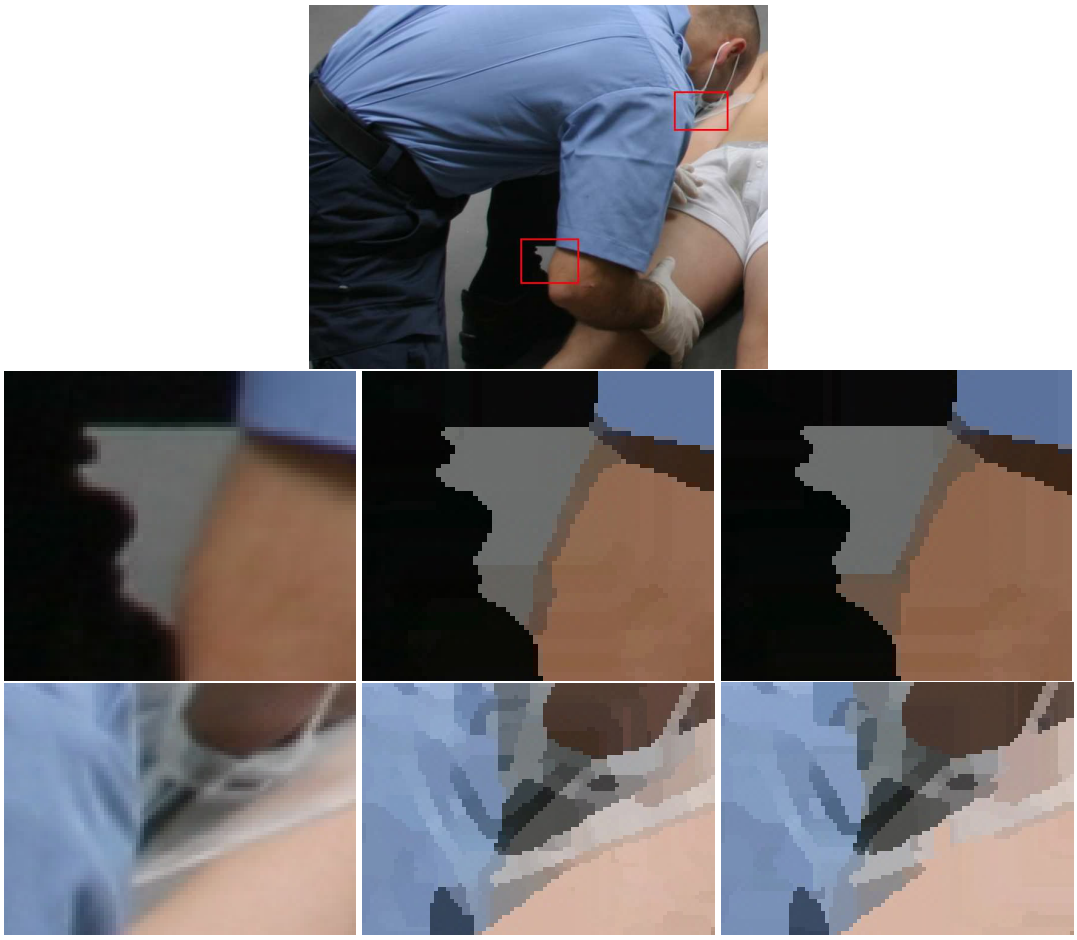


Figure 4.5: $\begin{matrix} & a & \\ b & c & d \\ e & f & g \end{matrix}$ a) Original image showing zoom areas. b) Zoom of original. c) Watershed of b, no normalisation, σ={1, 2, 4, 8}. d) Watershed of b, γ-normalisation, σ={1, 2, 4, 8}. e) Zoom of original. f) Watershed of e, no normalisation, σ={1, 2, 4, 8}. g) Water shed of e , γ-normalisation, σ={1, 2, 4, 8}.

4.3 Creating the Graph

Having the pre-segmented image the next step is to create the graph. As mentioned earlier, we intend to use every small segment from the toboggan watershed segmentation as a node in the graph. The concept is illustrated in Figure 4.6.

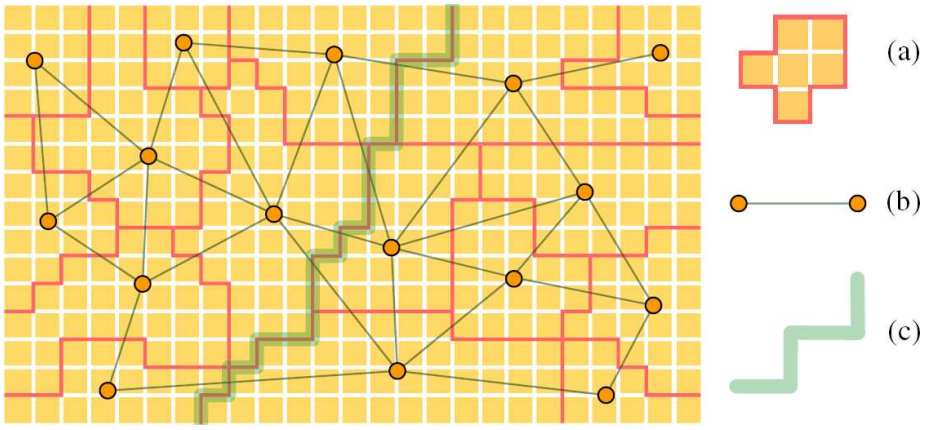


Figure 4.6: a) Segments from the toboggan watershed segmentation. b) Nodes and edges in the graph. c) Boundary from the graph cut segmentation. (Image from [11])

Since a node is now representing a segment, it has to carry more information than if it had represented a pixel. The colour of a node is calculated as the average of the pixels it represents.

On pixel level we normally have 4 or 8 neighbours per pixel. When pre-segmenting, the number of neighbours varies from node to node. We define two nodes as being neighbours if the small segments they represent have a shared boundary. Potentially, this means that a node can have as many neighbours as the number of pixels on its border. However, our experiments with the toboggan watershed algorithm in section 4.2.1.3 show that each segment in average has 5 to 6 neighbours.

4.4 Setting the Energy Function

In section 3.1 we described the energy function we are minimising:

$$E(L) = \sum_{p \in \mathcal{P}} D_p(L_p) + \lambda \sum_{(p,q) \in \mathcal{N}} V_{p,q}(L_p, L_q) \tag{4.4.1}$$

However, no specific formulae were given for the energy terms $D_p(\cdot)$ and $V_{p,q}(\cdot)$. In the following we will go into more detail on how the two terms are set.

4.4.1 Likelihood Energy, $D_p(\cdot)$

This term represents how similar the colour of a node is to the selected foreground and background seeds, denoted $\mathcal{F}$ and $\mathcal{B}$ respectively. Furthermore, for the graph $\mathcal{G}$, $D_p(\cdot)$ represents the t-link capacities, $c(p,s)$ and $c(p,t)$. To make the similarity estimation the user starts by entering foreground and background seed. How this is done will be described in Section 4.5. The colour samples of seeds in $\mathcal{F}$ and $\mathcal{B}$ are clustered by the K-means method into K clusters. For each node p in the graph $\mathcal{G}$ we calculate the shortest distance in colourspace from the node colour $C(p)$ to all clusters in each of the sets, $K_n^{\mathcal{F}}$ and $K_n^{\mathcal{B}}$:

$$d_p^{\mathcal{F}} = \min_n ||C(p) - K_n^{\mathcal{F}}|| \quad , \quad d_p^{\mathcal{B}} = \min_n ||C(p) - K_n^{\mathcal{B}}|| \tag{4.4.2}$$

$D_p(\cdot)$, or equivalent the edge capacity to each of the terminal nodes s and t, is then calculated as shown in Table 4.1.

t-link	capacity	for
$e(p,s)$	∞	$p \in \mathcal{F}$
	0	$p \in \mathcal{B}$
	$\frac{d_p^{\mathcal{B}}}{d_p^{\mathcal{F}}+d_p^{\mathcal{B}}}$	$p \in \mathcal{P}, p \notin \mathcal{F} \cup \mathcal{B}$
$e(p,t)$	0	$p \in \mathcal{F}$
	∞	$p \in \mathcal{B}$
	$\frac{d_p^{\mathcal{F}}}{d_p^{\mathcal{F}}+d_p^{\mathcal{B}}}$	$p \in \mathcal{P}, p \notin \mathcal{F} \cup \mathcal{B}$

Table 4.1: Terminal edge capacities. It should be noted that the edge weights specified in this table differ from what is described in [11]. We are however sure that [11] made a printer's error and unintendedly exchanged the edge capacities for s and t.

The edge capacities specified in Table 4.1 ensures that seeds selected by the user to be in $\mathcal{F}$ or $\mathcal{B}$ remain in the selected group. Nodes, that are not in either $\mathcal{F}$ or $\mathcal{B}$, are encouraged by the above equation to have a label similar to the seeds in $\mathcal{F}$ or $\mathcal{B}$. The equation is made so that the total t-link capacity for a node sums up to one.

$$c(p,s) + c(p,t) = 1 \tag{4.4.3}$$

If $c(p,s) > c(p,t)$, p is initially most likely to be part of the foreground and vice versa.

4.4.1.1 K-Means Clustering

To cluster the two sets of seeds, $\mathcal{F}$ and $\mathcal{B}$, we use a standard K-means clustering algorithm as described in [26]. There are two major reasons to cluster foreground

and background seeds. Firstly, if we did not cluster the seeds, we would have to compare all nodes in the graph with all seeds in $\mathcal{F}$ and $\mathcal{B}$ in order to obtain the minimum distances to each set of seeds. $\mathcal{F}$ and $\mathcal{B}$ can easily include more than a thousand seeds each. Thus, to compare all nodes in the graph with all seeds in $\mathcal{F}$ and $\mathcal{B}$ would be very time consuming. However, by clustering $\mathcal{F}$ and $\mathcal{B}$ into K mean centres, we reduce the number of comparisons. As [11] we will set $K = 64$.

The other important benefit of clustering $\mathcal{F}$ and $\mathcal{B}$ is that it reduces noise because we now use a mean colour $K_m^{\mathcal{F}}$ and $K_m^{\mathcal{B}}$ in the computation of $D_p(\cdot)$

4.4.2 Prior Energy, $V_{p,q}(\cdot)$

The prior energy term $V_{p,q}(\cdot)$ represents a node p's similarity with its neighbour q. In the graph $\mathcal{G}$, $V_{p,q}(\cdot)$ represents the n-link capacity $c(p,q)$. $c(p,q)$ is calculated as a gradient expression

$$V_{p,q}(\cdot) = c(p,q) = \frac{1}{||C(p) - C(q)||^2 + 1} \tag{4.4.4}$$

One, is added in the denominator to ensure that it does not become, zero when $C(p) = C(q)$. The capacity between p and q will be high if they are similar in colour and low if they are different in colour. The neighbour capacity is shown as a function of the colour gradient in Figure 4.7. This fits well with what was explained in Section 3.2.3 on how the minimum cut is found. Edges in the graph with low capacities will be saturated first making it likely to get a cut between neighbouring nodes that are very different in colour.

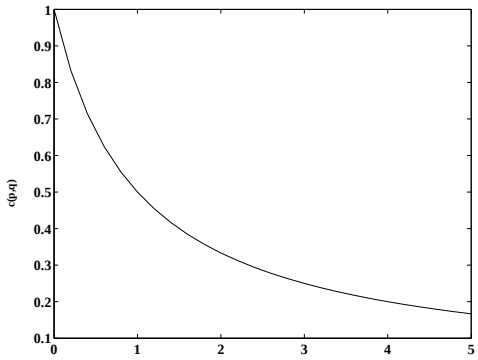


Figure 4.7: Showing the neighbour capacity $c(p,q)$ as a function of the colour gradient $||C(p) - C(q)||^2$ between p and q.

4.5 Entering Seed Points

This section presents some important issues related to the seed points that are essential to the min-cut/max-flow algorithm. It is relevant to look at which implications the human operator has on the system in entering seed points. Obviously, seed points entered by an operator allows a portion of subjectivity into the segmentation process. This can be thought of as a weakness of algorithm, however, if used correctly, it can be turned into great benefit in terms of quality of segmentation result.
The seed points both serve as a sampling spot for the regional colour statistics and as a hard constraint for the image labeling. We will look at the details of these roles to show how the quality of the segmentation is effected by the number of seed points and their location.
We will present some rules that can be followed when entering seeds in order to obtain a good segmentation.

4.5.1 User Interaction

As described in Section 4.4.1 after the user has entered seed points in both foreground and background the t-links capacities are set according to Table 4.1.
The actual input is done in a graphical user interface using a mouse-controlled brush tool. The user chooses between the two types of seeds by left and right clicking and enters a suitable number of points by drawing in the input image. All nodes touched by the brush will become seeds of the chosen type. It is possible to change the size of the circular brush to include more or less nodes per mouse click. Foreground seeds are marked with green, and background seeds are marked with red.

4.5.2 Minimum Requirement

The minimum requirement for the algorithm to run is one foreground seed and one background seed. However, this will only suffice for simple artificial images e.g. a binary image with a white box on a black background as in Figure 4.8.

Figure 4.8: An example of a simple image, for which only one foreground seed and one background seed is needed.

For larger and more complex images like natural images, more seeds are needed to produce the desired segmentation. Therefore, it is convenient to have a drawing

tool, for entering seeds. In the following we will describe the role of the seeds in more detail.

4.5.3 Colour Statistics

For every seed point, $p \in \mathcal{F}$ or $p \in \mathcal{B}$ a colour sample is added to either the foreground or background statistics. The number of seeds should be large enough to provide sufficient colour information from all the different parts of both foreground and background. What is sufficient depends highly on the image. The operator should consider the relative colour variation of the different areas. A rule of thumb is that many seeds should be placed in highly inhomogeneous areas, whereas few seeds are needed to capture the relevant information from an almost equally coloured area.

4.5.4 The Neighbourhood

The neighbourhood term regards discontinuities between neighbouring nodes. This means that diffuse or soft edges are more difficult to segment than hard and very distinct edges.
For the images we are looking at there can appear hard edges on the border of the desired segment, but also inside the segment. This fact can result in a disconnected segmentation if nothing is done to prevent it. A rule of thumb is therefore always to enter seed points across hard edges inside the segment. This will enforce a connected segment. The concept is illustrated in Figure 4.9.
Another important property that relates to the graph neighbourhood is the fact that when a node is pointed out as a seed point, it is ensured that the t-link to the respective terminal, $term \in \{s, t\}$ is never saturated because $c(p, term) = \infty$. Therefore, the seeds are guaranteed to remain direct children of their terminal, and are as such extremely important for the structure of the two search trees. In Section 3.2.5.4 we described how orphan nodes should choose the parent closest to the terminals s or t in order to reduce the length of the path. Keeping this rule in mind, the fact that seeds have their terminal as parent means that orphans will always be adopted by seeds in preference to non-seed valid parents. This leads us to argue that shorter paths, $s \rightarrow t$ can be obtained by entering more seeds. Shorter paths in turn are known to result in a faster segmentation.

4.5.5 The Neighbourhood Weight

The neighbourhood term, $V_{p,q}(\cdot)$ has a weighting parameter, λ, that increases or decreases the capacities of the n-links. λ balances the amount of influence coming from the neighbourhood compared to the influence from colour statistics. That is, an increase in λ gives relatively higher capacities in the n-links, which means that t-links are more likely to saturate first. This in turn yields a denser segmentation result that is less likely to contain disconnected segments (”islands”).

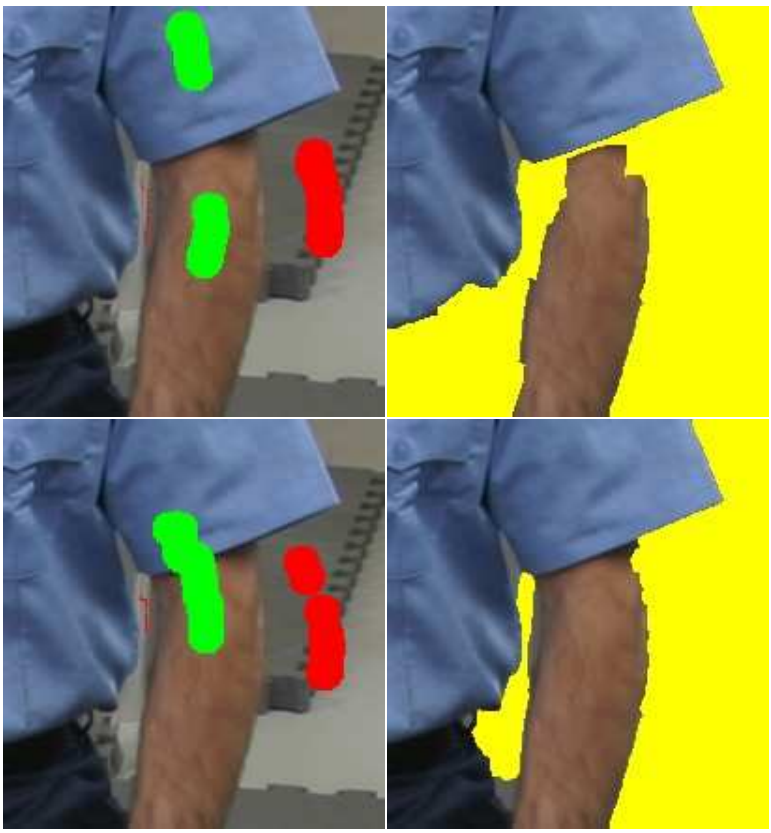


Figure 4.9: $\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}$ a) Entered seed points used to get segment in b. b) The segmentation result from the seeds in a. Observe that the segment is disconnected because a hard edge appears between the arm and the shirt. c) Entered seed points used for d. d) By entering seeds across the hard edge we obtain a connected segment.

If $\lambda = 0$, all n-links are initialised to zero in which case only the colour input from the seeds control the segmentation. This is probable to result in a highly scattered segmentation since all information on the neighbourhood smoothness is omitted. On the other extreme, if λ is set to a very high value we practically disregard the colour information. This would correspond to setting all non-seed t-links to zero, which would reduce the role of the seeds to entry points for the graph search into image. This would force the operator to input seeds of the relevant type into all parts the image to get a correct segmentation. We will to test this in Section 7.1.3.

4.5.6 LS Segmentation Example

We have received a number of segmentation examples form LS. A quick study of these images shows some image features that are likely to be difficult for our segmentation method. We show an example to analyse some common types of problems we may encounter when the segmentation is done semi-automatically.

Figure 4.10a shows a crop of a typical LS image. The manual segmentation result in Figure 4.10b done by an LS employee is what we want to achieve using the semi-automatic method. In Figure 4.10a the medic's shirt contains different shades

Figure 4.10: a b a) A 612x758 crop of an original LS image. The aim is to segment the medic's torso, head and right arm as well as the forceps. b) Result of manual segmentation.

of its main colour, blue. Thus, to get a good colour representation a sweep by the seed brush should be made across the shirt.
It may also be a good idea to input a line of seeds crossing the transitions from head to shirt and from shirt to his right arm. This is done to ensure that the subparts are gathered in one connected segment.
Also the forceps in Figure 4.10a may need some extra attention. Foreground seeds should be placed closely in such elongated structures, to avoid problems of saturation of edges inside the desired foreground area. Too long distance between the seeds could give a scattered segment consisting of separated parts. This effect may even be increased since the forceps also have spots of highlights, that have a different colour. Thin elongated structures in general present a challenge for graph-cut based segmentation methods because entering seeds into this type of objects require high precision from the user. We will return to this topic in the next section and propose a method for easier entering of seeds into such structures.

4.5.7 Entering Seed Points Into Elongated Structures

By the example of the forceps in the previous section, we argued that it would be both tedious and difficult to input all the necessary seed points correctly. An additional tool for inputting seeds in thin elongated structures would be of outmost convenience.
In the following we will look at a problem that is sufficiently similar to our case to suggest that we can benefit directly from its solution.
Dam and Lillholm [5] present a graph-based tool called *Interactive Route Measuring* for finding routes along streets in a scanned street map. The user enters a number of control points in a map and the program computes the shortest path along streets that connects the points. A special edge cost function is constructed to force all parts of the path to follow and remain inside the streets in the image. The method uses a standard Dijkstra shortest path algorithm [4] to find the shortest path between the control points.
We claim that the method can be extended and applied in our context as an efficient help in entering seed points into thin elongated structures. The operator would then have to enter a few control points inside the elongated object and the program would find a path through the object and convert this into the necessary seed points.

4.5.7.1 Building the Graph

Compared to the graph widely referred to in this thesis this method uses both different graph structure and a different cost function. We build a graph, such that each image pixel corresponds to one node, and edges connect all adjacent nodes in an 8-neighbourhood. The costs of the edges are computed using a weighted cost function based on contributions from

1. Distances between pixels, $dist(p, q)$
2. Differences in pixel intensities, $dist(C_p, C_q)$
3. Local orientation of the path, $Ori(\theta)$

The first contribution is set to 1 for edges connecting horisontal and vertical neighbours and $\sqrt{2}$ for edges between diagonal neighbours. This cost alone can be used to find shortest paths between control points.
The second contribution is based on differences in pixel intensities between neighbouring pixel. This term is used to penalise quick intensity changes, to make it more expensive to leave the street than to continue on the street.
The purpose of the third contribution is to penalise edge directions that do not conform with the local orientation. The computation of orientation takes into account the second-order information of the image to analyse the curvature. The second-

order information is represented in the Hessian matrix:

$$H(I) = \begin{pmatrix} I_{xx} & I_{xy} \\ I_{xy} & I_{yy} \end{pmatrix}, \tag{4.5.1}$$

where $I_{xx} = \frac{\partial^2 I}{\partial x^2}$.

The eigenvalues and eigenvectors of the Hessian describe the principal curvature and the direction of the principal curvature. A street in the image can be thought of as a ridge. Hence, for each ”street pixel” the local orientation is the direction perpendicular to the direction of the principal curvature - that is along the street.
The method uses the scale space framework to handle streets of different size. The image is convolved with second order gaussian derivatives of different standard deviations ($\sigma_0, \sigma_1, ..., \sigma_i$) before the directional information is extracted. The final scale selection is preceded by a scale normalisation by factor, σ_i.
The above analysis results in an indication of the local pixel orientation (θ) over scale. Finally, the orientation term of the cost function is computed by

$$Ori(\theta) = exp\left(\frac{-(\frac{\pi}{2} - \theta)^2}{\delta^2}\right) \tag{4.5.2}$$

where $\delta = \frac{1}{2}$.

The three terms are added in the weighted cost function

$$cost(p, q) = dist(p, q) + \alpha\, dist(C_p, C_q) + \beta\, Ori(\theta) \tag{4.5.3}$$

which is used to compute the edge costs of the graph.

4.5.7.2 User Input

In Figure 4.11 we show two examples of the tool operated with LS image data. The examples have been produced independently from our main program using the original implementation by the authors of [5].
The program finds the curvature of the objects quite well, and the centre of the found paths is inside the structures in their entire range. This indicates that the tool could save the operator from a lot of tedious work if it became a part of our graph cut implementation. The only thing missing is a way of converting the path into a dense row of seed points.

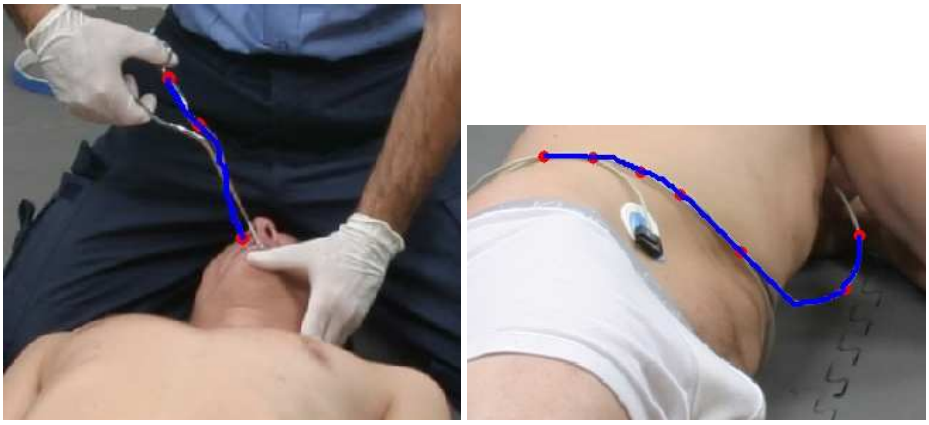


Figure 4.11: Two examples of the Interactive Route Measuring tool applied on LS image data. In both images the full range of the elongated structures is found by inputting only a few control points.

4.5.8 Background Subtraction

Included in the image material from LS are some images of the empty scene. These pictures can be thought of as background pictures. It is a standard routine for LS to take a picture of an empty scene i.e. we can always rely on having a background image when segmenting. Until now we have not made use of the background image, but we got the idea to use the background image to automatically input background seeds. The idea is to subtract the background image (I_{bg}) from the foreground image (I_{fg}) hereby leaving the object of interest and some noise due to camera variance and lack of registration. We should then be able to mask out the background and use this as input of background seeds. Doing this should have two positive effects:

- By using the background image to enter background seeds the user only has to enter foreground seeds.
- Using the background image we will have the ability to enter a very high amount of background seeds which should speed up the minimum cut algorithm because more nodes are labeled initially. Furthermore, it could have a positive effect on the segmentation quality.

To test the use of background subtraction we have made a small test application in Matlab to see if we should include the idea in the final C++ application. For the test we assume that I_{fg} and I_{bg} are properly registered.

We wish to find a binary mask $M(x, y)$ for which:

$$M(x,y) = \begin{cases} 1 & , |I_{fg}(x,y) - I_{bg}(x,y)| \geq t, \text{ foreground} \\ 0 & , |I_{fg}(x,y) - I_{bg}(x,y)| < t, \text{ background} \end{cases} \tag{4.5.4}$$

where t is a threshold that sorts out small differences in I_{fg} and I_{bg}. However, Equation 4.5.4 results in a noisy mask due to camera noise and registration errors. To reduce the noise we perform a morphological noise removal as follows:

$$M = (M \circ E) \bullet E \tag{4.5.5}$$

where E is the structuring element.

