Implementing the "GrabCut" Segmentation Technique as a Plug-in for the GIMP

Submitted in partial fulfilment
of the requirements of the degree
Bachelor of Science (Honours)
of Rhodes University

Matthew Marsh

7th November 2005

Abstract

The open source community require a segmentation tool that is fast and easy to use. The GIMP has built-in segmentation tools, but under some circumstances these tools perform badly. The GIMP allows extra functionality to be added to it by writing plug-ins. The aim of this project is to implement the "GrabCut" segmentation technique as a plug-in for the GIMP.

"GrabCut" is an innovative segmentation technique that uses both region and boundary information in order to perform segmentation. It uses graphs and a Min-Cut/Max-Flow algorithm to segment the graphs, and in doing so perform segmentation. "GrabCut" also includes a border matting technique.

We have implemented three variations on the "GrabCut" segmentation algorithm as plug-ins for the GIMP. The first method works on greyscale images and uses histograms to model region data. The second method works on colour images and uses Gaussian Mixture Models to model region data. This method did not produce good results, so we developed a new method of modeling region data. This method uses colour histograms to model region data. The greyscale and colour histogram segmentation techniques produce good results quickly.

The border matting technique suggested by the authors of the "GrabCut" paper has been implemented, but due to time constraints the energy function which produces the good matte is minimized by a genetic algorithm. The border matting implementation significantly improves the overall appearance of segmented images.

Acknowledgements

I would like to acknowledge my supervisor Shaun Bangay for his endless help, especially with regards to the maths - which has come back to haunt me from first year! Without him I would never have understood Gaussian Mixture Models. Thanks go to Adele Lobb for her support as a supervisor. I would also like to thank my parents and sister for their encouragement and support throughout the year. Without them I would have given up ages ago. Special thanks go to Mike Horne for his endless patience in debugging my error-full programs. Special thanks also go to Rhodes University for the Dean of Research bursary which has enabled me to study for a Computer Science Honours degree.

Contents

List of Figures

Chapter 1

Introduction

1.1 Problem Statement

The open source community require a quick, easy to use segmentation tool. This tool is to be implemented as a plug-in for the GNU Image Manipulation Program (the GIMP). The GIMP already contains various segmentation tools, however these tools are not always sufficient to produce good segmentation results. The lack of an efficient segmentation tool poses a problem to all image editors using the GIMP.

Recent research into using graph cut techniques for the purpose of image segmentation has led to the development of the "GrabCut" segmentation technique. The authors of the "GrabCut" paper, Rother et al. [2004], claim that their segmentation technique addresses the problem of efficient image segmentation. This project aims to implement the "GrabCut" technique as a plug-in for GIMP, and in doing so test the validity of their results.

1.2 Background on image segmentation

Image editing requires the ability to quickly and easily extract foreground objects from digital images. A foreground object refers to any object of interest in an image. The background of the image refers to all pixels in the image that are not part of the foreground object.

The process of separating an image into foreground and background parts is known as image segmentation. Humans have the ability to recognise objects, and can perform image segmentation by just identifying an object in an image. Children going to preschool learn how to segment images by cutting an object out of a picture along a dotted line. We find image segmentation to be an extremely simple task, but it is definitely not a trivial task for computers.

In order for computers to perform image segmentation the algorithm performing the segmentation needs to use information encapsulated in the digital image to calculate the best segmentation. Computers have no means of intelligently recognising objects, and so many different methods have been developed in order to segment images. The segmentation process is based on various features found in the image. This might be colour information that is used to create histograms, or information about the pixels that indicate edges or boundaries or texture information [Ballard and Brown, 1982].

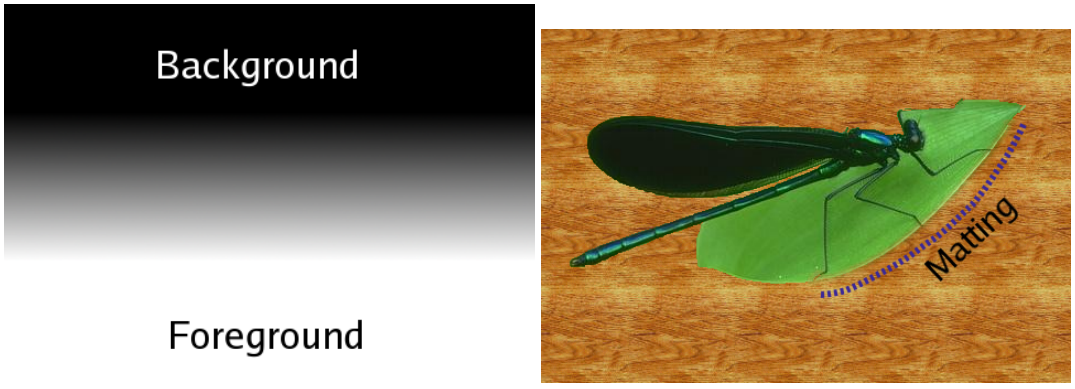
Edge detection and region detection methods are most commonly used to segment images. Boundary information is the edge information in an image which can be gained by noting the difference between the colours of nearby pixels. A large difference in colour will indicate an edge. A region is the area enclosed by an edge, and the colour of neighbouring pixels in a region are similar.