An example of background subtraction is shown in Figure 4.12, in which we use an image of size 1421x1455. The colour values are in the range of [0;255], t is set to 5 and E is of size 15x15. From Figure 4.12e we can see that the background subtraction works fairly well. In some areas, like the head in Figure 4.12f, we actually get a mask very close to the foreground object, which is good. However, the example also reveals some weaknesses of the method. Large parts of the mask are far from the foreground boundary even for areas clearly belonging to the background. There are two main reasons for this:

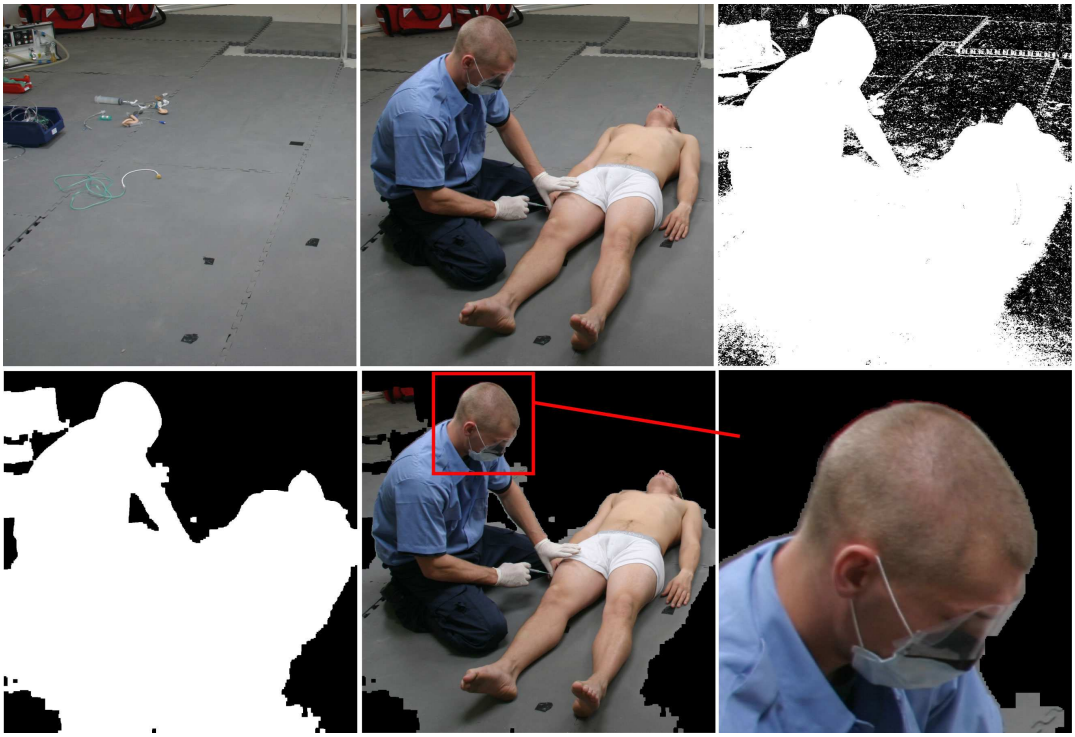


Figure 4.12: $\begin{smallmatrix} a & b & c \\ d & e & f \end{smallmatrix}$ Example of background subtraction. a) Background image I_{bg}. b) Foreground image I_{fg}. c) Result of subtracting I_{bg} from I_{fg} as in Equation 4.5.4. d) Result of morphological noise removal on c as in Equation 4.5.5. e) Result of applying d on b. f) Zoom in on man's head to show how precise the mask is.

- As can be seen from Figure 4.12a there are objects lying in the background which are not in the foreground image. These objects result in holes in the mask outside of the actual foreground object.
- The foreground objects throw shadows on the floor. These shadows have a large effect on the final mask. In the upper part of the mask in Figure 4.12e, where no shadows are present, the mask border is very close to the foreground object. In the lower part, where shadows fall on the floor, the mask border is far from the object.

One immediate positive result of the example is that non of the pixels marked as background intersect the foreground persons. This means that we can test the idea of using the difference image to input background seeds. However, since the desired foreground object is rarely alone in the scene, the user is still left with marking other objects as background.

4.5.8.1 Using the Background Image as Input

Figure 4.13a and 4.13b show the result using background subtraction for input of background seeds. The seeds found in Figure 4.12 were used together with the normal background seeds (marked with red) in Figure 4.13a to represent background seeds. For comparison, we have made a normal segmentation where all seeds are entered by the user, see Figure 4.13c and 4.13d. There are no significant difference in the quality between the two segmentations. Table 4.2 specifies the segmentation time in the two cases. Here, however, we see that the time spent on K-means clustering when using background subtraction by far surpass the time reduction for the graph search. Having this knowledge we decide not to include background subtraction in our further work.

Method	**K-Means**	**Graph Search**	**Total**
Bg. Subtraction	6.3 s	0.6 s	6.9 s
Normal	2.5 s	0.8 s	3.3 s

Table 4.2: Run-times for different parts of the segmentation process using two different approaches for input of seed points. *K-Means* show the time used for clustering the colour of seed points by K-Means, and *Graph Search* show the time used to run the Boykov-Kolmogorov augmenting path algorithm.

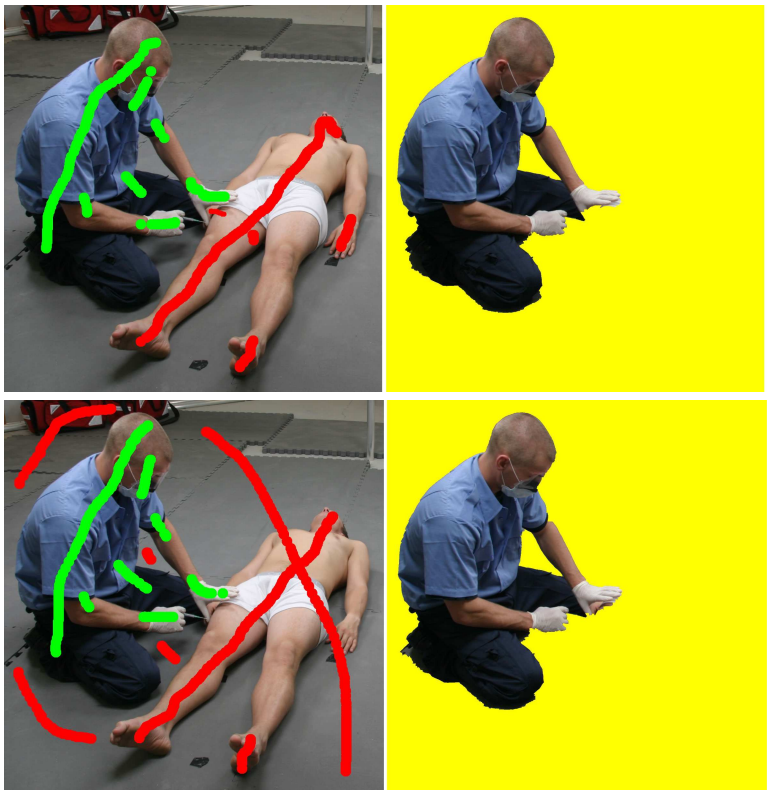


Figure 4.13: $\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}$ a) Seed points added in addition to the background seeds found in the example in Figure 4.12. b) Segmentation result using background subtraction. c) Input of seeds without the use of background subtraction. d) Segmentation result of c.

4.6 Fine Tuning by Adding Seeds

The resulting segmentation based on the initial seed points may not be satisfactory. Thus, it is relevant to provide a possibility of fine tuning the segmentation. The standard fine tuning methods in graph cut is to add additional seeds into the graph. Entering of new seeds is done just the same way as for the initial seeds. However, the subsequent procedure of how the seeds are treated and which updates are made to the graph is subject to discussion. The only source to our knowledge which provides details on how new seeds are added is Boykov & Jolly [1]. Their method becomes central to the following discussion of different ways of computing the min-cut after adding more seeds.

4.6.1 Recomputation of All t-links

Assume that we have a graph $\mathcal{G}$ for which we have found a min-cut/max-flow which results in an initial segmentation. The user now adds a new seed to correct the segmentation, and we need to compute a new min-cut/max-flow to obtain the updated segment. Intuitively, reuse of the previous flow is an attractive property because it would be much faster than recomputing the entire flow from scratch. Since new seeds change the basis for the colour statistics on which t-links are based, it would be obvious to recompute all t-links based on the updated sets of seeds $\mathcal{F}$ and $\mathcal{B}$. Conversely, the n-links of $\mathcal{G}$ should be kept intact because they are invariant to changes in the colour statistics.

This method, however, gives the problem that the revised residual capacities of some t-links can become negative because the original capacities of $\mathcal{G}$ are recomputed to values lower than the flow that they have already carried. This results in $\mathcal{G}_f$ containing negative edge capacities, which is an unsupported property for graphs in the framework of flow networks [4]. Figure 4.14 illustrates the concept of this problem. There is no trivial way of handling this incorrectness, and since

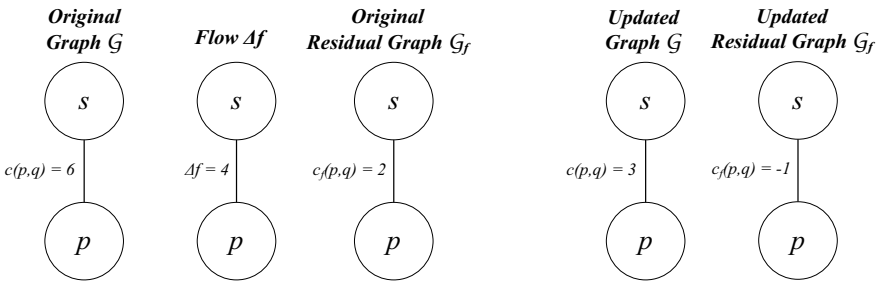


Figure 4.14: a b c d e a) The original capacity of the t-link between terminal s and pixel p. b) The flow Δf in edge $s - p$. c) The residual capacity, $c_f(s, p) = 6 - 4 = 2$ d) The updated graph based on new seeds. e) A negative residual capacity, $c_f(s, p) = 3 - 4 = -1$ is not allowed.

the optimisation problem has been changed, it can no longer be solved using min-cut/max-flow algorithms. Therefore, we need another approach for updating the graph.

4.6.2 Recomputation of t-links at New Seeds

Boykov & Jolly [1] present an updating method that reuses the initial flow and is claimed to still find the global minimum of the original problem after new seeds are added. They only change the t-link capacities of the new seeds, p, and the problem of negative flow is circumvented by adding a constant c_p to the two t-links at all $p \in \mathcal{F} \bigcup \mathcal{B}$. According to [1] this does not affect the optimal cut since a constant contribution cannot influence which one of the two t-links at p that is cut first.
However, the article [1] is not very clear about how c_p is derived, or how it is possible to find a consistent optimal solution to the original problem when changes to the graph are only made locally around new seeds. Moreover, the article is based on greyscale images and therefore uses different cost functions for both n-links and t-links.
We find it reasonable to believe that [1] uses a value for c_p so that the negative t-links from $p \in \mathcal{F} \bigcup \mathcal{B}$ become exactly zero.
Though having different settings from [1], we are inspired to propose a new fine tuning method that reuses the flow from the initial segmentation. The t-link capacities of the new seeds are modified, according to Table 4.3.

t-link	**capacity**	**for new seeds where**
$e(p,s)$	∞	$p \in \mathcal{F}$
	0	$p \in \mathcal{B}$
$e(p,t)$	0	$p \in \mathcal{F}$
	∞	$p \in \mathcal{B}$

Table 4.3: Modified terminal edge capacities for new seeds.

Since the colour of the new seed is not added to the global colour statistic, the new seeds will only effect local parts of the segment. The effect is that when we add a new foreground seed in an area that has been labeled B during the initial segmentation, we force-label the node, $p \in \mathcal{F}$ to F. Neighbour nodes $q \in \mathcal{N}_p$ of similar colour to p will have a high neighbour capacity $c(p,q)$. This combined with the infinite t-link capacity, $c(p,s) = \infty$ makes it likely for neighbouring nodes to also change label to F.
Using this method results in fast fine tuning, and we avoid the problem of negative edges in the graph. However, since new seeds do not influence regional colour

statistics through t-links the initially entered seeds become very important. Hence, it becomes very important to be thorough when entering seeds initially. We no longer solve for the global minimum of the original problem since we modify the flow between the first and the second run of the algorithm. But as we will show throughout the test in Chapter 7, this method produces good results for local fine tuning and is very fast.

4.6.3 Recomputation of All Edge Capacities

Alternatively, we could suggest a global recomputation of all edges in the graph, in which t-links are based on the updated sets of seeds $\mathcal{F}$ and $\mathcal{B}$. The min-cut would then be recomputed from scratch in the new graph.
Recomputing the t-link capacities for all nodes based on a new K-means clustering for both foreground and background would of course be much slower, than making local changes to the graph.
However, if we choose this approach it is important to reuse the previously computed cluster centres as starting points in the new K-means computations. This would eliminate the risk of random changes in the segmentation result, due to sudden changes in the colour clustering. Furthermore, the computation of updated centres will converge much faster, under the assumption that the existing seeds outnumber the new seeds.

4.7 Local Graphs

As mentioned, Lazy Snapping presents three fine tuning methods. The first is the traditional fine tuning method where the user enters additional seed points as described in Section 4.6. The last two are based on a curve representation of the segment border and are used to correct minor segmentation errors on the border. We have not included the curve representation in our implementation and can therefore not provide the last two fine tuning methods presented by [11].

In this section we will, however, present our own fine tuning method, based on building local graphs in the image. This method provides many of the same properties as the two omitted fine tuning methods from Lazy Snapping. Our method continues using a graph cut framework as the basis to obtain a pixel labeling. The difference from the graphs we have worked with until now is that we do not use the whole image to build a graph but only a local part of the image. Therefore, we refer to the method as a *local graph*. The local graph does not require a curve represented of the border, and since it is graph based, it is possible for us to reuse our graph cut implementation.

4.7.1 Building the Local Graph

The local graph used for fine tuning is not a stand-alone segmentation method, but requires that the image has been segmented e.g. using the global graph cut segmentation described in previous sections. The local graph is based on the user selecting an area on the segment border that has not been satisfyingly segmented. This selection is done using a mouse guided brush tool identical to the brush used to enter seed points in the global graph. The concept is shown in Figure 4.15. In

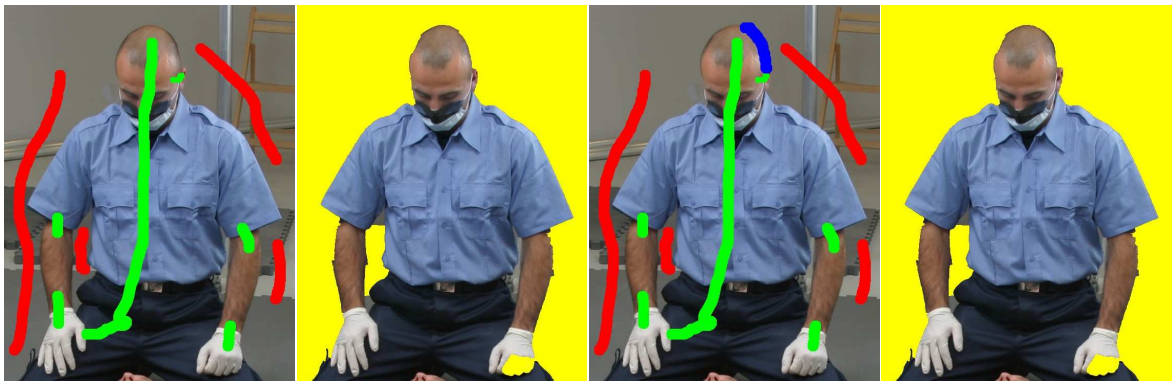


Figure 4.15: *a b c d* a) Seed points entered for the global segmentation. b) Result of global segmentation. c) Input of local graph marked in blue. d) Result of local segmentation combined with global segmentation

Figure 4.15a and 4.15b we have made a segmentation using the global graph cut

method. The result is, however, not satisfying on several parts of the border. Figure 4.15c shows a local graph, marked with blue, entered on the border of the man's head. This area is referred to as the unknown area U. The graph is build in the U alone. Contrary to the global graph, the local graph is build using 4-neighbourhood pixels as nodes instead of the watershed pre-segments. For the global segmentation pre-segmentation was done mainly to reduce the runtime. However, since the local graph is build in such a small area of the image, the runtime is no longer a problem. Eventually, the local segmentation overrides the global segmentation locally and results in Figure 4.15d.

4.7.2 Automatic Seed Point Selection

In order to make the use of the local graphs as easy as possible seed points are chosen automatically. We solve this by using the initial segmentation as guide for what is foreground and background. Therefore, the unknown area has to be placed on the border of the initial segment. This restriction enables us to find a row of pixels on each side of the unknown area to be foreground and background seeds respectivly. This is done using a sequence of morphological operations. Having a mask M from the initial segmentation and the unknown U selected by the user, the two sets of seeds $\mathcal{F}$ and $\mathcal{B}$ are obtained through the following two operations

$$\mathcal{F} = \{\, p \,|\, p \in (M - U) \;-\; [(M - U) \ominus E] \,\} \tag{4.7.1}$$

$$\mathcal{B} = \{\, p \,|\, p \in [(M \cup U) \oplus E] \;-\; (M \cup U) \,\} \tag{4.7.2}$$

where E is a 3x3 structuring element.

The example from Figure 4.15 is continued in Figure 4.16. Here we see how the rows of foreground and background pixels (Figure 4.16d and 4.16e) are obtained by applying Equation 4.7.1 and 4.7.2 to the initial mask M (Figure 4.16b) and the user selected unknown area U (4.16c). Finally, in Figure 4.16f we illustrate the combination of unknown area and seed points.

This way of entering seed points is very fast and fully automatic. Furthermore, it is ensured that the foreground and background seeds are selected inside and outside the initial segmentation. A drawback with this approach is, however, that the unknown area has to be selected along the border of the initial segment. It is therefore not possible to enter a local graph entirely inside or outside the the initial segment, which sometimes might be preferred.

4.7.3 Setting the Energy Functions

Using graph cut locally in the image has advantages related to both the t-link and n-link capacities. The colour information used to compute the t-link capacity is gathered from the local surroundings of the border we want to fine tune. Compared

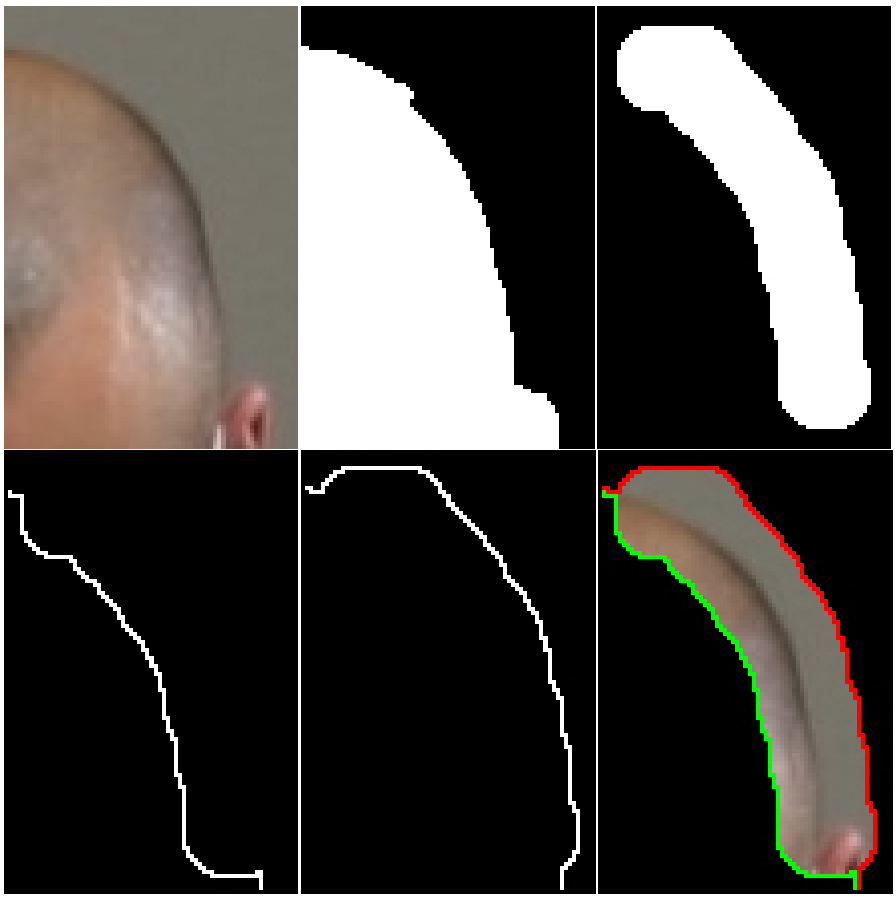


Figure 4.16: $\begin{smallmatrix} a & b & c \\ d & e & f \end{smallmatrix}$ a) Crop of original image used for the local graph. b) The mask M obtained from the global segmentation. c) Unknown area U selected by the user. d) Result of performing morphological sequence from Equation 4.7.1. Mask used for foreground seeds. e) Result of preforming morphological sequence from Equation 4.7.2. Mask used for background seeds. f) Illustration of how the local graph is build combined with seeds.

with the global graph cut this has the advantage that background objects with the same colours as foreground objects are less likely to occur. The colour statistics for the foreground and background seeds are therefore less likely to overlap.
A poor initial segmentation often occurs in areas of the image where there are low contrasts separating foreground and background. The function selected to set the n-link capacity for the global graph, described in Section 4.4.2, has been chosen to fit well with a global segmentation. However, for the local graph we have the possibility to choose a function more suitable for detecting low gradient boundaries. The function we propose is a slightly modified version of the function used for the global segmentation in Equation 4.4.4.

$$V_{p,q}(\cdot) = c(p,q) = \frac{1}{[\,||C(p) - C(q)||^2 + 1\,]^s} \tag{4.7.3}$$

In Equation 4.7.3 we simply add an exponent s to the denominator. Increasing s will cause the function to decent faster which causes lower gradients to result in even lower capacities for n-links in the graph. In Figure 4.17 we show the capacity $c(p, q)$ plotted as a function of the gradient for different values of s.

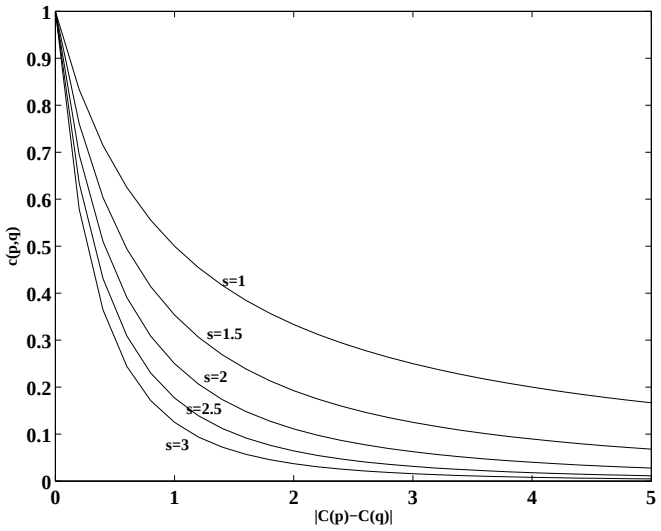


Figure 4.17: Illustration of varying the exponent s for the proposed neighbour weight function. Higher values of s result in faster falloff which accentuate lower gradients.

4.7.4 Splitting Up Pre-Segments

Besides the advantages described in relation to the local colour statistics and the neighbour weight function, one of the most important properties of the local graph is that it allows splitting up pre-segments down to single pixels. In order to pre-segment the image we filter the image using a gaussian edge detection filter. Doing this may cause low gradient edges to be blurred away which can result in pre-segments being created across edges. This can make it impossible to obtain the desired segmentation in the global segmentation step if the global segmentation is based on the pre-segmented image alone. However, having the local graph we have become able to correct these mistakes.
In Figure 4.18 and 4.19 we show three examples of how a poor pre-segmentation causes the global segmentation to fail. In all three examples we additionally show how a the local graph corrects the errors.

Another issue released to the pre-segmentation which can cause problems is the fact that the colour of a pre-segment is set to the average of the pixels it contains. The problem is illustrated in Figure 4.20. Figure 4.20a shows the original image and 4.20b shows the pre-segmentation of the original. If we focus on the darker

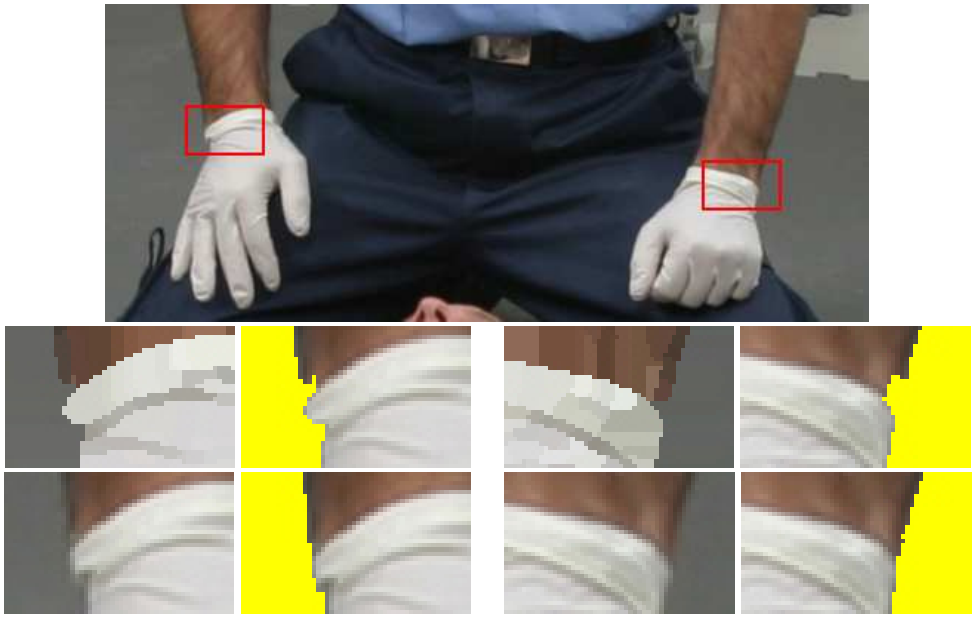


Figure 4.18: $\begin{smallmatrix} & a & \\ b & c & \\ d & e & \end{smallmatrix}$ $\begin{smallmatrix} f & g \\ h & i \end{smallmatrix}$ a) Original image showing the zoom in areas of the two examples. b) Pre-segmented image of the left zoom in. c) Segmentation result of global segmentation. d) Left zoom in of original image. e) Segmentation result using local graph. f) Pre-segmented image of the right zoom in. g) Segmentation result of global segmentation. h) Right zoom in of original image. i) Segmentation result using local graph.

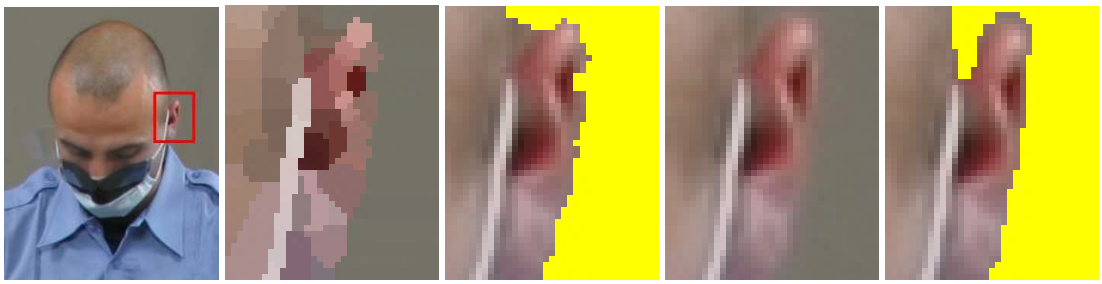


Figure 4.19: $a\ b\ c\ d\ e$ a) Showing zoom in area. b) Pre-segmented image. c) Result of global segmentation. d) Zoom in of original image. e) Segmentation result using local graph

area of hair on the border of the man's scull, it is clear that the transition to this area appears more smooth in the original than in the pre-segmented image. The problem is due to the averaging of pixel colours over pre-segments which results in a step wise transition causing a poor segmentation (Figure 4.20c). Using the local graph on the border of the head results in a much better segmentation shown in Figure 4.20d.

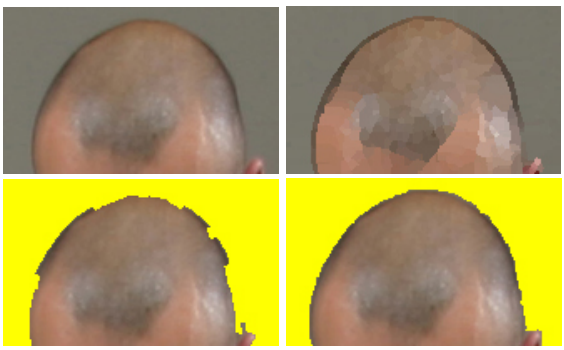


Figure 4.20: $\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}$ a) The original image. b) The pre-segmented image. c) Result of global segmentation. d) Result of segmentation using global graph.

4.7.5 Summary

The local graph has many good properties as a fine tuning tool. It uses local colour information which lowers the probability of overlapping colour statistics for foreground and background seeds. It allows detection of lower gradient edges. The graph is based on pixels which makes it possible to split up the pre-segmented image. Furthermore, because the local graph is build in a very small area of the image, it delivers instant user feedback which fits well into the interactive segmentation framework.
A drawback of the method in its present form is that it has to be entered on the border of the global segment. The similar fine tuning method presented in [11] does not have this restriction and therefore allow a higher level of user control. On the other hand, our implementation does not require any spline representation of the segmentation boundary and is therefore easer to implement. In addition, the graph cut implementation used for the global segmentation can be reused for the local graph.

When testing the performance of the final interactive segmentation method in Section 7, we will, however only include the local graph as a fine tuning tool to adjust the global segmentation.

Chapter 5

Alpha Estimation

In Chapter 1.2 we described how LS uses the alpha channel to soften the borders of their segments to produce a smooth transition from foreground to background when the segments are composited against new backgrounds. This smoothing has shown to be an extremely important part of the work related to image composition. Therefore, we have chosen to spend some effort on finding a softening method suitable for the task at LS.

As can be seen from the two new interactive segmentation methods for image editing [19] and [11] they also combine a hard segmentation[1] method with a method for softening the borders of the segments. In order to compare segmentation results with these state-of-the-art methods, we ourselves are interested in combining our segmentation method with a softening method.
The task boils down to estimating an alpha value for each pixel within some distance of the hard segment border found during the segmentation step.

In this chapter we will demonstrate three different methods for automatic estimation of alpha. In the first method we use a Gaussian low-pass filter to blur the alpha channel. The second method only operates locally on the border of the segments and uses a sigmoid function to estimate alpha values across the segment border.
The first two methods are computationally fast ways of getting an rough alpha estimate for simple borders. We refer to borders as simple if there are no fine structures like long straws of hair or transparent objects that let through colour from the background of the original image.

The third method, which we mentioned in Section 2.1.1.2 is a state-of-the-art alpha matting method based on a Bayesian probabilistic framework [3]. As previously mentioned matting techniques are known for their capability of solving highly complex alpha estimation tasks. Matting methods are, however, also known to be rather time-consuming.

[1] A hard segmentation method can be described by a binary mask with no degree of transparency.

5.1 Gaussian Transition

A very simple and computationally inexpensive way of obtaining a soft transition at the segment border is to apply a smoothing filter on the alpha channel. In Fig-

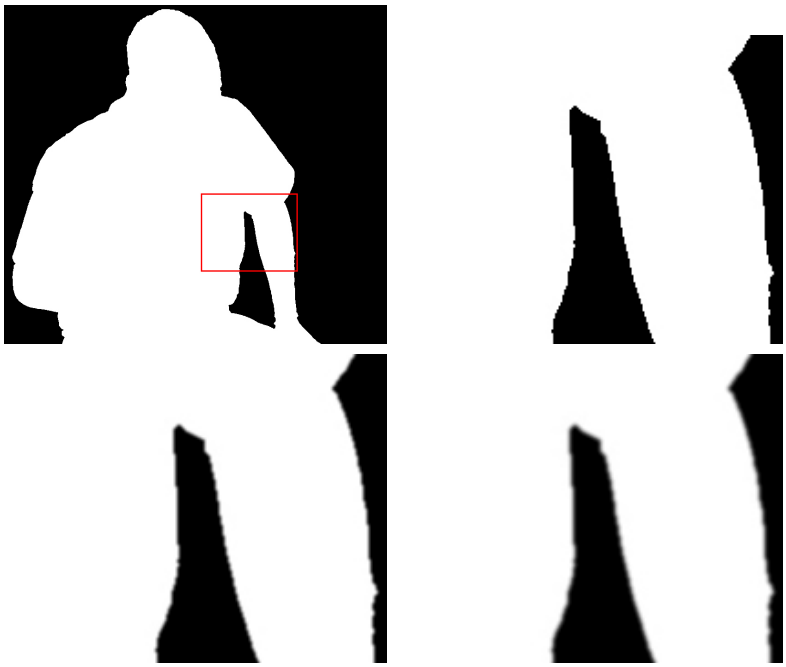


Figure 5.1: $\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}$ Shows the alpha before and after filtering. a) Image overview. b) A hard segment border. c) A softened segment border obtained by applying a Gaussian 3x3 kernel to the hard segment. d) Softened using a Gaussian 5x5 kernel.

ure 5.1c and 5.1d are seen the results of applying a 3x3 and a 5x5 Gaussian filter, respectively, to soften the alpha channel. Figure 5.2 shows the result of combining the original image with the hard and the two softened alpha channels. A comparison of the three resulting images shows that the segment border in 5.2c and 5.2d has become softer. It is also seen that the Gaussian filtering smoothes small irregularity along the border. This is a visually good effect and the segment becomes more ready for compositing against a new background.

Despite the fact that it is a rather simple method, the Gaussian filtering solves the task convincingly well for a simple border as the one in Figure 5.1 and 5.2. Since the majority of the actors used in LS's simulators are men with very short hair, it would be reasonable to assume that most segments have simple borders.
An important note to make is that the Gaussian filtering only takes a few seconds, which is very suitable for our interactive context. Another advantage is that it is quite trivial to implement the filter, and that it is already a part of our platform Picture Factory.
However, there are some problems with this global approach to the alpha channel.

Figure 5.2: $\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}$ Shows how the original image has been softened by combining it with the filtered alpha channel. a) Image overview. b) A hard segment border. c) A softened segment border obtained by applying a Gaussian 3x3 kernel to the hard segment. d) Softened using a Gaussian 5x5 kernel.

Firstly, the size of Gauss that produces an excellent transition in one part of the segment may be too big and ruin an otherwise good segmentation elsewhere. Secondly, the removal of irregularities along the border also may get out of hand, i.e. we risk to remove small but important image structures.
For these reasons we are interested in an adaptive approach that takes local conditions into account.

5.2 Local Alpha Estimation

In this section we demonstrate the concept of a fairly simple local approach to the alpha estimation problem. We only process pixels close to the border of the segment and use a sigmoid function to model the transition.

The alpha value at each pixel is computed as a function of the distance to the segment border. All pixels are processed in one sweep in which both distance and alpha value are computed.

The computation of distances is preceded by a few morphological operations. Based on the segmentation mask M and the 3x3 structuring element E we find

the two rows of pixels located on each side of the border. The pixels on the inside are found by

$$Border_{in} = M - (M \ominus E). \tag{5.2.1}$$

And the outside pixels are found by

$$Border_{out} = (M \oplus E) - M \tag{5.2.2}$$

For all pixels $p \in Border_{out} \cup Border_{in}$ the distance to the border is 0.5 pixels. Repeatedly, new rows of pixels are found using a breadth-first search scheme on an 8-neighbourhood. For every new row the distance value is increased by 1. Figure 5.3 illustrates this concept. When the distance to the border is found, we can

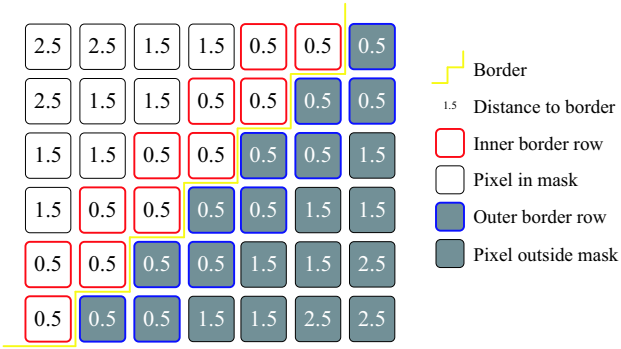


Figure 5.3: An illustration of the hard border and the distances to the border. The rows of pixels D_{in} and D_{out} are located respectively inside and outside the segment.

compute the alpha value. We have modeled the border transition using a sigmoid function oriented perpendicularly to the border

$$f(x) = \frac{1}{1 + e^{(x_0 - x)/\sigma}}, \tag{5.2.3}$$

where x is the distance from the border and σ is used to vary the steepness of the transition. The sigmoid is centered around the border and yields values in the range $0 \leq f(x) \leq 1$, which is the range we need for alpha estimation. As opposed to the Gaussian function the sigmoid is symmetric and has the the same falloff on both sides of the border. This is important because it results in identical blurring of foreground and background pixels. See plots of the sigmoid in Figure 5.4. Figure 5.5 shows an example of the sigmoid used to soften the border of a segment.
The border is rather simple because the person has very short hair, but the result shown in Figure 5.5c is not quite satisfactory. The blurring of the border creates a disturbing jagged pattern.
The second example in Figure 5.6a shows a crop of some hair on a woman's head. The estimated alpha channel in Figure 5.6b and the result in Figure 5.6c also show the jagged pattern in a yet worse edition.

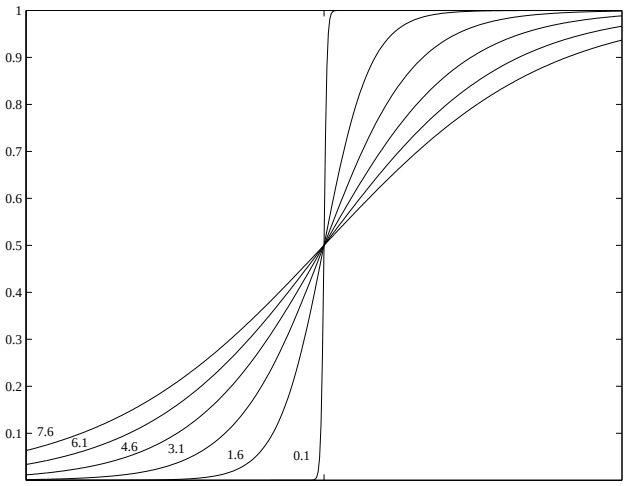


Figure 5.4: Plots of the sigmoid function from Equation 5.2.3 for 6 different values of σ.

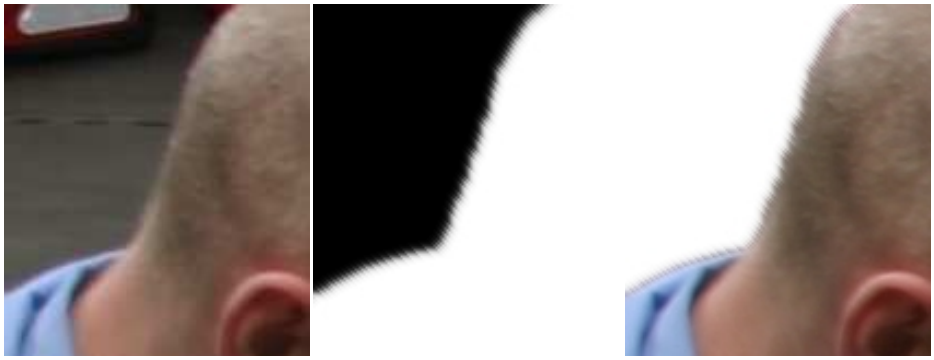


Figure 5.5: $_{a\ b\ c}$ a) 140x161 crop from original LS image. b) Locally sigmoid estimated alpha channel using $\sigma = 0.5$. c) Resulting segment after alpha blending.

One of the potentials of the local approach is that we can let the local amount of softening depend on the local image gradient perpendicular to the border. This would probably help the problem related to the Gaussian filtering method that some parts of the border become too blurred.
However, the jagged patterns in the two examples are enough to discard the method in its current state. We believe that the patterns stem from the oversimplified distance measure obtained from the breadth-first search. Hence, we expect that a more precise distance measure would improve the results significantly. We choose to leave this problem for further work.
In this section and the previous section we have shown that simple methods for alpha estimation can be used if the border we are trying to model is simple. However, in order to estimate a useful alpha channel for more complex borders i.e. borders

Figure 5.6: $_{a\ b\ c}$ a) 140x161 crop from original LS image. b) Sigmoid estimated alpha channel $\sigma = 2.5$. c) Resulting segment after alpha blending.

containing hair straws we need an actual matting method.

5.3 Alpha Matting Using a Bayesian Framework

In this section we will demonstrate the capabilities of the Bayesian matting [3] algorithm to model complex border transitions. The choice of matting method fell upon Chuang et al. [3] because it is the foundation for many of the newer matting methods and therefore is a well tested technology. Another reason is that many of the newer matting methods are poorly documented with respect to implementation details.
First we will go into a detailed theoretical description of alpha matting in a Bayesian framework. Subsequently, show some we will results of applying the matting method to difficult segment borders.

5.3.1 Method Overview

In addition the the image the input for a matting method is a trimap that specifies an unknown area together with known foreground and background areas. For all pixels in the unknown area the goal is to estimate an opacity value, α, that blends between the foreground object and any new background that the object is placed into.
Samples from the known foreground and background along with already estimated pixels are gathered in the neighbourhood of the current unknown pixel. These samples are divided into clusters which form the statistical foundation to estimate the alpha value in a Bayesian framework.

5.3.2 Obtaining the Trimap

If alpha matting is used as a stand-alone application, the trimap is usually entered by the user. However, when using alpha matting to soften the border of a hard segmentation, the most common approach, used both by [11] and [19], is to dilate the border of the hard segmentation with a 4-6 pixels wide structuring element. The dilated border is the unknown area U. Knowing the mask M for the hard segmentation we are able to obtain the foreground area F and the background area B:

$$U = M \oplus E, \tag{5.3.1}$$

$$F = M - U, \tag{5.3.2}$$

$$B = M^c - U, \tag{5.3.3}$$

where E is a structuring element for which the size determines the width of U.

5.3.3 Compute Processing Order

Estimating the alpha value is done for all pixels in the unknown area one at a time. We will refer to the pixel in processing as the current pixel. When estimating the alpha value for unknown pixels, we rely on known foreground and background pixels

but also on already estimated surrounding pixels. These pixels are often closer to the current pixel and are therefore more likely to present a good representation for the current pixel. However, if an error was made in the computation of a previous pixel, using this pixel in the estimation of the current pixel could cause the error to accumulate. In order to minimising errors in accumulating we start the algorithm by processing the pixels closest to the known foreground and background areas and work our way towards the centre of the unknown area. In practice this means that we start by processing the pixels with the shortest distance to the nearest known foreground and background area.
To determine the distance to the nearest foreground and background pixel we use a breadth first growing technique and grow from the border of both the known F and B. This yields an integer distance to the foreground and background border which is not a precise distance measure, but it is sufficient to ensure a correct processing order. The computed distances are saved in two distance maps D_F and D_B representing the distances to the nearest known foreground pixel and known background pixel respectively. D_F and D_B will be used in the next step of the matting method.

5.3.4 Pixel Sampling

To model the distribution of foreground and background colours we perform a pixel sampling in the surrounding area of the current pixel. The area in which we sample should cover both known foreground and background pixels, and pixels close to the current pixel for which an alpha value is already estimated. There are, however, different approaches to this sampling. Chuang et al. [3] sample in a circle within a radius of the current pixel as shown in Figure 5.7a. The radius of the circle should be large enough to cover both known foreground and background pixels which therefore will include a very high number of samples. Karlsson [8], however, states that the number of samples have a drastic effect on the run-time of the matting algorithm. He therefore suggests a different approach where sampling is done in the surroundings of the nearest foreground and background pixel and the current pixel, (see Figure 5.7b). Since our aim is to provide an interactive segmentation tool, and [8] shows that his sampling approach is much faster than the approach of [3], we decide to use the sampling approach shown in Figure 5.7b.

To find the nearest known foreground and background pixel we use the previously estimated distance maps D_F and D_B. Using the distance maps as two gradient images we can ”slide down hill” to the newest foreground and background pixel much like the toboggan watershed shed algorithm explained in Section 4.2.1.3.

5.3.5 K-Means Clustering

From the sampling step we now have a set of foreground pixels and a set of background pixels. The two sets are partitioned into clusters using K-means clustering. Clustering is done to reduce the amount of data and to reduce noise in the sampling

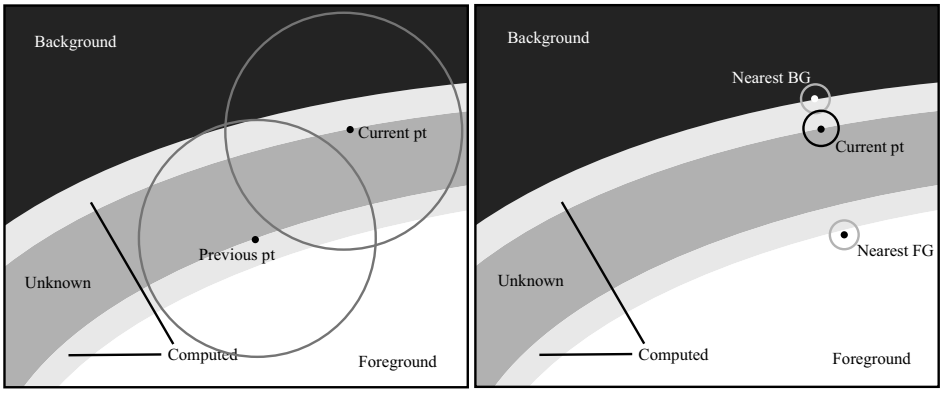


Figure 5.7: $_{a\ b}$ a) Sampling approach suggested in Bayesian matting [3]. b) Sampling approach suggested by Karlsson [8].

sets. Inspired by [8] we use a slightly changed version of K-means. This version is characterized by dynamically choosing the number of clusters, K. This is done by increasing K until all samples are closer to their cluster centre than some threshold T. The concept is shown in the algorithm below:

```
Dynamic K selection for K-means clustering
Set K to 0
while max(dist(data point, cluster centre))>T
   K=K+1
   Choose the point for which max(dist(data point, cluster centre)
   perform normal K-means
end while
```

5.3.6 Bayesian Matting

Having a foreground area F and a background area B the goal of of Bayesian matting is to find the most likely estimates for a foreground colour $\vec{F}$, a background colour $\vec{B}$ and an α-value given an observed colour $\vec{C}$ for all pixels in U. As the name of the method indicates, this is done in a Bayesian framework using the maximum a posterior (MAP) technique to solve for the most likely $\vec{F}$, $\vec{B}$ and α. The probability P is described by Bayes's theorem as follows:

$$\arg\max_{\vec{F},\vec{B},\alpha} P(\vec{F},\vec{B},\alpha|\vec{C}) = \arg\max_{\vec{F},\vec{B},\alpha} \frac{P(\vec{C}|\vec{F},\vec{B},\alpha)P(\vec{F})P(\vec{B})P(\alpha)}{P(\vec{C})} \quad (5.3.4)$$

This expression can be split up into sums of log likelihoods $L(\cdot)$:

$$\arg\max_{\vec{F},\vec{B},\alpha} P(\vec{F},\vec{B},\alpha|\vec{C}) = \arg\max_{\vec{F},\vec{B},\alpha} L(\vec{C}|\vec{F},\vec{B},\alpha) + L(\vec{F}) + L(\vec{B}) + L(\alpha), \tag{5.3.5}$$

where $P(\vec{C})$ is a normalisation term that is constant with respect to $\vec{F}$, $\vec{B}$ and α and can therefore be left out.

5.3.6.1 Defining the Log-Likelihoods

The problem is now to define the log-likelihoods $L(\vec{C}|\vec{F},\vec{B},\alpha)$, $L(\vec{F})$, $L(\vec{B})$ and $L(\alpha)$.
The first term models the log-likelihood of $\vec{C}$ given the optimization parameters $\vec{F}$, $\vec{B}$ and α. This term models the error measurement between the observed colour $\vec{C}$ and the estimated $\vec{F}$, $\vec{B}$ and α. It can be interpreted as a measure of how well they fit each other. $L(\vec{C}|\vec{F},\vec{B},\alpha)$ is defined as:

$$L(\vec{C}|\vec{F},\vec{B},\alpha) = \frac{-||\vec{C} - \alpha\vec{F} - (1-\alpha)\vec{B}||^2}{\sigma_C^2}, \tag{5.3.6}$$

where σ_C is the camera variance or camera noise. This camera variance is modeled by a Gaussian probability distribution centered at $\bar{C} = \alpha\vec{F} + (1-\alpha)\vec{B}$. This distribution is illustrated as the centre circle in Figure 5.8.

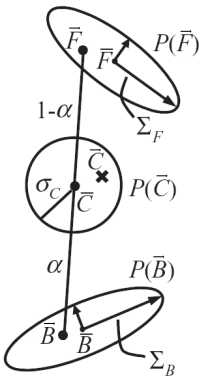


Figure 5.8:

Modeling the distributions of the foreground and background colours $L(\vec{F})$ and $L(\vec{B})$ is done using the clustered samples from known and estimated foreground and background pixels.
To make a more robust model each pixel in the foreground clusters is applied a weight w_i. The weight is a product of two factors. Firstly, foreground samples are

weighted by α_i^2 to give more confidence to pixels with high alpha values. Background samples a similarly weighted with $(1-\alpha_i)^2$. Secondly, all samples are weighted by their distance to the current pixel x_i giving more weight to pixels close to x_i. The distance weight is modeled using a Gaussian falloff centered at x_i. The combined weight function for foreground samples are given as

$$w_i = \frac{\alpha_i^2 e^{\frac{-(\vec{x}_i-\vec{\mu})^2}{2\sigma^2}}}{\sigma\sqrt{2\pi}}. \tag{5.3.7}$$

As Chuang et al. [3] we use σ =8 in the weight function.

Having a set of foreground clusters with weighted pixels the next step is to calculate the weighted mean colour $\bar{F}$ and the weighted covariance matrix $\Sigma_{\mathbb{F}}$ for each cluster. This is done as follows:

$$\bar{F} = \frac{\sum_{i\in N} w_i \vec{F}_i}{\sum_{i\in N} w_i}, \tag{5.3.8}$$

$$\Sigma_{\mathbb{F}} = \frac{\sum_{i\in N} w_i (\vec{F}_i - \bar{F})(\vec{F}_i - \bar{F})^T}{\sum_{i\in N} w_i}, \tag{5.3.9}$$

where N is the set of all pixels in a cluster. Similar formulas are given for $\bar{B}$ and $\Sigma_{\mathbb{B}}$.

Using the weighted mean $\bar{F}$ and covariance matrix $\Sigma_{\mathbb{F}}$ the log-likelihood $L(\vec{F})$ can be models as an oriented Gaussian distribution. This is illustrated in Figure 5.8 where the upper ellipse is the foreground distribution and the lower ellipse is the background distribution. $\bar{F}$ is the centre of the ellipse and $\Sigma_{\mathbb{F}}$ is the largest radius of the ellipse. The general formula for $L(\vec{F})$ and $L(\vec{B})$ is given as:

$$L(\vec{X}) = \frac{-(\vec{X}-\bar{X})\Sigma_{\mathbb{X}}^{-1}(\vec{X}-\bar{X})}{2} \tag{5.3.10}$$

Chuang et al. [3] have made an important correction to their article related to the camera noise σ_C. The correction can be viewed from the article homepage but we will describe it here below.
As mentioned earlier an amount of camera noise σ_C is added to the observation colour $\vec{C}$. Because foreground and background samples are observations from the same camera, the camera noise should be added to these observations. This is done by decomposing the covariance matrix for each cluster using a singular value decomposition (SVD) and adding the noise σ_C^2 to each of the three axes. The approach is shown in the following equation.

$$\Sigma = U\cdot D\cdot V^T \ , \ D' = \begin{bmatrix} d_1+\sigma_C^2 & 0 & 0 \\ 0 & d_2+\sigma_C^2 & 0 \\ 0 & 0 & d_3+\sigma_C^2 \end{bmatrix} \ , \ \Sigma = U\cdot D'\cdot V^T \tag{5.3.11}$$

Doing this has another effect. The covariance matrix for a cluster is constructed from pixels which are often similar or even identical. When pixels are identical the covariance matrix becomes singular which makes the calculation of the inverse covariance matrix Σ^{-1} impossible. However, by adding noise to the matrix as in Equation 5.3.11, the number of singular matrices is reduced.

Chuang et al. [3] assume the log-likelihood $L(\alpha)$ to be constant. This term is therefore omitted in the maximization in Equation 5.3.5.

5.3.6.2 Maximizing the Log-Likelihoods

Having defined each of the log-likelihoods we now go on to estimate the parameters for $\vec{F}$, $\vec{B}$ and α which maximizes Equation 5.3.5. $\vec{F}$ and $\vec{B}$ are 3×1 vectors, which gives a total of seven unknowns to determine. Since we only have three equations, the problem is decomposed into two sub-problems. First, we maintain a constant α value and solve for $\vec{F}$ and $\vec{B}$ in the equation below (from [3]).

$$\begin{bmatrix} \Sigma_{\mathbb{F}}^{-1} + \frac{\mathbb{I}\alpha^2}{\sigma_C^2} & \frac{\mathbb{I}\alpha(1-\alpha)}{\sigma_C^2} \\ \frac{\mathbb{I}\alpha(1-\alpha)}{\sigma_C^2} & \Sigma_{\mathbb{B}}^{-1} + \frac{\mathbb{I}(1-\alpha)^2}{\sigma_C^2} \end{bmatrix} \begin{bmatrix} \vec{F} \\ \vec{B} \end{bmatrix} = \begin{bmatrix} \Sigma_{\mathbb{F}}^{-1}\bar{F} + \frac{\vec{C}\alpha}{\sigma_C^2} \\ \Sigma_{\mathbb{B}}^{-1}\bar{B} + \frac{\vec{C}(1-\alpha)}{\sigma_C^2} \end{bmatrix} \tag{5.3.12}$$

I is a 3×3 identity matrix. The matrixes are of size 6×6, 6×1 and 6×1 respectively. Having found $\vec{F}$ and $\vec{B}$, α is found as a projection of $\vec{C}$ onto the line $\vec{F} - \vec{B}$.

$$\alpha = \frac{(\vec{C} - \vec{B})(\vec{F} - \vec{B})}{||\vec{F} - \vec{B}||^2} \tag{5.3.13}$$

By alternating the two above equations, using the result of the previous iteration, $\vec{F}$, $\vec{B}$ and α is found when α converges. Initially, α is set to the average of its eight neighbouring pixels.
Foreground and background pixels are divided into clusters and for each cluster there is a mean and covariance matrix associated with every cluster. To find the pair of foreground and background clusters that yields the highest maximum of Equation 5.3.5, the above optimization is made for all combinations of foreground and background clusters. The optimal $\vec{F}$, $\vec{B}$ and α values are chosen from the combination of clusters which results in the highest likelihood.

5.3.7 The Bayesian Matting Algorithm

The previous sections described the general approach of Bayesian matting. The method, however, contains a large number of steps. To ease the understanding of each of the order of the these steps we provide the Bayesian alpha matting algorithm below.

Bayesian Matting Algorithm

```
Create distance map
Compute processing order
for all unknown pixels
   Acquire foreground and background samples
   Compute weight w_i for all samples
   Cluster foreground and background samples using K-Means
   for all foreground and background clusters
      Compute mean colour
      Compute covariance matrix
   end for
   for each pair of foreground and background cluster
      repeat
         Find best foreground and background color given α
         Find best α given foreground and background color
      until α converges
      Compute current likelihood=L(C|F,B,α)+L(F)+L(B) of
         converged parameters for cluster pair
      if current likelihood > previous best likelihood then
         Set current likelihood to best likelihood
      end if
   end for
end for
```

5.3.8 Alpha Matting Examples

After going through the theory related to the Bayesian matting technique we now show some examples of the method applied to a few images. Subsequently, we will address some pros and cons of our matting implementation compared to the more simple softening methods.

The first problem we encountered when testing Bayesian matting was the run-time of the algorithm. Table 5.1 shows the run-times of our implementation on 3 different images. In the column *Unknown size* are found the number of pixels in the unknown area for each example. The immediate impression is that the time-per-pixel ration is very high. Studying the Bayesian matting algorithm we suspected

Image	Image Size	Unknown Size	Time Used
Man's scull	130x83	1289	14.0 s
Woman's hair	76x200	8829	127.7 s
Author's eye	98x130	2244	58.4 s
Uncropped Image	≈1400x1000	≈70000	-

Table 5.1: The run-times for the 3 example images are listed along with the image sizes in pixels. *Unknown size* is the number of pixels in the unknown area.

the method to be slow but were surprised of these run-times. However, since our project period were rather advanced by the time we encountered this problem we have not been able to optimise our alpha matting implementation to an extend that reduced the run-times notably. It occurs to us that our implementation of the described algorithm is not optimal, since the our processing times by far surpass those stated in [8].

Having a slow implementation has forced us to test the alpha matting on very small images as shown in Table 5.1. To save time we drew the trimaps manually to get small unknown areas centered around the most interesting image details. This reduced the size of the unknown area which meant that we managed to focus our effort on the most difficult parts of the borders.

For the larger non-cropped image (listed in Table 5.1) we obtained the trimap by dilation as described in Section 5.3.2. The unknown area contained approximately 70.000 pixels in a 13 pixels wide band around the entire segment. Due to the long processing time we have not been able to determine useful programme parameters to provide a meaningful matting example for images of this size.

The first example seen in Figure 5.9 shows matting applied to a simple border. The cropped image is the scull of one of LS's short haired actors. In Figure 5.9b

we have specified a trimap in which the dark grey area indicates the unknown area of pixels to be processed. Figure 5.9c shows the matte derived from the original image and Figure 5.9d shows the resulting segment on a new background. Apparently,

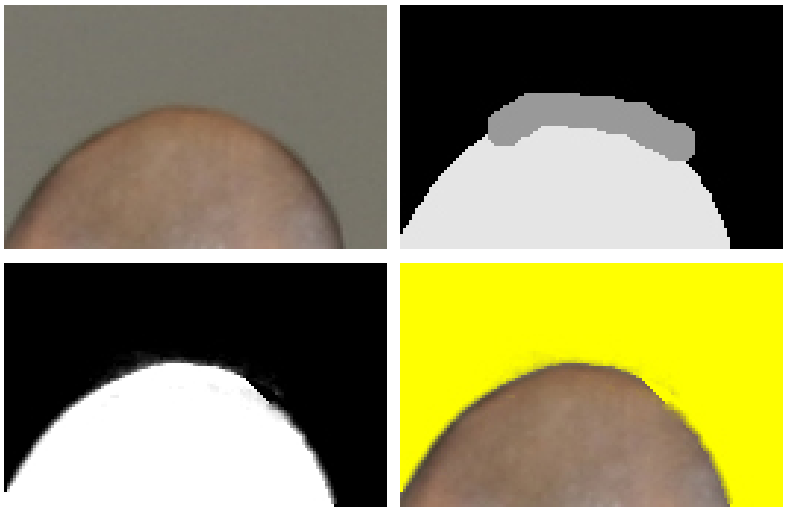


Figure 5.9: $\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}$ a) Original image cropped to size 130x83. b) User specified trimap. The unknown area (marked with dark grey) contains 1289 pixels to be processed. Foreground pixels are light grey and background pixels are black. c) Matte derived from the original image. d) The resulting segment on a new back background.

the result is less good than those previously obtained using the Gaussian transition. From the matte we see that the method has caused an exaggerated blurring within the entire unknown area. The amount of blurring obtained on the actual border varies a lot. In the right part of the processed area there is a hole where the border has not been softened at all. It seems like the error arises at the border of the unknown area and propagates from there.

The second image example in Figure 5.10a is cropped from the head of an LS female actress. The crop mainly contains straws of hair which is known to represent a problem to the simple methods and for segmentation methods in general. A look at the derived matte in 5.10c should convince the reader that the matting method is capable of something that the simple methods cannot present. We argue that the matting to some extend has been successful because the individual straws of hair are easily distinguished. There are, however, some problems with this estimation. For instance in Figure 5.10d the yellow colour of the background shines very intensely through the foreground in the upper part. This happens because the algorithm cannot differ the colour of the original background from the elongated highlight in the hair. In general, a more successful matte should probably have fewer transparent pixels in the dense area close the foreground.

The third example is a 98x130 crop of the eye of one of the authors shown in

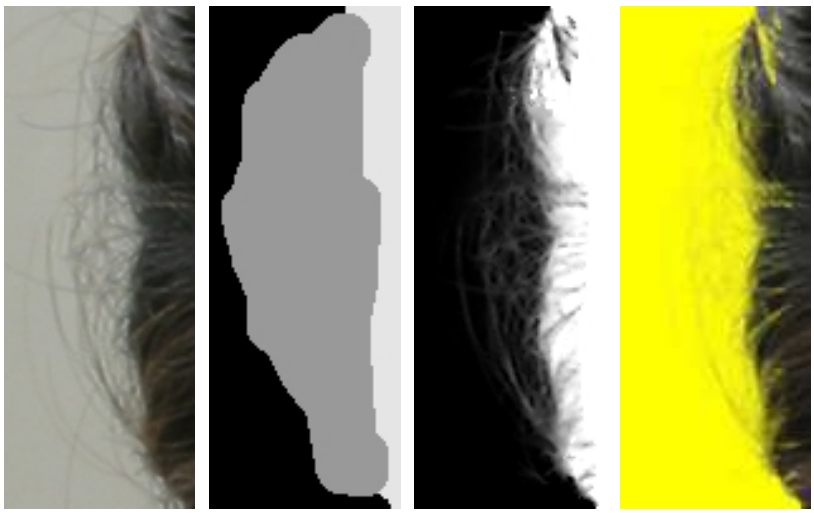


Figure 5.10: $a\ b\ c\ d$ a) Original image cropped to size 76x200. b) User specified trimap. The unknown area (marked with dark grey) contains 8829 pixels to be processed. Foreground pixels are light grey and background pixels are black. c) Matte derived from the original image. d) The resulting segment on a new back background.

5.11a. In this image the most challenging feature should be the eyelashes. These are estimated rather well as seen in the matte in Figure 5.11. Still, the ends of the individual straws of hair are blurred away. For the area around the eye the problem is the same - the matte is too transparent.

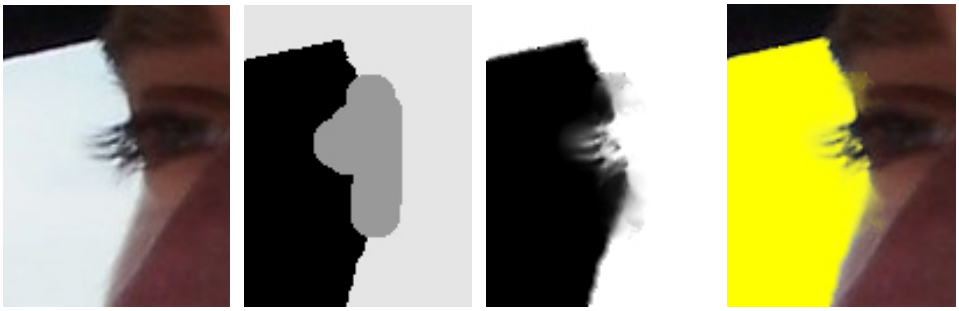


Figure 5.11: $a\ b\ c\ d$ a) Original image cropped to size 98x130. b) User specified trimap. The unknown area (marked with dark grey) contains 2244 pixels to be processed. Foreground pixels are light grey and background pixels are black. c) Matte derived from the original image. d) The resulting segment on a new back background.

Our general impression of the Bayesian matting algorithm is that is it has a great potential when it comes to estimating alpha transitions for complex segment borders. However, even though we have shown some of our best results none of the presented matting examples are fully satisfactory. A possible cause of the problems we have encountered is that we have not been able to find an optimal combination

of the many program parameters for the algorithm.

Another concern is the time spend to process even relatively small images. Based on our observations in this section we conclude that our present matting implementation does not match the demand for interactivity and therefore cannot be a part of our prototype application.

Chapter 6

Implementation Details

In this chapter we will go through some details of our program implementation. First, we explain our choice of C++ as main implementation language. Second, we will deal with the program structure and explain important decisions regarding the design. Finally, we will discuss the relevance of the implemented graphical user interface.
The source code can be viewed from the CD-Rom that is attached to this thesis. Because of licence restrictions we have not included a version of the running application on the CD-Rom.

6.1 Choice of Programming Language

All implementation for this thesis is done in C++. This is primarily because it has been the aim to make an implementation that is an integrated parts of LS's C++ based Picture Factory.
The use of C++ allows us to use fast pointer operations and avoid massive reallocation of big memory block when we pass large images back and forth between different part of the program.
With C++ we can also benefit from intuitive object oriented design possibilities. This means that the program code can be broken down into semantically meaningful objects. The implementation is then accomplished by composing the objects into a final structure.
Finally, the choice of C++ has allowed us to use well-tested third-party packages such as CVM and IPP from Intel.

6.2 Program Structure

This section provides a brief description of the main components of our implementation. To help the understanding a diagram of program structure is shown in Figure 6.1. In the figure ”Main Method” refers to the graphical user interface of the Picture Factory framework.

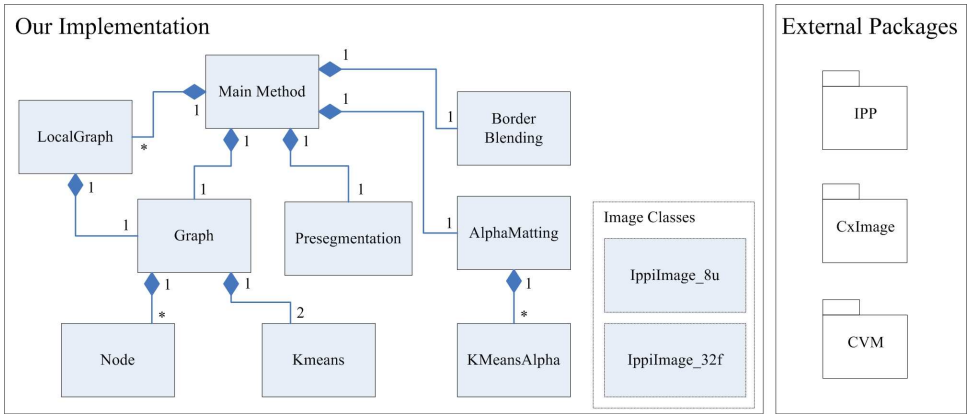


Figure 6.1: Simplified UML diagram showing the overall structure of the implemented program.

Graph The `Graph` class plays a leading role in our program structure. The component contains references to all nodes, and it has methods to handle seed points and run the graph cut algorithm. The `Graph` class is constructed based on a pre-segmented image. It creates all nodes in the graph and build their internal neighbourhood relations.

Node As the name indicates this class is used to represent the node entities in the graph. Because a `Node` represents a pre-segment, it contains references to the pixels it embodies. The node computes, and stores their average colour of the segment. Moreover, it has a knowledge of neighbouring nodes and it can compute edge capacities for connecting edges. In the relation to the Boykov-Kolmogorov algorithm it has knowledge of which of the two search trees s or t it belongs to and whether it is an active node.

KMeans Receiving a list of $\mathcal{F}$ or $\mathcal{B}$ seeds this class computes and stores the K-means cluster centres used to compute the t-link capacities.

Presegment The `Presegment` class performs the toboggan watershed segmentation based on a given input image.

LocalGraph The `LocalGraph` class receives a pointer to the original image, the initially segmented mask and a list of pixels that describes the area of which the local graph is build. `LocalGraph` has an instance of the `Graph` class and all pixels in the input list are created as nodes in the graph. By using the mask `LocalGraph` is automatically defines seeds for the graph and finds the min cut.

BorderBlending The class contains methods used to create a simple softening of segment borders.

AlphaMatting The `AlphaMatting` class contains advanced methods used to estimate an alphamap for segments with complex borders.

KMeansAlpha The `KMeansAlpha` class does the clustering of the colour samples from `AlphaMatting`.

IppiImage_32f & IppiImage_8u Our original idea was to implement everything using the image class in Picture Factory provided by LS. However, the image class related to Picture Factory, `CxImage` stores pixels as `char` values with no possibilities of floating point precision. We need floating point precision to, among other things, work correctly with fourier transformed images. Therefore, we need a different image representation. Early in the process we decided to use the image library IPPI, which is part of Intel Performance Primitives (IPP).

IPPI contains a lot of image processing functionalities. These are, however, implemented in a low-level structure to optimise performance. In order for us to get easy access to image data, we have implemented two almost identical wrapper classes `IppiImage_32f` and `IppiImage_8u`, which are based on IPPI and have access to the functionalities of IPPI. As the name indicates the class `IppiImage_32f` is constructed to contain 32 bit floating point pixel data. `IppiImage_8u` is used to store unsigned 8 bit i.e. ranging from 0 to 255. To facilitate interaction with Picture Factory we have implemented methods in the IppiImage classes that convert back and forth between `IppiImage_32f/8u` and `CxImage`.

6.2.1 Filtering in the Fourier Domain

In Section 4.2.1 we performed a filtering $I_b * \frac{\partial N_\sigma}{\partial x}$ in the making of the gradient magnitude image. We have chosen to filter in the frequency domain because LS have very large images (3072x2048), and we want the process to be as fast as possible.
The size of the used filter also influences our choice of domain. Implementing the filtering in a multi-scale framework we may want to use larger scales. For instance a value of $\sigma = 32$ would give a filter of size 288x288 if we wanted to represent $\pm 3\sigma$.
Convoluting one of LS's colour images with a filter of the above size in the spatial domain would be very demanding in terms of computational time. Therefore, we choose to do the filtering in the frequency domain.

IPPI contains an implementation of Fast Fourier Transformation (FFT) of images. However, in order to perform an FFT in IPPI, the image has to be an order-two size. We solve this by zero padding both the image and the filter. An example is shown in Figure 6.2.
At this point in time the size of LS's images represents a problem to the overall performance of our program. When running the multi-scale filtering method, we

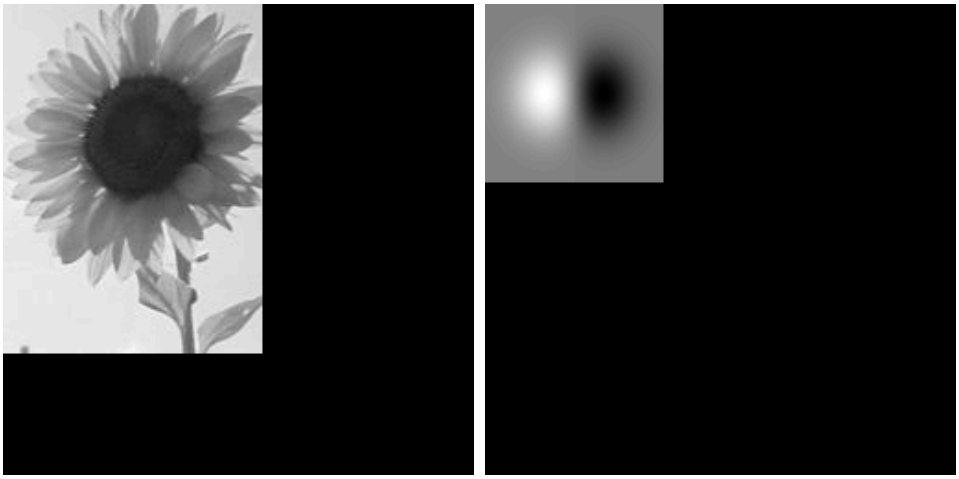


Figure 6.2: $_{a\ b}$ a) The input image of size 190x140 is zero padded to the nearest order-two size 256x256. b) The filter $\frac{\partial N_\sigma}{\partial x}$ is also zero padded to size 256x256.

allocate space according to 5 colour images in the memory. LS's images are in size 3072x2048 and are zero padded to size 4096x2048. Since we are working with 32 bit floating point 5 colour images require approximately 490 MB of memory. Table 6.1 shows which images we allocate space for. Since our test computer is only

Image	Channels	Memory
Original image	3	98.3 MB
FFT of the original image, $F(X)$	1	32.8 MB
Spatial filter	1	32.8 MB
FFT of the filter, $H(X)$	1	32.8 MB
$H(X)F(X)$, x-direction	3	98.3 MB
$H(X)F(X)$, y-direction	3	98.3 MB
Final gradient magnitude image	3	98.3 MB
Total		491.6 MB

Table 6.1: A list of images that we allocate memory for when preforming frequency edge detection filtering. The memory sizes are computed for 32 bit floating point images of size 4096x2048.

equipped with 512 MB RAM, it starts to swap to the hard drive, which slows down the processing enormously. We will come back to this time issue and possible ways around in Section 7.2.1.

6.3 Graphical User Interface

As has been mentioned widely throughout this thesis, our segmentation method is supposed to be used interactively. This in general term means that a user gives some input, and the program swiftly and accordingly returns an output.
Furthermore, since the min-cut/max-flow algorithm is based on initial user input, we need a user interface. We have set up a temporary graphical user interface (GUI) to facilitate the interaction needed during the developing and testing of our program. The GUI is temporary the sense that it is not finally customised to fit into LS's production.
Figure 6.3 shows a screen dump from a segmentation task using our implementation. The left window is used to input seed points, and the segmentation result is shown in the right window. At the bottom is the segmentation toolbox dialog, which is also shown in larger format in Figure 6.4. To the left in the dialog are too

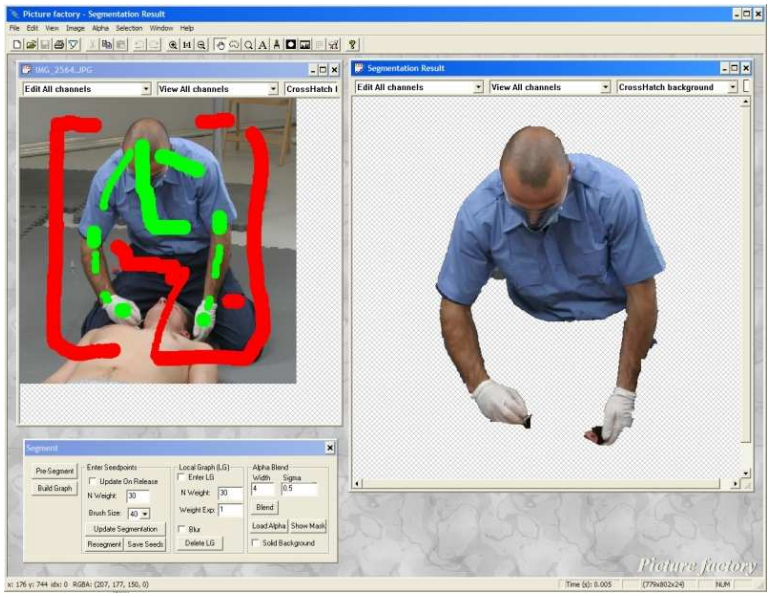


Figure 6.3: Picture Factory GUI during interactive segmentation.

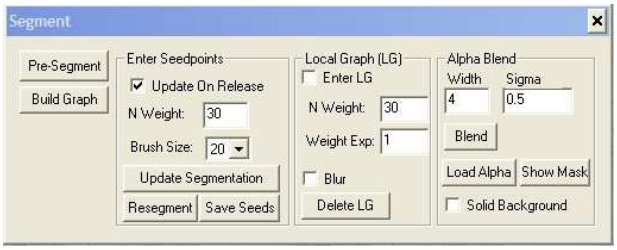


Figure 6.4: The segmentation dialog box.

buttons used to carry out the initial pre-segmentation and to construct a graph over the input image. These functionalities should probably not be in a final GUI, since they precede the interactive part of the segmentation process.
The rest of the instruments in the dialog are gathered in boxes containing related functionalities. First we have the components used to enter seeds. Here the user has the possibility to change the size of the brush used to input seeds, and he can choose to either update the segmentation or re-segment from scratch. The neighbour weight can be changed ahead of re-segmentation.
The centre box contains the functionalities used for local graphs.
In the right side of the dialog box are presently features associated with our implementation of the local alpha estimation.

We have identified three additional features that would greatly increase the useability of the GUI with respect to the segmentation task. The most important is an ”undo” feature for deleting erroneous seeds. The second would be the possibility of entering seeds directly in the result window. A third convenient functionality would be an intelligent resizing of the image windows to fit the size of the processed image.

Chapter 7

Test

This chapter presents tests of various aspects of the produced implementation. First section contains what we call a limited functional test. Here we want to show that the most critical part of our program, namely the Boykov-Kolmogorov algorithm behaves predictably under changing circumstances. This is taken as an indication of the correctness of the implementation.
Secondly, we present a time study that illuminates important performance issues in the implementation.
Finally, we will make a qualitative assessment of the image segments outputted from our program. A part of this test will be to compare the produced segments with the manually segmentation results from LS's current working procedure.

Throughout the test we use image material provided by LS. Images of paramedics in blue shirts and dark trousers seem to be over-represented in the provided material. This image type, however, represents some of LS's most important products which is why we show our implementation's performance on these images in particular. We have received a few other images which we also include in the test.

7.1 Limited Functional Test

The purpose of this section is to investigate whether our implementation works correctly.
Firstly, we compare segmentation result for a very small image with ground truth to be convinced that our graph cut implementation finds the correct segment.
Secondly, we test whether the run-time for our implementation stays within the worst case upper-bound for the Boykov-Kolmogorov algorithm.
Thirdly, we look at the relative importance of the neighbourhood criteria versus the colour statistics. The neighbourhood weight, λ is adjusted to test the influence from the terms separately.

7.1.1 Manual Minimum-Cut

We have made a manual minimum-cut on a very small image. We run the Boykov-Kolmogorov min-cut algorithm by hand in order to obtain ground truth for the image. This is done to see whether our implementation finds the same paths and ultimately the same segment.
Figure 7.1a contains an illustration of the 4x4 pixel input image. A graph is constructed over the image. Two pixels are chosen as foreground seed and background seed, and we compute edge capacities according to the equations in Section 4.4. The initial graph is seen in Figure 7.1b. Note that all the shortest paths, $s \rightarrow p \rightarrow t$, through all nodes, p, have been pre-augmented to saturated half of the t-link before the the graph search is started.
Subsequently, we follow the stages of the algorithm as described in Section 3.2.5 which results in the cut shown in Figure 7.1c. The manually found cut is identical

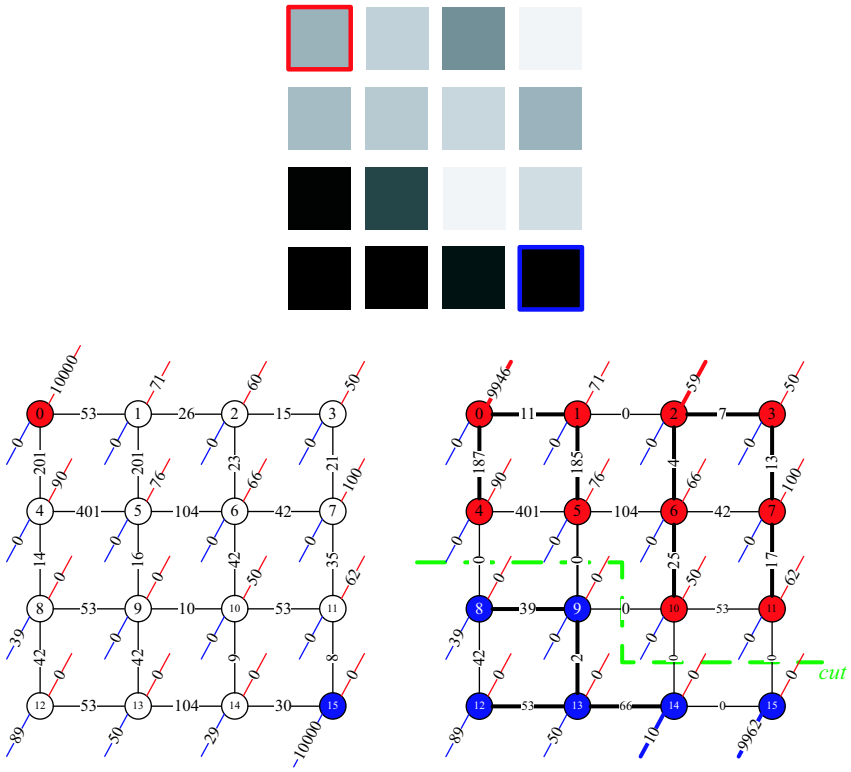


Figure 7.1: $\begin{smallmatrix} & a & \\ b & & c \end{smallmatrix}$ a) In the manual graph cut segmentation is used a grey scale image of size 4x4. b) The initial graph is based on the input image. The red edges are t-links connecting the nodes to s whereas the blue edges connect to t. Black edges are n-links. For the seeds we set the t-link capacities to 10000 to encode infinite capacity to their respective terminals. For convenience all edge capacities are scaled by a factor 100. Initially, only the terminals s and t are in the active list. c) The green line through the graph indicates the cut that has been made.

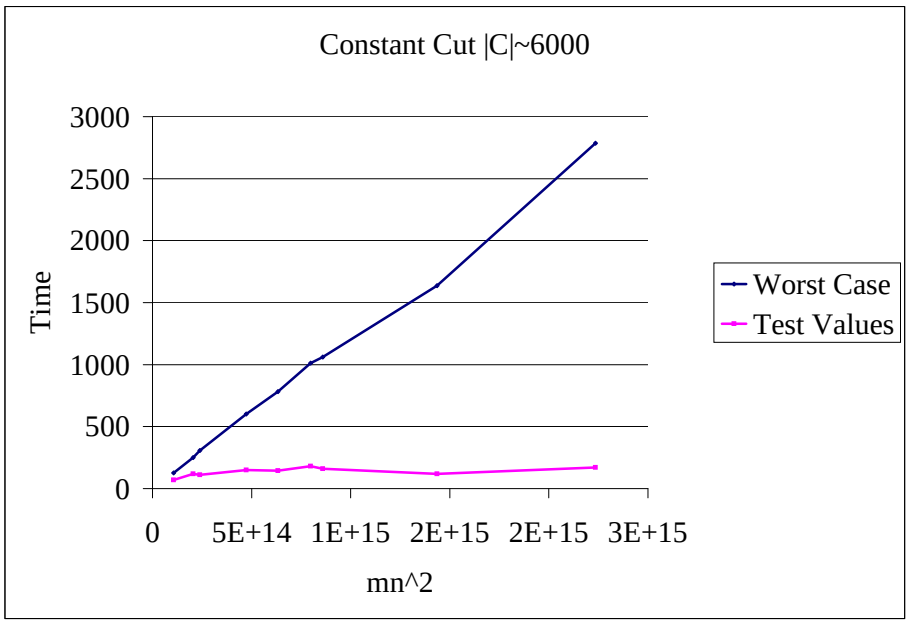
to the one found using our implementation. We interpret this as an indication that we have implemented the algorithm correctly.

7.1.2 Run-time Versus Worst Case Complexity

Boykov and Kolmogorov [2] argue that their augmenting path algorithm in practise outperforms other algorithms though it has a worse worst case complexity, see Section 3.2.5. The worst case complexity for the Boykov-Kolmogorov algorithm is $O(mn^2|C|)$. If [2] is right we should expect a run-time significantly better for our implementation than the worst case run-time.

There is a practical problem in having the worst case complexity expressed in terms of the cut capacity, $|C|$. The problem is that even for the same image the value of $|C|$ may vary depending on which segment we find in the segmentation. To make a proper comparison we must make a series of tests in which we maintain a constant cut capacity $|C|$ and vary the size of the graph mn^2.

Our approach has been to make a large number of tests and choose the test results that fall into a certain range of a constant $|C|$. This approach has resulted in the two graphs shown in Figure 7.2. We have used $|C| \approx 6000$ and $|C| \approx 10000$. The actual values of $|C|$ vary within $\pm 5\%$ of the chosen constants.
From both Figure 7.2a and 7.2b it is clear that the time evolvement of our implementation has a significantly lower gradient than its worst case for the algorithm. This indicates two things. First, it indicates that Boykov and Kolmogorov are right to say that their algorithm *in practice* performs very good though having a worse worst case complexity than other algorithms used to find a minimum cut. This being true, this test secondly indicates that our implementation of the algorithm is done correctly.



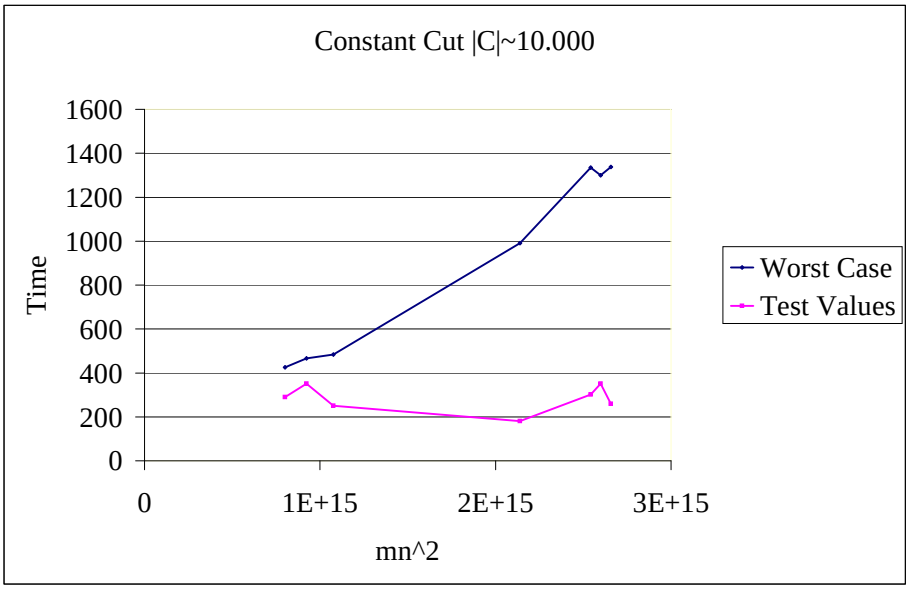


Figure 7.2: $\frac{a}{b}$ Plots of actual run-times vs. worst case run-times. $|C|$ is held constant while the graph complexity mn^2 is varied. The worst case times are calculated and scaled to fit the graph. a) For $|C| \approx 6000$ b) For $|C| \approx 10000$

7.1.3 Separating Energy Terms

The following test is performed to test the effect of varying λ. We will see whether the program behaves as expected when segmenting simple synthetic images for which we are able to predict the result. Inspired by Boykov & Jolly [1] we have prepared a synthetic image (see Figure 7.3) to test the relative importance of the neighbourhood versus the colour statistics. Referring to the energy equation (Equation 3.1.2) it is seen that $\lambda = 0$ corresponds to minimising only the colour statistic term, $D_p(\cdot)$. On the other hand extremely high λ-values will in practice correspond to minimising only the neighbourhood term $V_{p,q}(\cdot)$.

We are going to segment the image using the two different sets of seeds. In Figure 7.4 we only placed foreground seeds inside the black boxes to obtain a clear difference in the colour statistics between foreground and background. With these seeds we expect to segment only the black boxes in the image, when we use low λ-values.

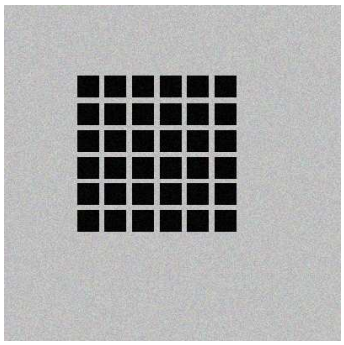


Figure 7.3: The input image is a synthetic greyscale image of size 400x400. We added some Gaussian noise to the image to obtain more interesting colour distributions for the entered seed points.

Indeed, for $\lambda = 0$ no capacity is given to the n-links, and predictably we get the desired result as seen in Figure 7.4 since the segmentation is purely based on colour. The result is repeated for all values of $\lambda < 83$. After this point the n-links get enough importance to effect the segmentation, and some of the boxes are omitted from the segment. Ultimately for $\lambda \geq 250$ the relative importance of the t-links has become so insignificant that only boxes which have been explicitly marked with foreground seeds become part of the segment.

In figure 7.5 foreground seeds are inputted as a sweep over the black boxes and the grey background in between. With these seeds we expect to segment all the boxes along with parts of the background that are between the boxes.
The sweep of foreground seeds gives overlapping colour distributions for foreground and background. For $\lambda = 0$ this results in a scattered segmentation because the algorithm only relies on colour information. All over the image some

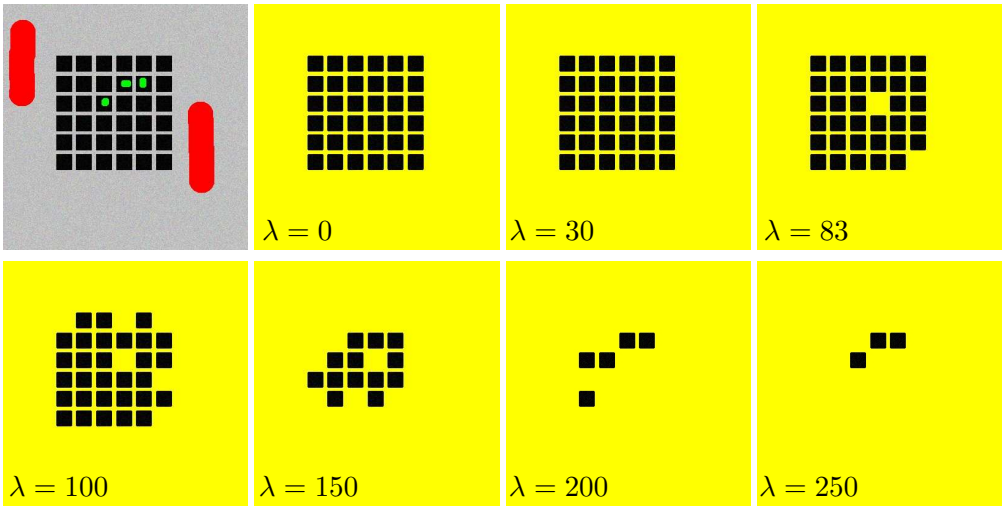


Figure 7.4: Segmentation results for different values of λ using the seeds shown in the upper left image.

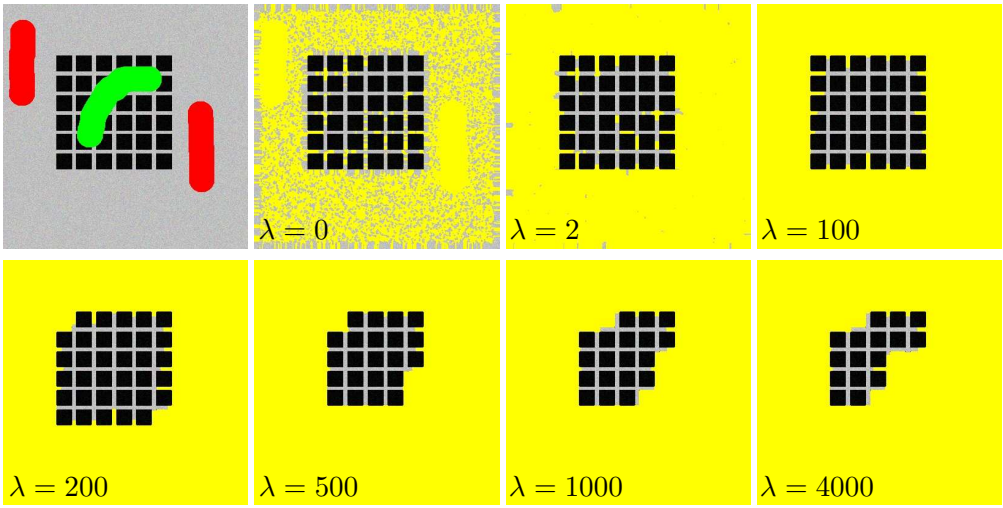


Figure 7.5: Segmentation results for different values of λ using the seeds shown in the upper left image.

pre-segments get the foreground label apart from in the areas explicitly marked as background.
For $\lambda = 2$ it has already changed significantly - there are only some islands here and there. The results support our expectation that an increase in λ yields a more dense segment, because the neighbour term enforces coherence. In the segmentation result for $\lambda = 100$ all the boxes and almost all the area in between are included in the segment. The n-links have become sufficiently important to remove islands and to close holes between the boxes.

For $\lambda > 100$ the segmentation omits an increasing number of boxes from the segment. It is no longer sufficient for the boxes to have similar colour to the foreground statistics. Finally, for $\lambda = 4000$ we see that only boxes marked as foreground are included in the segment, because we practically disregard the colour statistics.

The two tests in this section confirm our expectation that the influence from the two terms of the energy equation can be controlled using different values for λ. We used extreme λ-values to show the separate effect of the two terms. All results indicate that each of the two energy terms have the expected and desired properties.

7.2 Time Study

In this section we go into a detailed test of algorithmic run-times for different parts of the segmentation method. The purpose of this test is to determine the conditions under which our application can be used as an interactive segmentation method. Furthermore, the test should give the reader a general impression of the application's speed and bottlenecks etc.

All tests in this section are run on a Pentium M 1.6 GHz with 512 Mb RAM. The run-times have an average error of $\pm 10\ ms$ [24].

7.2.1 Pre-Segmentation

The pre-segmentation consists of two major parts: obtaining a gradient magnitude image in a multi-scale framework, and running the toboggan watershed algorithm. As explained earlier the gradient magnitude image is obtained using an edge detection frequency filter. The fact that the FFT function we use requires image and filter to be of order 2 size has the effect that the filtering time can be divided into groups of order 2 images. We illustrate this in Figure 7.6. The figure shows filtering run-times on images for sizes ranging from 512x256 to 4096x2048. For all runs four scales were used in the multi-scale edge detection, σ=$\{1, 2, 4, 8\}$. The

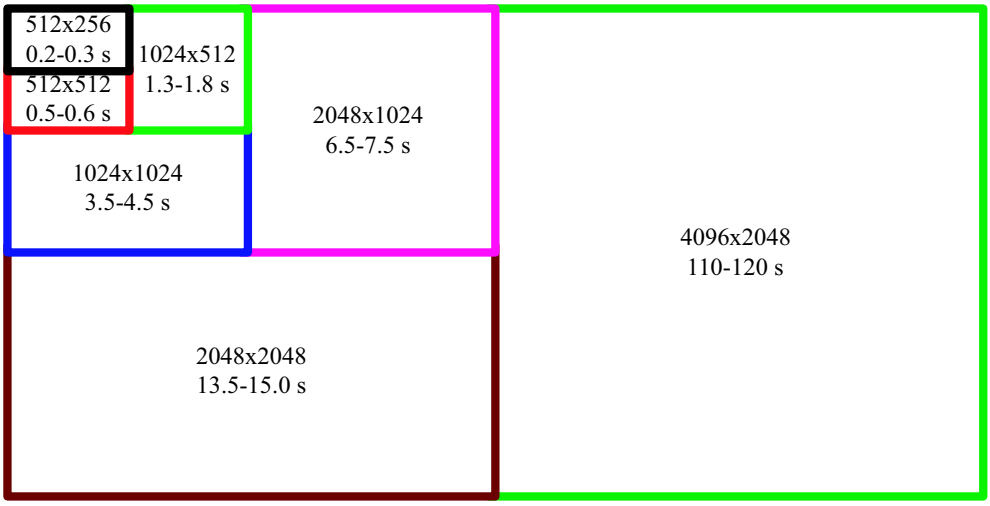


Figure 7.6: Illustration showing frequency filtering run-times for different sizes of order 2 sized images.

run-time specified for the largest image 4096x2048 is significantly higher than the others. The reason for this is that for images of this size the test computer runs out of conventional RAM and starts to swap to the hard drive. This, of course, slows the filtering process a lot. We have, however, included this example to show the

limitation of the implementation. A computer with 1024 Mb of RAM would have been able to filter images of this size without running out of RAM.
Exact run-times can be seen in Table 7.1 for eight images of different size.

Having the gradient magnitude image the next step is to run the toboggan watershed algorithm. The toboggan watershed run-times can be viewed in Table 7.1 for the eight example images.

Size	No. of Nodes	NR	Filtering	Toboggan Watershed	Total
256x359	4400	20.9	0.24 s	0.11 s	0.4 s
500x692	16007	21.6	1.38 s	0.40 s	1.8 s
899x881	47580	16.6	3.45 s	0.92 s	4.4 s
1195x885	83349	12.7	6.67 s	1.21 s	7.9 s
1082x1228	99413	13.4	13.79 s	1.53 s	15.3 s
1794x1305	159990	14.6	14.00 s	2.77 s	16.8 s
1629x1602	190635	13.7	15.00 s	3.13 s	18.1 s
3072x2048	554778	11.3	110.00 s	9.75 s	119.8 s

Table 7.1: Time table for the pre-segmentation step. The table shows the time used to obtain the gradient magnitude image and the time used for the toboggan watershed algorithm. The total time reflects the time the operator has to wait before the actual segmentation can take place.

Figure 7.7 shows the measured toboggan run-times. The plot shows that our implementation runs in linear time. When evaluating the total time used for pre-

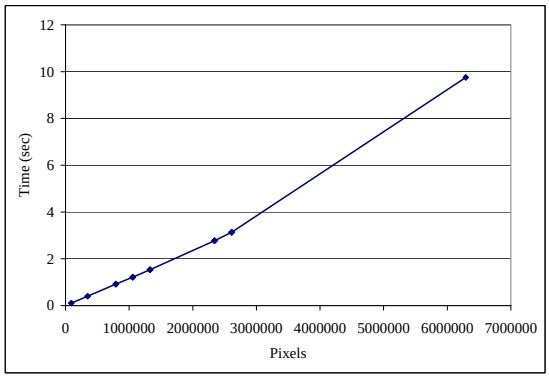


Figure 7.7: A plot showing that our implementation of the toboggan algorithm runs in linear time.

segmentation, we must keep in mind that the pre-segmentation is done to obtain interactivity in the graph cut segmentation. This means that the time used to pre-

segment should not surpass the time won by running the graph cut algorithm on a smaller graph. We will, make come back to this in Section 7.2.2.1.

As seen from Table 7.1 the pre-segmentation of small images is almost instant. For larger images the time used for pre-segment increases significantly. In LS's case the images are very large - 3072x2048 pixels, which means that the pre-segmentation time will be in the area of $\frac{1}{2}$ minute, which is too much idle time for an operator. LS should consider the possibility of pre-segmenting all images offline as a pre-processing step. The operator would then load pre-segmented images and perform the graph cut segmentation interactively.

An alternative solution to the time problem could be to allow the operator to make a crop in the image before pre-segmenting. This would also serve to speed up the graph cut segmentation.

7.2.2 Graph Cut Segmentation

The time used for graph cut segmentation can be split up in smaller sub-groups. In this section we will run tests on the same eight images as in the previous section. The settings used for the preceding pre-segmentation will therefore also be the same as in the previous section. For all test we will use an undirected graph.

Table 7.2 gathers the test results for the eight images. The table contains information about the following: The *Size* of the original image. The number of *Nodes*, in the graph corresponds to the number of pre-segments from the initial pre-segmentation. *No. of paths* is the number of paths augmented to obtain the segmentation. *Av. path length* is the average length of all augmented paths where the length of a path is given as the number of edges in the path. *Build graph* is the time used to construct the graph having a pre-segmented image. *Set n-links* is the time spend to compute the n-link capacities. *K-means* is the time used to perform the K-means clustering of all foreground and background seeds. *Set t-links* is the time used to compute the t-link capacities. *Graph search* is the time used to run the Boykov-Kolmogorov augmenting path algorithm.

To see the effect of the number of seeds added for each segmentation we have run some of the tests with two different amounts of seeds. Run-times for the graph construction and the assignment of edge capacities only appear once per image example because they are independent of the number of seeds used. The times specified in Table 7.2 can be divided into two groups: The time spend before user input and the time spend after user input. Building the graph and computation of n-link capacities are done before user input and K-means, computation of t-links as well as graph search is done after.

Table 7.2 shows that the time used before user input is low even for very large

Size	Nodes	No. of Seeds	No. of Paths	Av. Path Length	Build Graph	Set n-links	K-Means	Set t-links	Graph Search	Total
256x359	4400	1038	5444	2.63	0.08 s	0.01 s	0.02 s	0.05 s	0.01 s	0.17 s
500x692	16007	3041	19686	3.56	0.29 s	0.04 s	0.09 s	0.19 s	0.09 s	0.70 s
899x881	47580	5677	55698	4.12	0.79 s	0.12 s	0.25 s	0.59 s	0.23 s	1.98 s
		9830	55469	4.88			0.48 s		0.43 s	2.36 s
1195x885	83349	3307	88177	2.29	1.12 s	0.20 s	0.10 s	0.99 s	0.17 s	2.58 s
		12739	87884	2.25			0.45 s		0.17 s	2.94 s
1082x1228	99413	8937	111291	3.17	1.39 s	0.24 s	0.49 s	1.18 s	0.36 s	3.67 s
		16699	111259	3.07			0.99 s		0.34 s	4.16 s
1794x1305	159990	8941	181608	3.96	2.33 s	0.38 s	0.30 s	1.88 s	0.85 s	5.75 s
		24515	177570	3.31			1.21 s		0.63 s	6.44 s
1629x1602	190635	11930	216021	3.58	2.70 s	0.46 s	0.53 s	2.24 s	0.86 s	6.80 s
		30041	213686	3.38			1.76 s		0.79 s	7.99 s
3072x2048	554778	30035	569104	2.27	14.68 s	1.34 s	1.26 s	6.57 s	1.13 s	24.99 s

Table 7.2: Table showing run-times for different parts of the graph cut segmentation method. The test is run on eight different images of different size. For some images the test include adding a higher number of seeds, which explains why some images cover two lines in the table. All tests are run using an undirected graph.

images. This, in combination with our own experience with the program, leads us to conclude that the initial processing does not affect the interactivity of the method.

The table also shows that the time spend after entering seeds is highly depended on the number of seeds entered. We would expect a faster graph search with more seeds because more seeds are initially labeled. This is also the general trend, but the extra time used on K-means clustering because of more seeds by far surpasses the time saved in the graph search. This leads us to conclude that more seeds do not reduce the total segmentation time. It might, however, increase the segmentation quality, which we shall see in Section 7.4.

An interesting observation from Table 7.2 is that more seeds not always result in faster graph search. To find an explanation for this we have to look at the average path length which has a high correlation with the run-time of the graph search. The image of size 899x881 has higher run-times for more seeds. This is due to the placement of seeds that evidently results in longer paths.

Computing the t-link capacity is the most time consuming task after the input of seeds. This might seem a little odd compared with the time spend on computing the n-link capacity. The reason is, however, that computing the t-links requires comparison of all nodes with all K-means clusters in the foreground and background

sets.

7.2.2.1 Directed Versus Undirected Graphs

To make a time comparison of the directed graph and the undirected graph we have run the exact same examples as in Table 7.2 using a directed graph. Figure 7.8 shows the run-times for the graph search using graph types. The figure shows

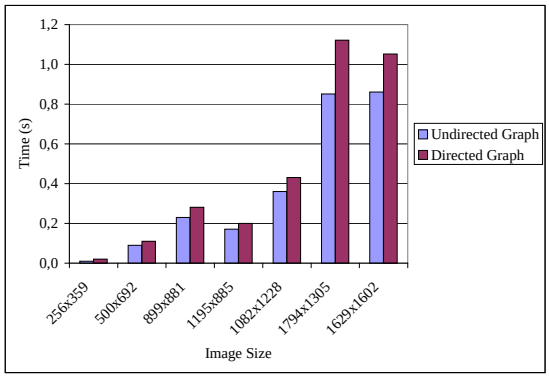


Figure 7.8: Shows the differences in run-time of the graph search for the directed and the undirected graph.

that the undirected graph is faster than the directed. The differences are, however, not large enough to discharge the directed graph. The decision of which graph type to use will therefore be dependent on the quality of the segments they produce.

7.2.2.2 Pre-Augmentation of t-links

In Section 3.2.5.4 we argued that is it was possible to saturate half of all t-links before running the Boykov-Kolmogorov augmenting path algorithm. This would correspond to augmenting the number of nodes n shortest pathes in the graph. In the following test we show that making the pre-augmentation has a significantly effect on the segmentation run-time. Practically, the pre-augmentation is done when we calculate the t-link capacities. Table 7.3 show the graph search run-times for three different image sizes. For the first row for each image we have used pre-augmentation of t-links and in the second we have not used pre-augmentation. From the table we can clearly see that by pre-segmentation the average path length decreases which results in a much faster graph search. Because be do the pre-augmentation when we set the t-links we see a small increase in the time used in this step. This is, however, of minor concern compared with the times used for graph search.

To ensure that the pre-augmentation step does not affect the result of the graph

Method	Size	Av. Path Length	Set t-links	Graph Search
Pre-augmentation	500x692	3.6	0.20 s	0.09 s
No Pre-augmentation		24.9	0.20 s	1.09 s
Pre-augmentation	1195x885	2.3	1.01 s	0.18 s
No Pre-augmentation		64.3	0.95 s	28.28 s
Pre-augmentation	1794x1305	4.0	2.01 s	0.89 s
No Pre-augmentation		53.0	1.84 s	49.48 s

Table 7.3: Time table for running the Boykov-Kolmogorov augmenting path algorithm with and without initial pre-augmentation of t-links.

cut segmentation we have compared output images for each of the approaches and the these show exactly the same segmentation. We have, however, decided not to show two identical segmentations.
Boykov & Kolmogorov [2] does not describe this pre-augmentation step in their algorithm. However, comparing our run-times above with run-times presented in [2] it could indicate that they in fact do something similar to our pre-augmentation step.

7.2.2.3 Graph Cut Without Pre-Segmentation

The main reason for pre-segmenting the image before building the graph is to lower the size of the graph, which in turn lowers the segmentation time. Table 7.4 shows run-times for the three first examples from the previous section - but now without the initial pre-segmentation. Each pixel in the image now represents a node in the graph. These three examples clearly show that pre-segmenting the image has

Size	Nodes	No. of Seeds	No. of Paths	Av. Path Length	Build Graph	Set n-links	K-Means	Set t-links	Graph Search	Total
256x359	91904	1082	133987	6,4325	0,61 s	0,19 s	0,01 s	1,51 s	1,03 s	3,34 s
500x692	346000	3168	373458	15,9106	2,02 s	0,65 s	0,01 s	6,32 s	15,16 s	24,17 s
899x881	792019	10186	831435	10,7532	4,38 s	1,48 s	0,15 s	13,32 s	23,45 s	42,78 s

Table 7.4: Table showing run-times for the three first test examples without pre-segmentation. The number of nodes in the graph is now equal to the number of pixels in the graph.

a significant importance in order to get an interactive segmentation method. In Figure 7.9 we illustrate the difference in run-time between using a pre-segmented image and a not pre-segmented image. The run-times shown for the pre-segmented case include the time used for pre-segmentation but still it is significantly faster. It should, however, be noted that some optimisation could be done if the graph

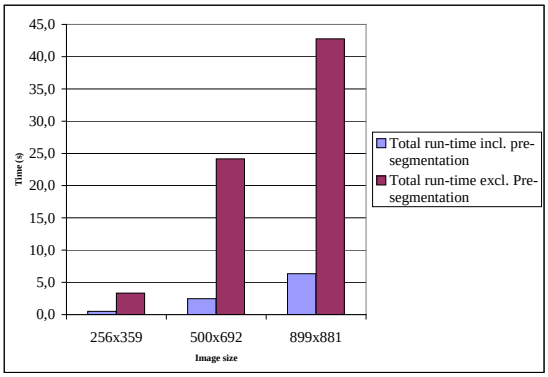


Figure 7.9: Shows the differences in run-time between running a segmentation on a pre-segmented image and an image with no pre-segmentation. The time used for filtering and watershed segmentation is included in the figure.

was specially designed to always work on a pixel-based graph. This is a probable explanation to why the run-times for the pixel-based graph presented in [11] are much lower than our.

7.2.2.4 Adding Additional Seeds

As explained in Section 4.6 we can fine tune the segment by inputting additional seeds. The fastest way to do this is to reuse the graph obtained from the initial graph search. We have made two examples showing the time used for an updating graph search after new seeds have been added. The results are shown in Table 7.5. For each of the two images we have added more seeds twice. The table shows how many seeds were added and how many additional paths were augmented.

Size	Nodes	No. of Seeds	No. of Paths	Av. Path Length	Build Graph	Set n-links	K-Means	Set t-links	Graph Search
899x881	47580	6312	57458	5,21	0,77 s	0,14 s	0,20 s	0,55 s	0,41 s
		96	182	5,24	0,01 s	-	-	-	0,04 s
		57	332	5,37	0,01 s	-	-	-	0,07 s
1629x1602	190635	17195	219723	4,54	2,77 s	0,56 s	0,87 s	2,16 s	1,32 s
		358	209	4,54	0,04 s	-	-	-	0,14 s
		1549	431	4,55	0,05 s	-	-	-	0,18 s

Table 7.5: Table showing run-times for performing an updating graph search after adding additional seeds. The times specified under *Build Graph* cover the initial construction of the graph and reset of the search trees when adding more seeds.

Table 7.5 shows very fast updating times when adding new seeds. In practise, when the user lets go of the mouse button, the segmentation result will update instantly.

This, of course, fits very well into the interactive segmentation framework.

As mentioned in Section 4.6 we should, however, keep in mind that reusing the graph prevents us from using the colour statistic for all t-links in the graph. If we wish to use the newly obtained colour information, we have to reset the flow for the whole graph and find the minimum cut from scratch. This would be just as time consuming as the initial segmentation.

7.3 The Neighbourhood Weight

In Section 7.1.3 we tested the effect of varying the neighbourhood weight. In this section we also vary the λ-value, but this time the purpose is to estimate which λ-values are useful for real imaging tasks. The tests in this section are made on natural images.

The test shows several segmentations of 3 images with reuse of seeds for each image. For each run of the segmentation the value of λ is varied. We have picked out results of λ-values that give interesting differences in the segmentation. The segments found in this test are not expected to be perfect since no fine tuning has been done.

The first test image is seen in Figure 7.10a. Here we use the seeds from Figure 7.10b to segment torso, head and arms of the man to the right. We show results for $\lambda = \{1, 5, 10, 30, 50, 100\}$ in Figure 7.11.

Figure 7.10: a b a) Input image cropped to size 1024x1024 b) User inputted seed points

In this example $\lambda = 50$ gives a dense segment with no disconnected island and the arm and the hand to the left are found quite well. Later at $\lambda = 100$ the segment is changed slightly. There are more or less insignificant changes as some parts of the fingers are missing and a part of the background is added to the segment at the elbow junction. On the overall the returned segment is still of decent quality.
For low values of λ there appears a track of the ”seed point brush” in the left part of the segmented images. Due to overlapping colour distributions for foreground and background the area is labeled as foreground in the parts that are not marked explicitly as background. As expected the effect is not present for higher λ-values, because the neighbourhood is emphasised more relative to the colour statistics.
The second test image - seen in Figure 7.12a is segmented using the user input shown in Figure 7.12b. The segmentation results for $\lambda = \{1, 5, 10, 30, 60, 100\}$ are given in Figure 7.13.
In this example we see a dense segment already from $\lambda = 30$. Again there is a track

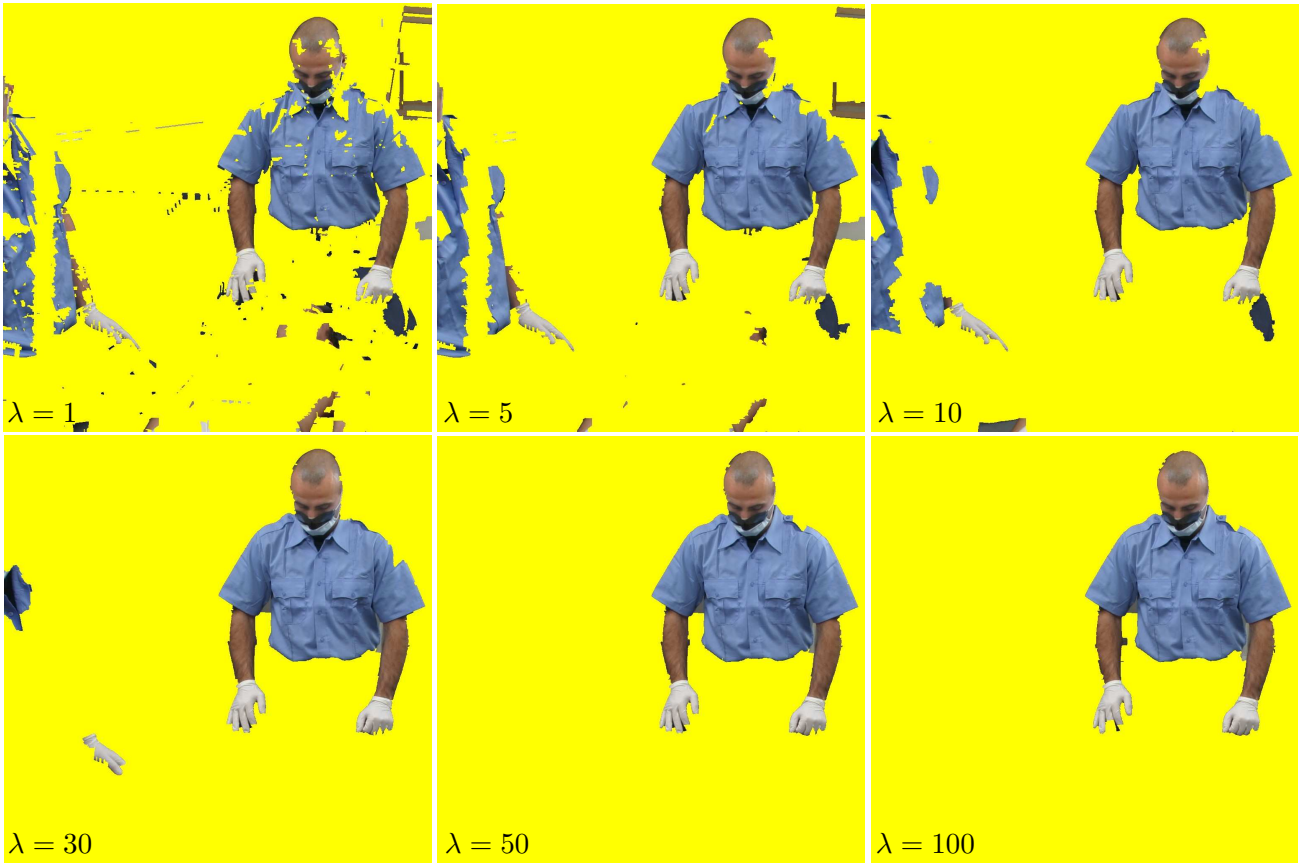


Figure 7.11: Segmentation results for different values of λ.

Figure 7.12: a b a) Input image cropped to size 1024x1024 b) User inputted seed points

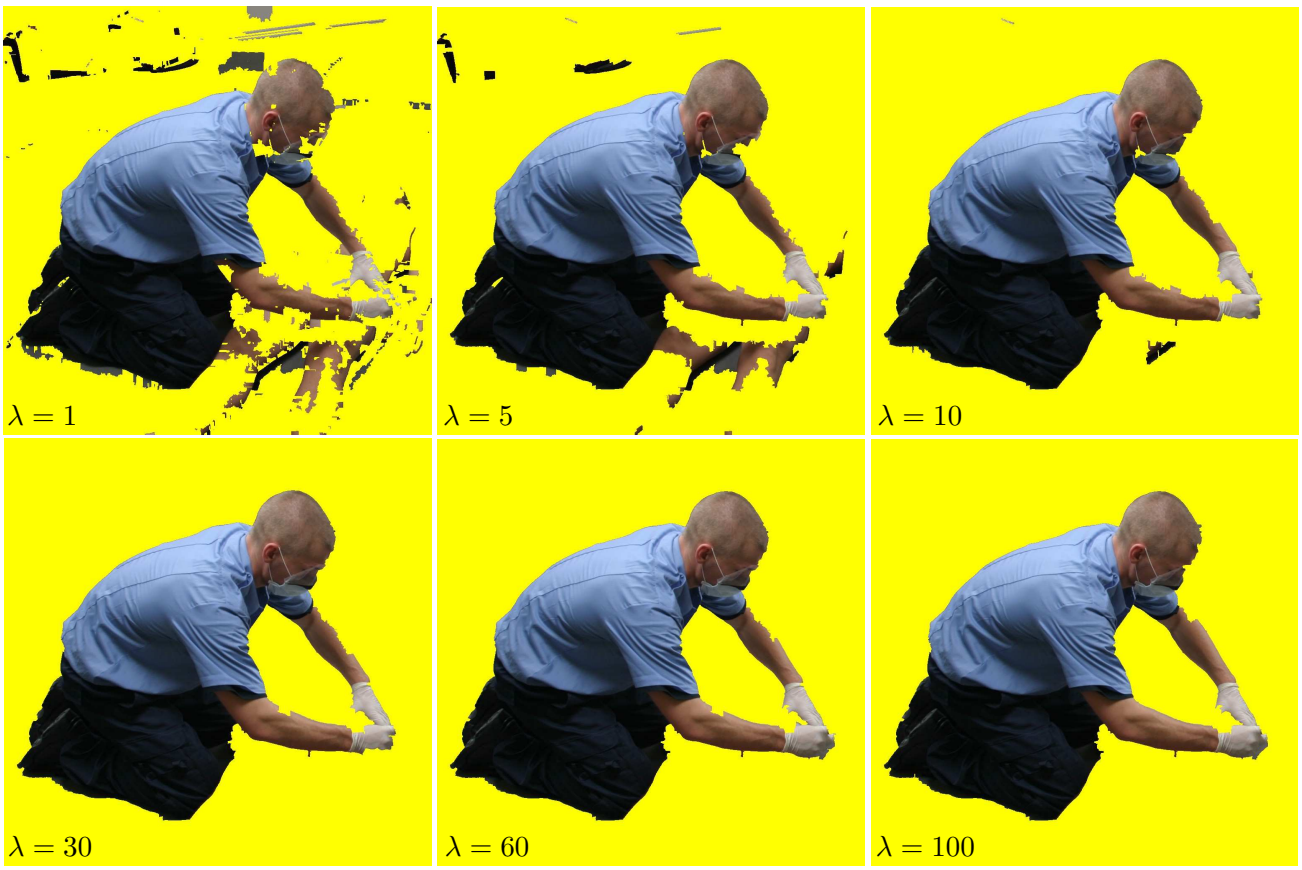


Figure 7.13: Segmentation results for different values of λ.

of the ”seed point brush” under the lower arm. This comes from the similarity in the skin colours of the men in the image. For $\lambda = \{30, 60, 100\}$ there is no clear conclusion to make regarding the quality of the segment. The high values give the best results at the man's arms, but for $\lambda = 100$ his forehead is less good.

The third test is run on the image seen in Figure 7.14a with the seeds from Figure 7.14b. In this example we show results for $\lambda = \{1, 5, 10, 30, 50, 100\}$ in Figure 7.13. This time we see that last island does not disappear until we reach $\lambda = 50$. At $\lambda = 100$ the region between the man's body and arm is filled and erroneously labeled pre-segments appear near the borders of his arms and legs.

The presented examples demonstrate that there is no universal value for λ that always gives the best result. The placement of seeds in relation to the desired cut plays an important role for our choice of λ. Generally, a higher value is needed to achieve a correct segmentation if foreground and background have overlapping colour distributions.

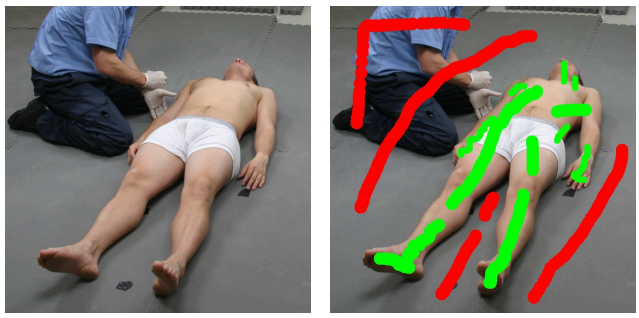


Figure 7.14: $_{a\ b}$ a) Input image cropped to size 1024x1024 b) User inputted seed points

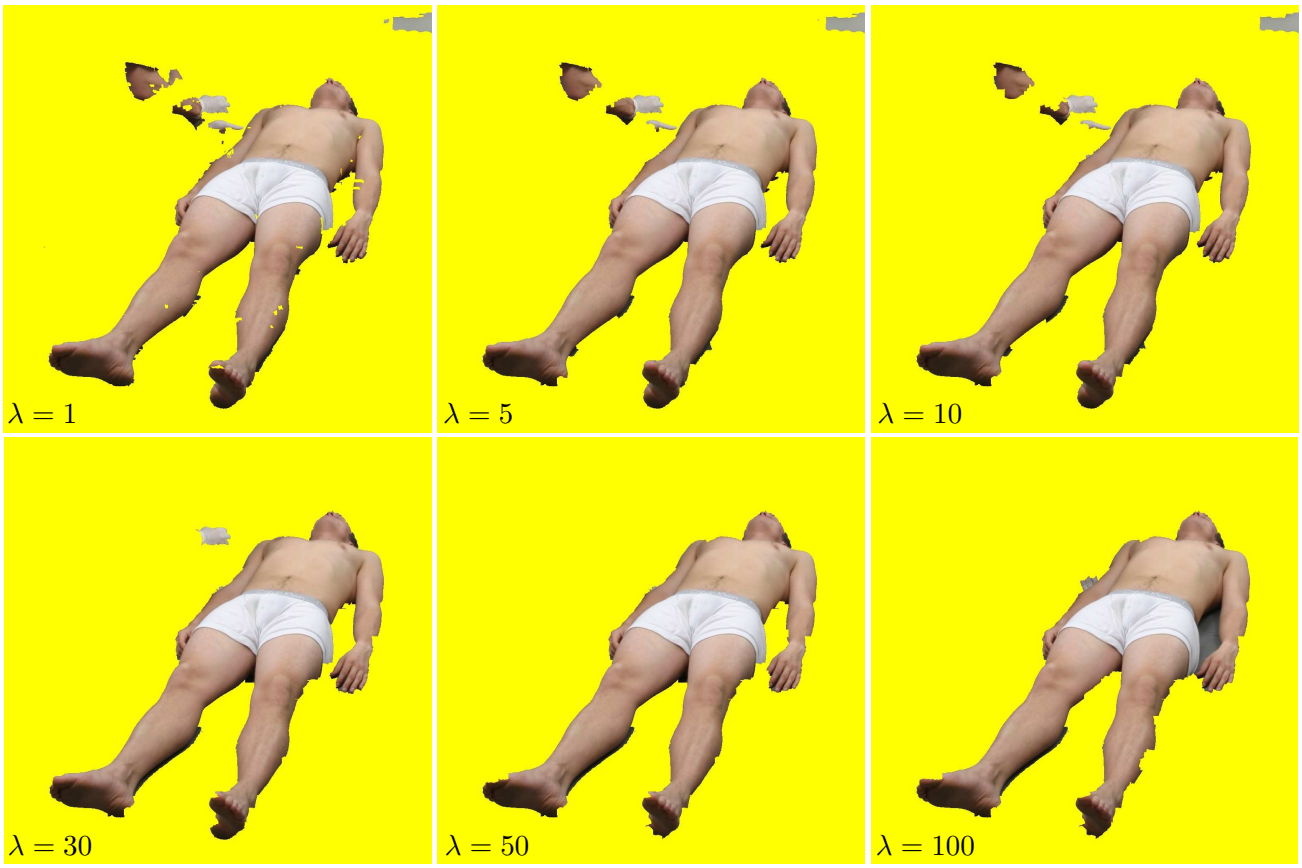


Figure 7.15: Segmentation results for different values of λ.

As a rule of thumb we say that the value should be above 30 to produce a non-scattered segment. On the other hand, we argue that too high λ-values overemphasise the neighbourhood causing the segmentation to cross the existing borders in the image. On this basis we choose to use λ-values, between 30 and 50 for the remaining tests in this chapter.

7.4 Directed Versus Undirected Graphs

We now go on to compare the quality of segments produced by the two different graph representation types that we have been working with - directed and undirected. The purpose is to decide which of the two our final implementation should be based upon.

7.4.1 Initial Segmentation

In the first test we want to show the difference in the initial segmentation using the two graph types. To test this properly we reuse initial seeds for each image, and only substitute the graph implementation before rerunning the segmentation.
We have been studying many LS images and found that in general there is no significant difference in the result of the initial segmentations. Figures 7.16a and 7.16d represent an example of this.
We have also seen some examples of limited differences as between Figure 7.16b and Figure 7.16e. Here the directed graph performs slightly better, but the difference is not quite big enough to discard the undirected graph on this basis.
The third example illustrates typical behavior of the undirected graph. In Figure 7.16f we see a jagged pattern on the segment boundary. There are often a few presegments which are missing here and there in the segmentation produced by the undirected graph. This illustrates a general trend that the directed graph produces the most rounded and complete segments in the initial segmentation.

7.4.2 Fine Tuning

In this second test we show how the graphs respond differently to the updates we make after adding new seeds. Therefore it is not relevant to use the same set of seeds in both graphs. Instead, we show a few representative examples from the fine tuning phase. The examples illustrate how effectively errors can be repaired depending on the underlying graph.

The first example (Figure 7.17) shows a zoom on a man's leg. In the upper row the directed graph was used and below the undirected was used.
The first two images of each row show the initial seed points and the resulting initial segmentation. Both of the segmentations in Figure 7.17b and Figure 7.17f show some problems with the heel of the shoe and with the shadow under the leg.
In the case of the heel the directed graph gives a better initial segment. The undirected graph produces a very jagged initial segment. In the fine tuning phase new seeds are entered as seen in Figure 7.17c and Figure 7.17g. In the undirected representation a lot of seeds are inputted to fix the heel - but still Figure 7.17h presents a bad result compared to Figure 7.17d. Figure 7.17h shows that there are still holes in the segmentation after the update.

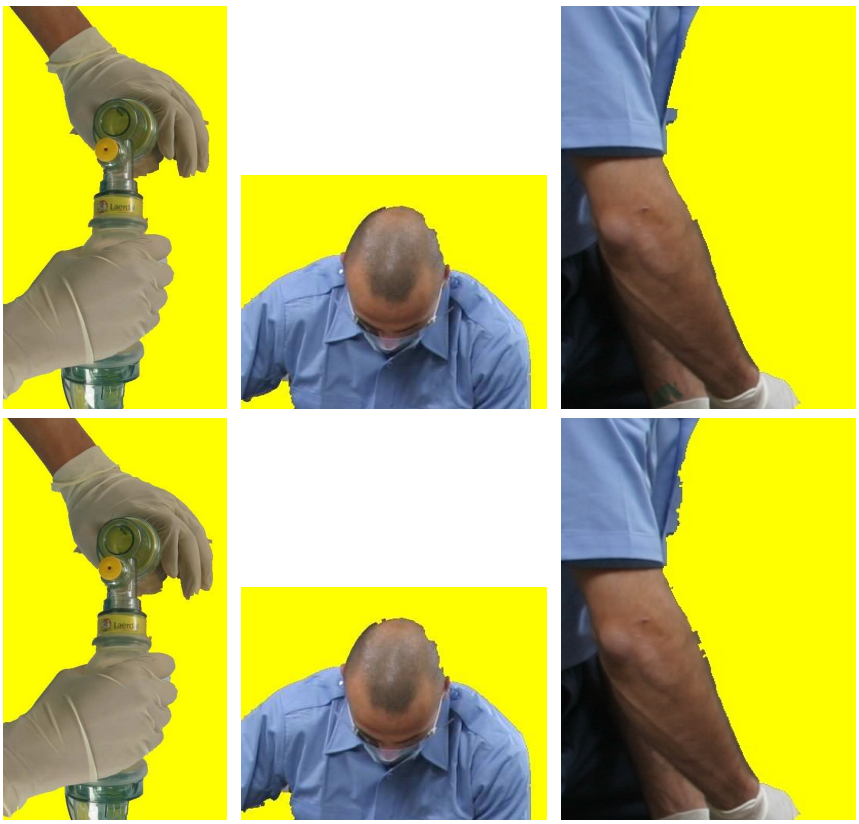


Figure 7.16: $\begin{smallmatrix} a & b & c \\ d & e & f \end{smallmatrix}$ A comparison of initial segmentation results for the two graph types. a, b & c) Results for the directed graph. d, e & f) Results for the undirected graph.

The same effect is seen for the shadow under the leg. In both Figure 7.17c and 7.17g we enter additional seeds to remove the shadow from the segment. Figure 7.17h shows that the undirected graph leaves a few pre-segments behind, these could of course be removed by yet more seeds.

Figure 7.18 shows more examples of the behavior illustrated by Figure 7.17. A comparison of Figure 7.18d and 7.18h shows that an almost identical portion of additional seed gives very different fine tuning results. Again, for the undirected graph some pre-segments remain after the first fine tuning.

7.4.3 Discussion

After studying the relative performance of the two graph representations we get the overall impression that the directed graph yields the best quality. With respect to the initial segmentation the difference is less significant than for the fine tuning. Generally, the directed graph has better properties for adding and removing coherent subparts of the image during the fine tuning. The reason is that the double

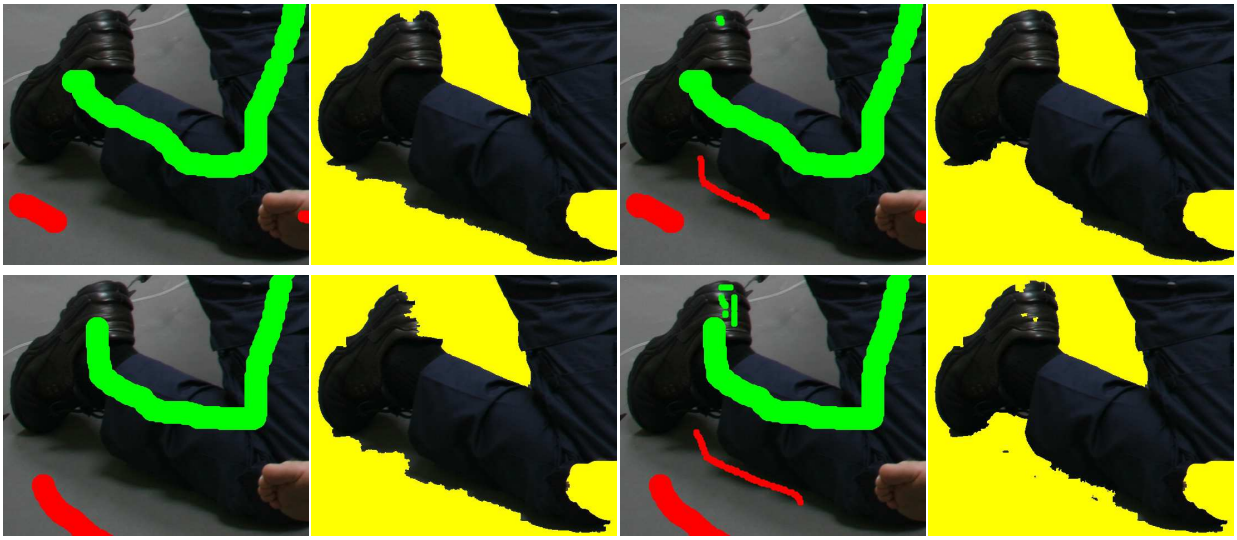


Figure 7.17: $\begin{smallmatrix} a & b & c & d \\ e & f & g & h \end{smallmatrix}$ Illustration of the fine tuning results for the directed graph and the undirected graph. The upper row shows results for the directed graph and below are results for the undirected graph. a & e) Initial seeds. b & f) Initial segmentation. c & g) Additional seeds. d & h) Updated segmentation.

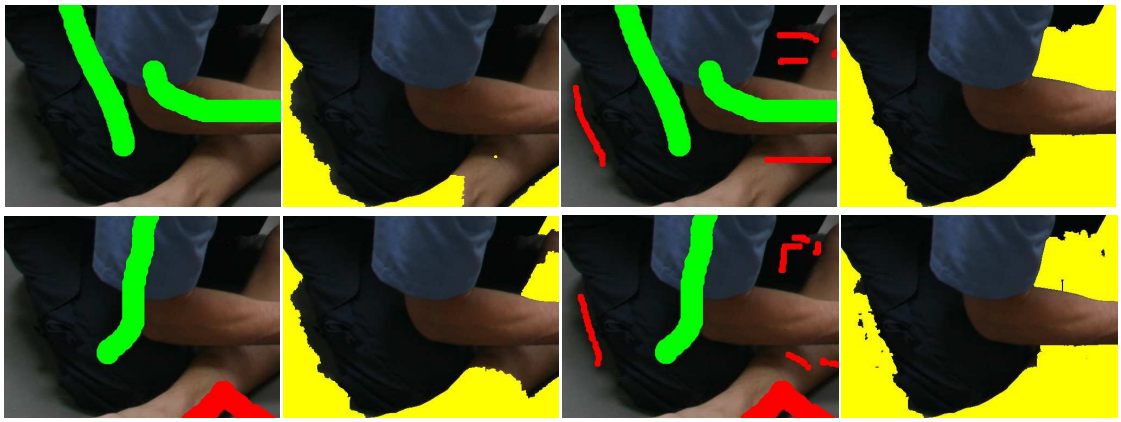


Figure 7.18: $\begin{smallmatrix} a & b & c & d \\ e & f & g & h \end{smallmatrix}$ Illustration of the fine tuning results for the directed graph and the undirected graph. The upper row shows results for the directed graph and below are results for the undirected graph. a & e) Initial seeds. b & f) Initial segmentation. c & g) Additional seeds. d & h) Updated segmentation.

connection between neighbouring nodes in the directed graph allows a saturated edge to opened again. This has the effect that larger areas of neighbouring nodes can shift search tree when we add flow into a local area of the graph. This is also the reason why the directed graph produces more smooth and finished segment borders when more seeds are adding.

7.5 Graph Update

Here we will briefly demonstrate and comment on the different behaviors of the two graph update methods put forward in Section 4.6. One method uses a local approach that only modifies t-links at new seeds, whereas the other method is global in the sense that it assigns new values to all edges in the graph. In Figure 7.19 we show that the local updating method is quite vulnerable to a poorly chosen initial seed points.

The first version of the colour statistics is made from the initial seeds in Figure 7.19a. When new seeds are added in (Figure 7.19c) in the attempt to include the entire paramedic in the segment the local updating method gives the unsatisfying result shown in Figure 7.19d. It is evident that the bad segmentation of his head and arms stems from insufficient colour statistics for these areas. If we instead use the global graph update method we recompute the all t-links based on the updated sets of seeds and n-links are reset to their original capacities. This results in the segmentation shown in Figure 7.19e.
The conclusion made from this small demonstration is not that the local updating method is useless, but rather that one should be thorough in inputting initial seeds. In fact, as will be seen from the rest of the test chapter the local method is very efficient, when used correctly.

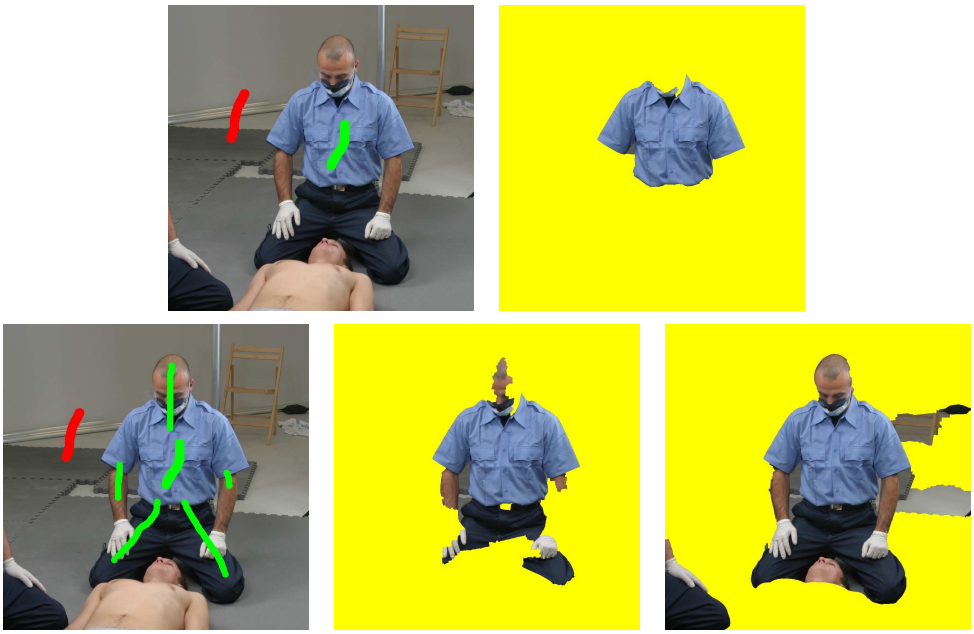


Figure 7.19: $\begin{smallmatrix} a & b \\ c & d & e \end{smallmatrix}$ a) Very few initial seeds are used. b) The segmentation result only includes the blue shirt. c) More seeds are entered in the attempt to include the entire paramedic in the segment. d) The result making local updates to the graph around the new seeds. e) The result of performing a re-segmentation based on the full updated sets of seed points.

7.6 Overall Performance

In this section we will test the overall performance of our implementation. Below we have listed the subparts that comprise our final application.

- Graph cut segmentation based on a pre-segmented image. We use a directed graph.
- Two methods for fine tuning. Firstly, the possibility to enter additional seeds. Secondly, the local graph (marked with blue in all the following tests).
- For border blending we blur the alpha channel with a 5x5 Gaussian filter.

The test is done by letting a test person segment five specific tasks. Hereafter, we evaluate the segmentation quality and total time spent to obtain the segment. Because LS have trained operators, we have chosen a trained operator (one of the authors) as the test person. The time shown is the overall segmentation time which include the time used for user-input as well as the run-time of the program. The time does, however, not include the time used for pre-segmentation.

Throughout this test we will compare our results with hand-made segments from LS when these are available. These segments can be thought of as a sort of ground truth. However, since LS's primary goal is to make their segments look good they sometimes manipulate the segments. These manipulations are not included in our segmentations.

For each segmentation tested we let one of our contacts at LS evaluate the segments and give some comments. These comments are cited at the end of every test.

7.6.1 Case 1

Figure 7.20 shows the original image for case 1. We aim to segment the paramedic in the image.

Figure 7.20: Original image for case 1. Image size 1794x1305. We aim to segment the paramedic.

Figure 7.21 shows three steps in the segmentation process. Figure 7.21a and 7.21b show that the operator relatively fast (40s) can obtain a global segmentation of the paramedic close to the wanted. However, a closer look reveals many smaller errors that have to be corrected. After 1m40s (Figure 7.21c, 7.21d) many of the errors have been corrected, but in order to obtain a satisfying result more fine tuning has to be done. Eventually, after 3m30s the result is satisfying. In the last fine tuning step we have made use of the local graph. The local graph has shown to be very useful in areas where entering of additional seeds would require high precision.

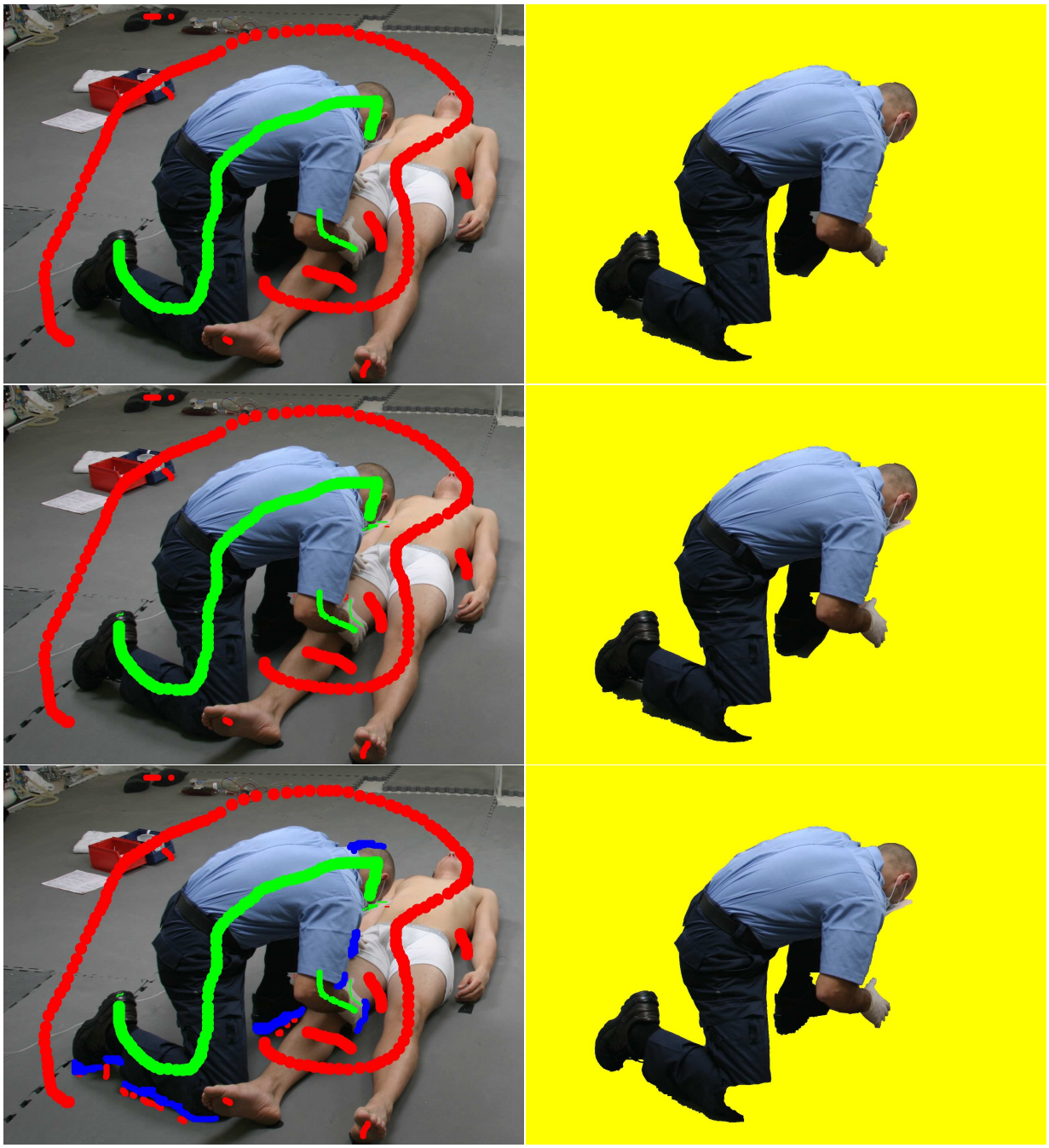


Figure 7.21: $\begin{smallmatrix} a & b \\ c & d \\ e & f \end{smallmatrix}$. Three steps in the segmentation process. a & b) Seeds and result obtained in 40s. c & d) Seeds and result obtained after 1m40s (1 minute 40 seconds). e & f) After 3m30s.

Figure 7.22: $\begin{smallmatrix} a & b \\ & c \end{smallmatrix}$ a) Our segmentation. b) LS segmentation. c) Showing segmentation difference. Cyan areas are where our segmentation is larger than LS and red areas are where LS's segmentation is larger than ours.

Figure 7.22a shows the segmentation result where the alpha channel has been blurred with a 5x5 Gaussian to give a more smooth transition on the border. Comparing with the segmentation from LS (Figure 7.22b) we see a few minor differences. The difference is clearly seen in Figure 7.22c where we have performed a different operation on the two alpha masks. Cyan marks where our segmentation is larger than LS's - and red shows the opposite First of all, LS has filled out the hole where the patient's foot covers the man's knee. Our implementation does not

include methods to make this kind of image editing. Therefore, this difference will not be regarded as an error. One of the most striking differences is the part of the man that touches the floor. Because the paramedic has very dark trousers and at the same time throws very dark shadows on the floor, the contrast between floor and trousers is very low. In our segmentation this results in a rather uneven border. Besides from the shadow areas we evaluate this segmentation as being very good.

Comments from LS *"There are problems with the shoes, the lower part of the right leg, around the fingers on the right hand and of course the knee which is manipulated in the LS segment. Besides this the segment looks good. In a few areas your solution is better than ours. Both your solution and our solution have a total lack of shadows. My impression is that the problem with the shoes and the lower part of the leg can be solved totally by including shadows in the segment. After this the need for manually labor is limited to the fingers and knee."*

7.6.2 Case 2

Figure 7.23 shows the original image for case 2. Again, we aim to segment the paramedic in the image.

Figure 7.23: Original image for case 2. Image size 829x1166. We aim to segment the paramedic.

Figure 7.24 shows three steps in the segmentation process. Again, we see that a satisfying result is obtained using in the area of 3m. However we must conclude that most of the time is used for fine tuning.

Figure 7.24: $\begin{matrix} a & b \\ c & d \\ e & f \end{matrix}$. Three steps in the segmentation process. a & b) Seeds and result obtained in 30s. c & d) Seeds and result obtained after 1m. e & f) After 3m.

Figure 7.25: $a\ b\ c$ a) Our segmentation. b) LS segmentation. c) Showing segmentation difference. Cyan areas are where our segmentation is larger than LS and red areas are where LS's segmentation is larger than ours.

Figure 7.25 compares our segmentation with that of LS. Generally, the result is good, but the comparison reveal that LS sometimes cut into the segment. This is done on the lower part of the paramedic's legs. In difference image, Figure 7.25c, the large cyan areas show where LS has cut into the segment. It seems that LS in general cuts a few pixels into the segment. Our segmentation is, however, good, and we estimate that the differences does not lower the quality of the segment.

Comments from LS *"A really good segmentation which can almost be used directly in our production. The only problem is the lower part of the left foot and the leg. Again, this problem can be solved by including shadows. In general, the segment lacks shadows."*

7.6.3 Case 3

Figure 7.26 shows the original image for case 3. We aim to segment the paramedic.

Figure 7.26: Original image for case 3. Image size 929x1166. We aim to segment the paramedic.

Figure 7.27 shows three steps in the segmentation process. We see the results after 40s, 1m50s and 4m.

Figure 7.27: $\begin{smallmatrix} a & b \\ c & d \\ e & f \end{smallmatrix}$. Three steps in the segmentation process. a & b) Seeds and result obtained in 40s. c & d) Seeds and result obtained after 1m50s e & f) After 4m.

Figure 7.28 compares our segmentation with the LS segment. What makes this segmentation difficult is that the paramedic is holding his white glove over the

Figure 7.28: $a\ b\ c$ a) Our segmentation. b) LS segmentation. c) Showing segmentation difference. Cyan areas are where our segmentation is larger than LS and red areas are where LS's segmentation is larger than ours.

patient's white shorts. This is, of course, a problem because the two white objects are hard to split up. Another difficult area is the lower part of the paramedic's right hand. Since this part of the hand is in the shadow, it becomes almost black and is therefore hard to separate from the background.
Figure 7.28c shows the difference between our segmentation and LS's segment. Generally, LS has cut one or two pixels into the segment, which is why there is a difference all around the segment.
The quality of this segment is not totaly satisfying. The hands and syringe presents a challenge that cannot be segmented probably using our segmentation tool.

Comments from LS *"Some problems with the left foot, the fingers and lower part of the legs. A piece of the lower part of the right hand is missing. No shadows in your solution."*

7.6.4 Case 4

Figure 7.29 shows the original image for case 4. We aim to segment the upper part of the body and the suction-tube.

Figure 7.29: Original image for case 4. Image size 608x833. We aim to segment the upper part of the body and the suction-tube.

Figure 7.30 shows three steps in the segmentation process. We see the results after 40s, 2m and 4m. We obtain a satisfying result in 4 m. As mentioned in Section 4.5.7 it is generally difficult to segment elongated structures because it requires many seeds placed closely all along the long structure. In this test we, however, discovered that the local graph very efficiently segmented the suction-tube structure.

Figure 7.30: $\begin{matrix} a & b \\ c & d \\ e & f \end{matrix}$. Three steps in the segmentation process. a & b) Seeds and result obtained in 40s. c & d) Seeds and result obtained after 2m. e & f) After 4m.

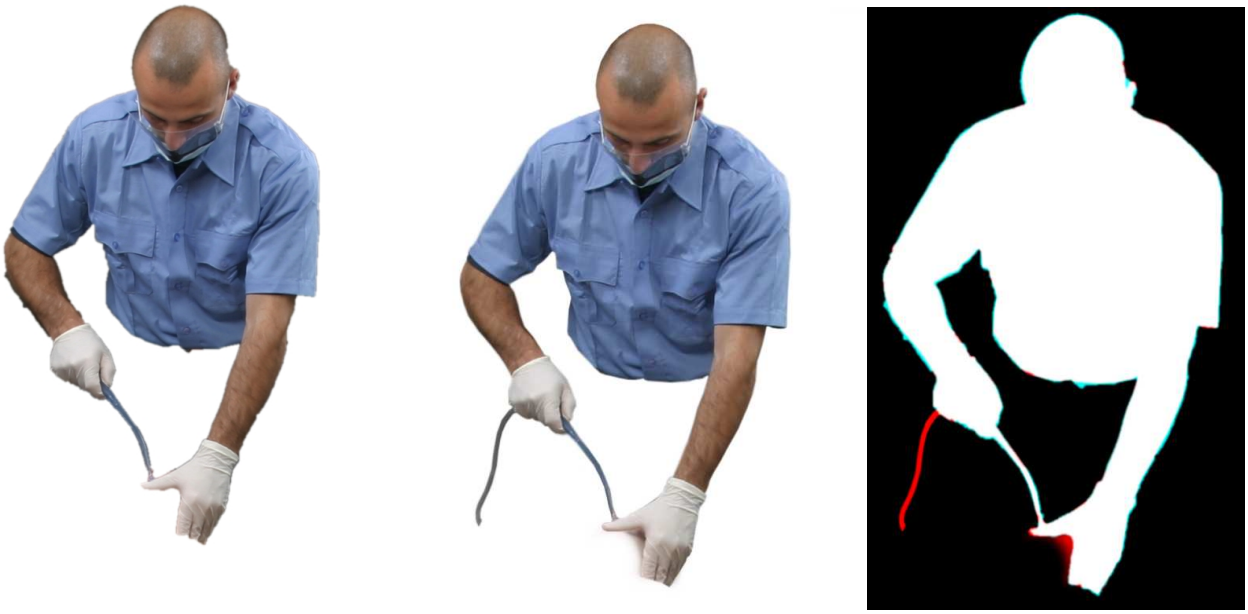


Figure 7.31: $a\ b\ c$ a) Our segmentation. b) LS segmentation. c) Showing segmentation difference. Cyan areas are where our segmentation is larger than LS's segment and red areas are where LS's segmentation is larger than ours.

Figure 7.31 compares our segmentation with that of LS. There are some problems in the segmentation around the right arm, the left ear and the lower part of the shirt. The right arm is problematic because the border of the arm is almost black and therefore blends in with some black lines in the floor. Similarly, the area around the left ear is problematic because the colour of the head is almost the same colour as the floor. The difference between the two segmentations can be seen in Figure 7.31c.

Comments from LS *"Some of the head is missing in the right side and half the tube is missing too. The man's right arm is poorly segmented and a bit of the left arm is missing. Minor problems on the lower part of the shirt."*

7.6.5 Case 5

Figure 7.32 shows the original image for case 5. For this segment we have no segmentation from LS. However, in order to demonstrate that our segmentation tool works on something different from paramedics in blue shirts we show this case. We aim to segment the arms and the respiratory device.

Figure 7.32: Original image for case 5. Image size 1396x1440. We aim to segment arms and the respiratory device.

Figure 7.33 shows three steps in the segmentation process. We see the results after 40s., 1m40s. and 3m.

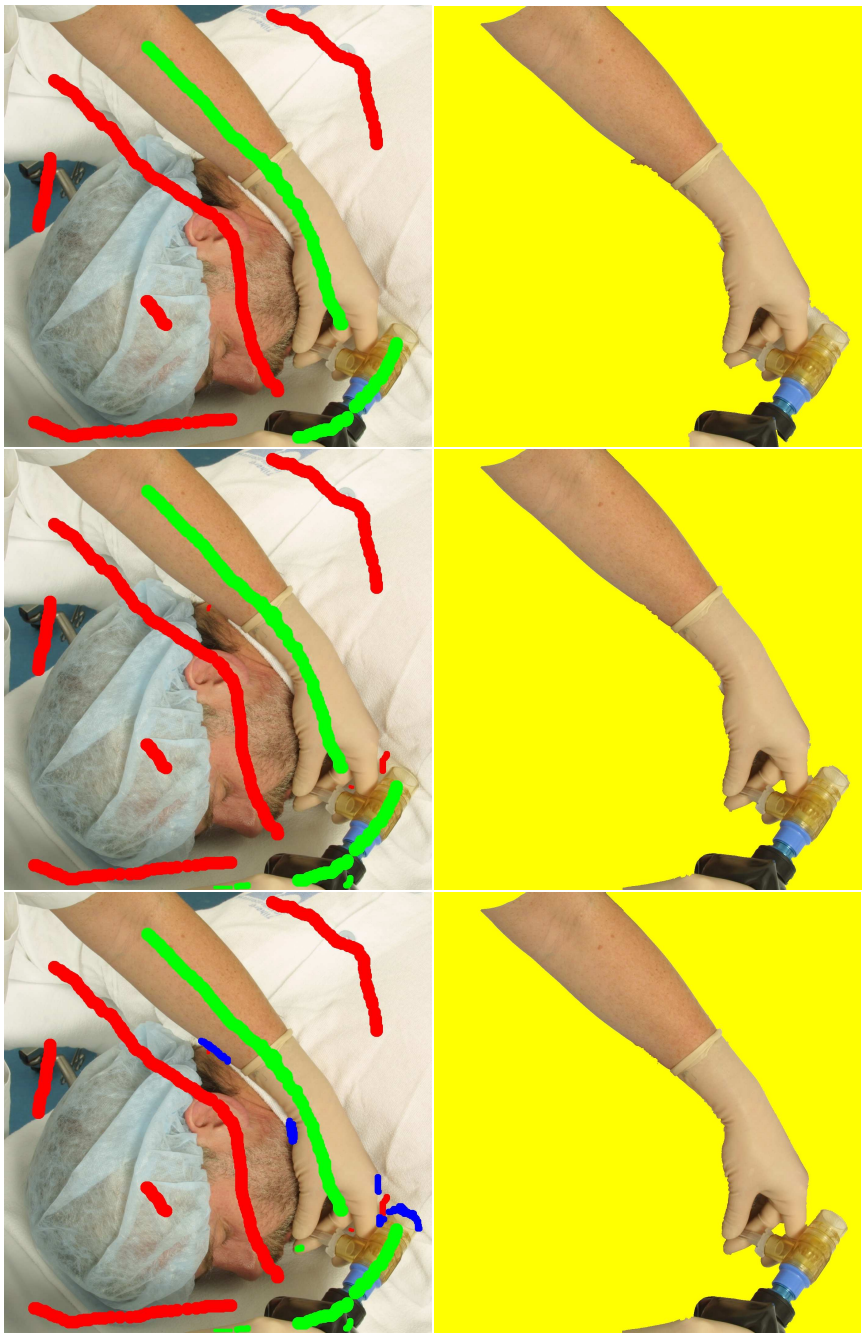


Figure 7.33: $\begin{smallmatrix} a & b \\ c & d \\ e & f \end{smallmatrix}$. Three steps in the segmentation process. a, b) Seeds and result obtained in 40s. c, d) Seeds and result obtained after 1m40s. e, f) After 3m.

Figure 7.34 show a very good segmentation result. The irregularities on the lower part of the arm is caused by the similarity with the patient's skin colour.

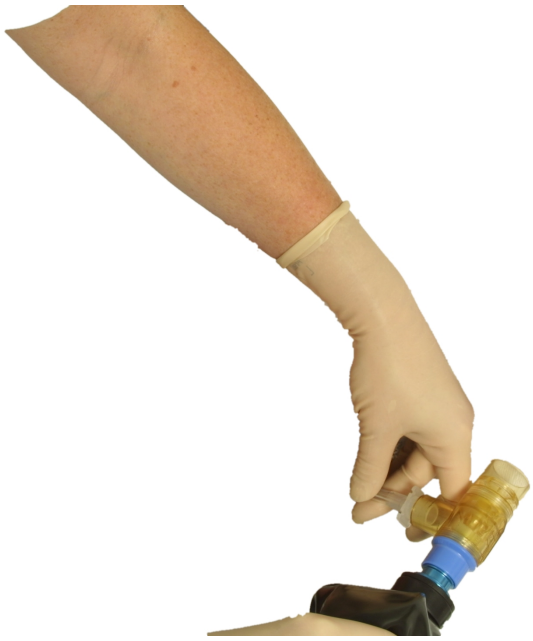


Figure 7.34: Our segmentation.

Comments from LS *"Very good. There are some irregularities on the lower part of the arm and on the glove."*

7.6.6 Difficult Image Example

For the next four examples we will only show the entered seeds and the result. These examples are made to show how our implementation handles difficult tasks such as transparent objects, overlapping colour distributions and thin hair structures.

Figure 7.35 show the segmentation of a transparent mask. The segmentation is

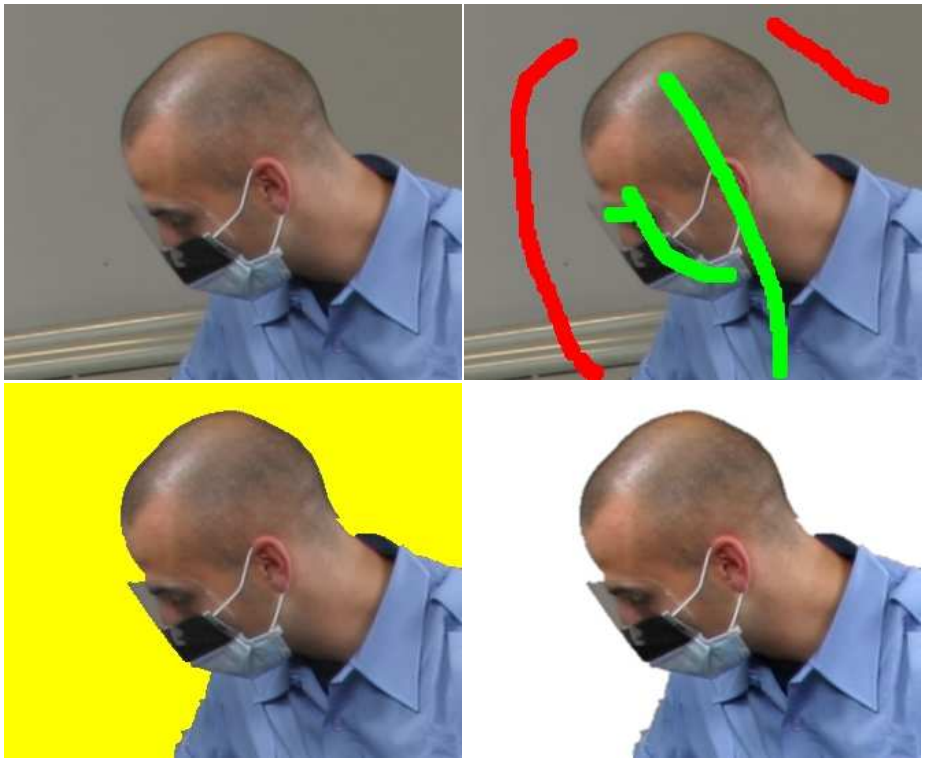


Figure 7.35: $\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}$ Image size 512x980. a) Original image size 334x274 b) Entered seeds c) Final hard segmentation obtained after 30 seconds. d) Final segmentation after blurring the alpha channel with a 5x5 Gaussian filter.

good, but this segment require some aftercare because the transparent mask would look strange on a new background.

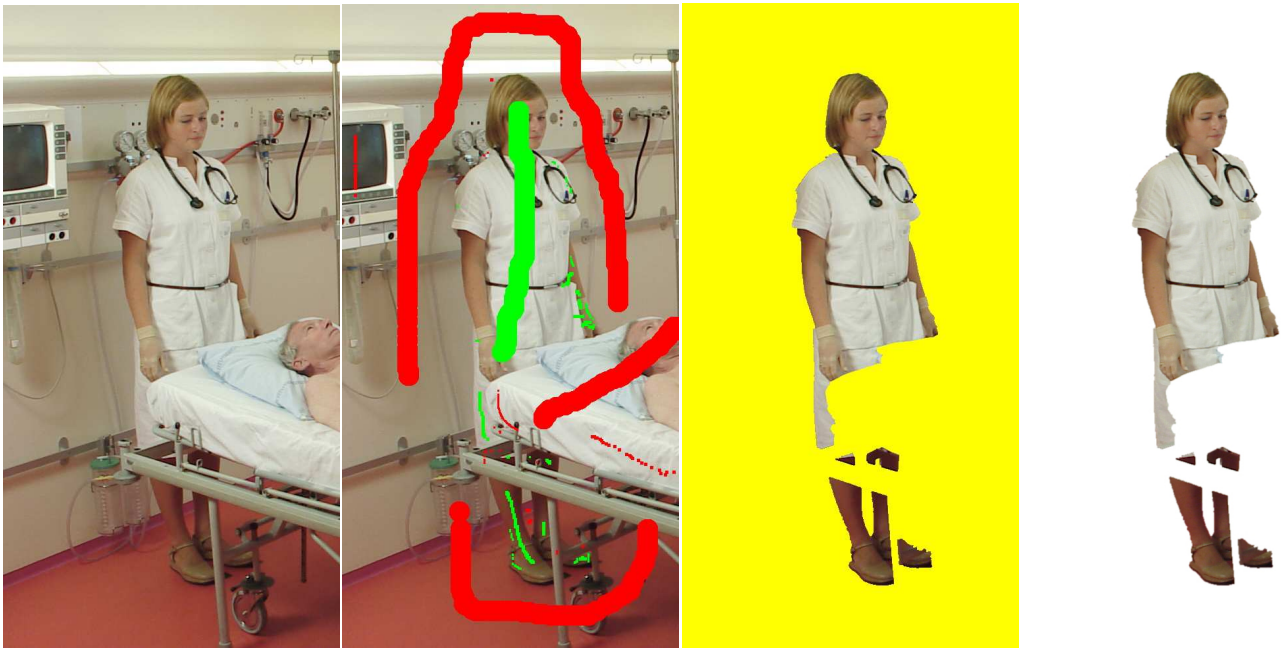


Figure 7.36: $a\ b\ c\ d$ a) Original image size 512x980 b) Entered seeds c) Final hard segmentation obtained after 5 minutes. d) Final segmentation after blurring the alpha channel with a 5x5 Gaussian filter.

The resulting image in Figure 7.36 shows a very good segmentation of a difficult image. The segment has many colours in common with the background. It is, however, possible to segment the nurse without appreciable difficulties though it took 5m of work to obtain a satisfying segment.

Figure 7.37: $\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}$ a) Original image size 1145x1097 b) Entered seeds c) Final hard segmentation obtained after 2m30s. d) Final segmentation after blurring the alpha channel with a 5x5 Gaussian filter.

Figure 7.37 shows another difficult segmentation task. Thin structures like the woman's hair are of course extremely difficult to segment properly. Not surprisingly, the segmentation result shows that the hair is impossible to segment. The segmentation simply cuts of all the loose straws of hair. Even though the segmentation in practise fails one can argue that the visual impression is not too bad.

Figure 7.38: $\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}$ a) Original image size 435x573 b) Entered seeds c) Final hard segmentation obtained after 5m. d) Final segmentation after blurring the alpha channel with a 5x5 Gaussian filter.

The final example in Figure 7.38 shows that colours which are too similar cannot be separated. The hand of the doctor blends almost totally together with the patient. Though obtaining a good result for many parts of the segmentation, we were not able to separate the doctors hand and the syringe from the background.

7.7 Discussion

The above test shows that our method performs well even for many difficult segmentation tasks. There are, however, some examples which are harder to segment than others. Adjacent foreground and background areas of similar colour and shadows often present a problem for our implementation. In many cases very dark shadows fall on the floor and blend together with dark trousers and shoes. These areas are particularly difficult to segment and in many cases it is even difficult for the human eye to distinguish between object and shadow. Because of the light setting, the underside of many objects are in shade and thus have a darker tone than the rest of the object. This phenomena has also presented a problem that often appears when segmenting bright objects like the white gloves of the paramedics. We saw an example of this in case 3.

In the comparison with the segments provided by LS we discovered that LS's segments in general are a few pixels smaller all around the segmentation border. The reason is that when LS manually segment the images they cut into the border of the object. Doing this still produces very realistic looking segments which is the main priority at LS. However, our implementation cannot take visual interpretations into account, but has to rely on the obtained min-cut. This explains the general difference between our segments and LS's.

The local graph has proven to be very important to obtain the necessary quality of the segments. The fact that the local graph can be entered rather unprecise helps the user in fine tuning complex structures more efficiently than by entering more seeds. Furthermore, the local graph often provided the necessary fine tuning properties in areas where the pre-segmentation failed. It was interesting to notice how well the local graph segmented elongated structures, which we predicted would be very difficult. There are, however, some problems connected with the use of the local graph. First of all, local graphs can only be entered on the border of the initial segment which limits the use. Furthermore, the result of the local graph can in some situations be slightly unpredictable which means that the user has to delete it and enter a new. Doing this is very time consuming and slows down the fine tuning process.

Looking at the overall performance in Section 7.6, it is evident that our implementation yields a good initial result in short time. It is possible to fine tune the segmentation to obtain even higher quality. Fine tuning has, however, shown to be the most time consuming part of the segmentation task. Our two possibilities of fine tuning, entering more seeds and the local graph, can produce good results, but in some cases much time could be saved by letting the user enter a hard constraint by editing the boundary directly. A method for this could for example the direct vertex editing tool described by [11]. Therefore, we must conclude that further development of our application should be concentrated on producing a faster and

more predictable fine tuning tool.

Throughout the test we have had LS comment on the results. Their comments are in general positive though they have some reservations with respect to some parts of the segmentation quality. Many of the comments are related to shadowing the segments to make them look more realistic on a new background. Shadowing has, however, not been a priority for us and therefore problems in terms of missing shadows etc. should not affect the overall assessment of our method. Nevertheless, we are of course interested in making an application that is open to the possibility of creating or segmenting shadows. We made a small experiment together with our contact at LS. Given a segmentation result in which we included the existing shadow (see Figure 7.39b) he manually modified the shadow in just 30 seconds to nearly fit their standards. Since it is often easier to include the existing shadows in the segment than to remove it, this tells us that we may already have a shadow solution that to some extend is useful for LS.

Figure 7.39: $a\ b\ c$ a) Input of seeds with the intend to include existing shadows in the segment. b) Segment including existing shadows. c) Manually modified shadow made with starting point in the existing shadows.

As a last finish for all cases we have blurred the alpha channel with a 5x5 Gaussian filter. This have proven to soften the border of the segment, which gives a nice visual effect.

On average LS uses 15 minutes to segment one image. This time includes segmentation, shadowing and quality control. Since we do not have the exact time that an LS operator uses for the segmentation task alone, we cannot perform a direct comparison. The times used to obtain our segmentations are in the area of 3-6 minutes depending of the segment difficulty. We do, however, not always obtain the necessary quality within this time range. Even if we used more time to add additional seeds our implementation does not guarantee the necessary quality. For this reason, our implementation cannot replace LS's current manual segmentation

method completely. Nevertheless, we are confident that our implementation, in its present state, can easily be combined with some manual adjustments to correct the last minor quality problems. We think that this would show to make LS's production more efficient. Furthermore, we believe that further development of our implementation can result in an application that successfully combines segmentation and shadowing.

Chapter 8

Conclusion and Further Work

8.1 Conclusion

In this thesis work we have presented a prototype tool for interactive segmentation of natural images. The tool has been implemented to test the possibility of replacing the manual segmentation method currently used at Laerdal-Sophus A/S who has been our collaborator for this project.

The basic technology for the segmentation method is graph cut segmentation. Because we aimed to provide an interactive segmentation method with fast user feedback we were inspired by [11] to precede the graph cut segmentation by a pre-segmentation of the image. Nodes in the graph represented small segments instead of pixels. As apposed to Li et al. [11] we used a toboggan watershed method to obtain the pre-segmented image. This method proved to be very efficient and provide good pre-segments. The gradient magnitude image used for the pre-segmentation was obtained in a multi-scale framework. We tested different normalisation methods for multi-scale edge detection. From this test we must conclude that normalisation over scale blurred edges too much for our use. This led us to not use any normalisation method. The test showed that using a multi-scale framework resulted in fewer pre-segments which in turn reduced the size of the graph. Constructing the graph based on a pre-segmented image has proven to be essential to obtain an interactive segmentation method with almost instant user feedback.

To obtain the min-cut/max-flow in the graph we used the augmenting path algorithm by Boykov-Kolmogorov. This algorithm proved to efficiently find the min-cut/max-flow in a graph. We have, however, shown that half of all the t-links in the graph can be saturated before running the graph search algorithm. This has, at least for our implementation, increased the speed of the segmentation method significantly without affecting the segmentation output.

For entering seeds, we have tested two methods that deviate from the common way

of entering seeds. For the purpose of entering seeds into elongated structures we tested the applicability of an interactive pathfinder to automatically enter a dense row of seeds into the object. Though, we have not included the method in our final solution the method shows high potential.
We also tried to use subtraction of a background image as a method for automatic entering of background seeds. The idea proved to work but the general benefit was small, that is we did not gain any quality and additionally the segmentation time increased.

We have included two methods for fine tuning the segmentation. First, there is the traditional fine tuning method where the user enters additional seeds in the graph. This method is an indispensable tool to obtain high quality segmentation. After adding seeds we update the graph locally reusing the initially obtained flow. This way of updating the graph has shown to be very fast, but does not take new seeds into account in the colour statistics. Therefore we must conclude that when using the proposed local updating method it is important to have a good initial input of seeds.
As a second fine tuning method we have proposed a local graph with automatic seed point selection. By using local image information the local graph has proven to optimise areas where the global segmentation fails. The fact that the local graph is constructed on pixel basis enables it to correct errors related to the pre-segmentation step. Though in many cases being indispensable to obtain a high quality segmentation, the local graph has for some cases shown to be somewhat unpredictable. This has unfortunately shown to increase the total segmentation time.

With the aim to create a soft transition on the segment border we tried using Bayesian matting. The method showed potential in the direction of producing high quality mattes even on very complex borders. Bayesian matting, however, showed to be very slow. Moreover, to obtain good results the method required individual settings of the program parameters for each image. These facts led us to omit Bayesian matting from the final solution. Instead, we propose to use a simple, but fast, Gaussian filtering of the alpha channel. Though simple, Gaussian filtering had a good visual effect on the segments, and because LS in general have very simple segment borders this method is often sufficient.

We have made a C++ implementation of the segmentation method along with a GUI in the framework of LS's image editing program Picture Factory. The combination of the two allowed us to test our implementation in an environment very close to the environment that a potential operator at LS would be presented with. Therefore, we conclude that the final overall performance test yields a good impression of the potential of our segmentation tool related to the segmentation task at LS.
We have conducted a thorough time study of our implementation. Their are two

mains things to conclude. First, we recommend that pre-segmentation is done off-line as a pre-processing step, prior to the actual segmentation. Second, we have shown that the feed-back time for our implementation allows for an interactive use of the tool.
To evaluate the overall performance of the final segmentation tool, we have compared our segmentation results with segmentations done manually by LS. These results show that we were able to produce segments very close to the quality of LS and in a few cases perform better than LS. The results were obtained in the range of 3-6 minutes per image. Our implementation does, however, not guarantee to find the necessary quality even when using more time to enter additional seeds.

It has been outside the scope of this thesis to create an actual tool for shadowing of segments. In the meanwhile, it has become clear throughout our work that shadows are necessary for LS before the produced segments can be used in their production. With help from one of our LS contacts it was shown that our implementation in its current state can already be used to help creation of segment shadows based on segmentation of existing shadows from the original images.

Finally, we conclude that LS can benefit from semi-automation of their image manipulation process. The tool we have presented in this thesis can provide the foundation for this semi-automation. However, in order to replace the current segmentation method at LS our implementation needs further development.

8.2 Further Work

Fine tuning The fine tuning step of our implementation showed to be the most time consuming part of the segmentation. For some types of initial segmentation errors it would have been easier and less time consuming to correct the segmentation manually. However, we have seen from [11] that it is possible to make a fine tuning tool that combines entering of hard constraints with direct editing of the segment boundary. This makes the input of the hard constraint easier and less time consuming. In the further development of our implementation a tool similar to that of [11] would be of high value.

Local graphs If the method is developed further, we are confident that it has the potential of being a good stand-alone interactive boundary-based segmentation method. The idea is that the user draws on the border of the desired segment using a wide brush. For each brush stroke a new local graph is entered into the image and a local min-cut/max-flow is found for each local graph. Combining all entered local graphs should eventually yield the segment boundary.

Better distance measure for alpha estimation To increase the quality of both Bayesian matting and local alpha estimation the distance measure in the unknown area should be more precise. This could be done by using a more advanced distance measure such as level set methods.

Bayesian matting using dynamic programming To increase the speed of the alpha matting algorithm it would be interesting to explore whether the problem can be solved using dynamic programming (DP). Many of the operations in the alpha matting algorithm are highly dependent on previous estimates. Furthermore, many operations, like sampling and clustering, are repeated for the same foreground and background samples. The algorithm therefore opens for a potential use of DP.

Colour space transformation Throughout this thesis we have only used the RGB colour domain. However, other colour domains, such as the HSI domain, have different properties that might improve parts of our implementation. Especially, subtraction of a background image for shadow detection in the HSI colour domain would be interesting to investigate.

Removal of irregularities along segment borders Many of the LS's segments have been further enhanced by removing irregularities in the direction along the border. This has a positive effect on how realistic a segment is perceived and therefore could be subject to further work. We have used a simple Gaussian filter on the alpha channel which proved a good visual effect despite its simplicity. We believe that experimenting with other noise removal filters along the segment border can

have a similar positive effect, with only little effort. These filters could e.g. be median filters or standard morphological operations.

Bibliography

[1] Y. Boykov and M.-P. Jolly. Interactive graph cuts for optimal boundary and region segmentation of objects in n-d images. *In Proc. IEEE Int. Conf. on Computer Vision*, 2001.

[2] Yuri Boykov and Vladimir Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(9):1124–1137, 2004.

[3] Yung-Yu Chuang, Brian Curless, David H. Salesin, and Richard Szeliski. A bayesian approach to digital matting. In *Proceedings of IEEE CVPR 2001*, volume 2, pages 264–271. IEEE Computer Society, December 2001. *http://grail.cs.washington.edu/projects/digital-matting/image-matting/*.

[4] Cormen, Leiserson, and Rivest. *Introduction to Algorithms*. MIT Press, Cambridge Mass., 1990.

[5] Erik B. Dam and Martin Lillholm. Interactive route measuring. Technical report, DTU, 2001.

[6] R. C. Gonzalez and R. E. Woods. *Digital Image Processing*. Addison-Wesley, Reading, MA, 2002.

[7] B.K.P. Horn and B.G. Schunck. Determining optical flow. *AI*, 17:185–203, 1981.

[8] Fredrik Karlsson. Matting of natural image sequences using bayesian statistics. Master's thesis, Department of Science and Technology Institutionen för teknik och naturvetenskap Linköpings Universitet, 2004.

[9] Vladimir Kolmogorov and Ramin Zabih. What energy functions can be minimizedvia graph cuts?. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(2):147–159, 2004.

[10] S. Li. *Markov Random Field Modeling in Computer Vision.* Springer, 1995.

[11] Yin Li, Jian Sun, Chi-Keung Tang, and Heung-Yeung Shum. Lazy snapping. *ACM Trans. Graph.*, 23(3):303–308, 2004.

[12] Tony Lindeberg. Scale-space: A framework for handling image structures at multiple scales. In *Proc. CERN School of Computing*, Egmond aan Zee, Netherland, September 1996.

[13] Tony Lindeberg. Feature detection with automatic scale selection. *International Journal of Computer Vision*, 30(2):79–116, 1998.

[14] Eric N. Mortensen. Vision-assisted image editing. 1999.

[15] Eric N. Mortensen and William A. Barrett. Toboggan-based intelligent scissors with a four parameter edge model. *In Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR 99)*, pages 452–458, 1999.

[16] Jakob Nielsen and Martin Wallengren Nilsson. Segmentering ved hjælp af grafsnit. Technical report, IT University of Copenhagen, 2003.

[17] P. Perona and J. Malik. Scale-space and edge detection using anisotropic diffusion. *IEEE Trans. Pattern Anal. Mach. Intell.*, 12(7):629–639, 1990.

[18] Peter Pilgaard Rasmussen and Ditte-Maria Christensen. Billedsegmentering ved hjælp af grafsnit. Master's thesis, IT University of Copenhagen, 2004.

[19] Carsten Rother, Vladimir Kolmogorov, and Andrew Blake. Grabcut: interactive foreground extraction using iterated graph cuts. *In Proceedings of ACM SIGGRAPH.*, 2004.

[20] Andrea Schenk, Guido P. M. Prause, and Heinz-Otto Peitgen. Efficient semi-automatic segmentation of 3d objects in medical images. In *MICCAI '00: Proceedings of the Third International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 186–195. Springer-Verlag, 2000.

[21] Jian Sun, Jiaya Jia, Chi-Keung Tang, and Heung-Yeung Shum. Poisson matting. *ACM Trans. Graph.*, 23(3):315–321, 2004.

[22] Adobe Creative Team. *Adobe Photoshop CS Classroom in a Book*. Adobe Press, 1 edition, 2003.

[23] Wikipedia the free encyclopedia. *http://en.wikipedia.org/wiki/Simulated_annealing*.

[24] Chih-Hao Tsai. Test report: Millisecond resolution timing with visual c++ and windows 95/98. 1998. *http://technology.chtsai.org/w98timer/*.

[25] Luc Vincent and Pierre Soille. Watersheds in digital spaces: An efficient algorithm based on immersion simulations. *IEEE Trans. Pattern Anal. Mach. Intell.*, 13(6):583–598, 1991.

[26] Eric W. Weisstein. *K-Means Clustering Algorithm*. MathWorld–A Wolfram Web Resource.
http://mathworld.wolfram.com/K-MeansClusteringAlgorithm.html.

[27] C. Xu, D. L. Pham, and J. L. Prince. Medical image segmentation using deformable models. In *Handbook of Medical Imaging – Volume 2: Medical Image Processing and Analysis*, pages 129–174. SPIE Press, 2000.