Region and edge detection methods are sufficient for segmenting images with clear boundaries. If the edges of the object to be segmented are fuzzy, pixels from the background can be incorrectly segmented as part of the foreground, and vice versa. This causes unsatisfactory segmentation results. In order to produce the best segmentation of an image with fuzzy boundaries it is necessary to perform matting. Matting is a technique which smooths the boundaries of segmented objects, and makes their appearance more natural. Figure 5.6 in section 5 is a good example illustrating how matting improves the appearance of segmented objects.

Pixels in digital images have individual alpha values. An alpha value is the opacity of a pixel, and is allowed to vary between 0 and 1. Thus a pixel with an alpha value of 0 will be totally transparent, and an alpha value of 1 will cause the pixel to be totally opaque. Without matting, a segmented object will have all its pixels set to an alpha value of 1. Matting is the process which smooths the boundaries of objects by setting alpha values along the objects boundary to change smoothly from 0 to 1. Image (a) in Figure 1.1 shows how matting smooths the transition from alpha values of 0 at the background to alpha values of 1 at the foreground. In this image black represents alpha values of 0 and lighter shades of gray represent higher values of alpha. Image (b) in Figure 1.1 shows how the matting process improves the appearance of a segmented object. In this image only the bottom right side of the leaf has been matted.

1.3 Approach

The aim of this project is to provide the open source community with a segmentation tool that is quick and easy to use. In order to pursue this goal we aim to implement a segmentation tool as a



(a) A smooth change in alpha

(b) Matting applied to part of an object

Figure 1.1: Border matting

plug-in for the GIMP.

The GIMP is an open source image editing package that requires an easy to use segmentation tool that produces good results. It already contains a magic wand segmentation tool and an intelligent scissors segmentation tool, but as the results of this project show, these tools are often cumbersome and perform badly under specific conditions (see section 5.4).

The GIMP is designed in such a way as to allow developers to add extra functionality to it by writing plug-ins. Plug-ins for the GIMP can use the GIMP API to gain access to all the built in tools, objects and methods. This allows the programmer to make use of the existing image manipulation framework in the GIMP, and so abstract from any low level image editing code.

There are numerous segmentation techniques which could be implemented to meet the goal of this project. The "GrabCut" segmentation technique has been chosen for the following reasons:

- "Grab-Cut" is a new innovative technique that has not been implemented in any widely used image editing packages to the best of our knowledge. Thresholding and Live-Wire boundary tools like intelligent scissors have been thoroughly tested and implemented in most image editing packages. Implementing "GrabCut" as a plug-in for the GIMP will test its viability as a segmentation tool, and test the claims made by Rother et al. [2004].

- "Grab-Cut" includes a border matting technique which is useful when segmenting images. All the other image editing approaches do not use matting. Including a matting technique

to use in conjunction with image segmentation will provide image editors with extra functionality, and the ability to produce better segmentation results.

- "Grab-Cut" makes use of both region and boundary information. The most widely used segmentation tools, Snakes and Live-Wire boundaries, do not adopt this approach.

1.4 Document Structure

The rest of the thesis is structured as follows:

Chapter 2 describes work related to image segmentation and matting, and the various approaches that have been taken towards solving these problems.

Chapter 3 describes how "GrabCut" works, and three techniques which have been used to model region data, namely greyscale histograms, Gaussian Mixture Models and colour histograms. It also details the way "GrabCut" performs border matting.

Chapter 4 discusses the details relating to the implementation of the "GrabCut" plug-in for the GIMP.

Chapter 5 highlights various results found after implementing "GrabCut". Results are divided into an image segmentation section and a matting section.

Chapter 6 provides a conclusion to this thesis

Chapter 2

Related work

There are numerous solutions to the problem of solving for the best segmentation of an image. These range from very simple thresholding methods to complex graph cut methods. All of these techniques make use of information encapsulated in the image to perform segmentation, with the majority of them making use of either region or boundary information.

The most simple method of segmentation is thresholding. Thresholding uses region information in the image to find areas of similar colour or intensity. More advanced techniques like Snakes and Live-Wire boundaries make use of edge detection methods.

Snakes use boundary information in the image to detect edges, and wrap around the object of interest. An energy function is created from the boundary information as well as certain constraints. This energy function is minimized in order to obtain the best segmentation.

Live-Wire boundaries also use boundary information to detect edges in the image. A graph is built in which the edge weights in the graph are set by edge information in the image. A 2D graph search is used to find the optimal segmentation.

The latest segmentation techniques make use of region and boundary information. “GrabCut” is such a technique. Recent research into using graph cuts as a form of energy minimization has led to the “GrabCut” segmentation technique. “GrabCut” uses a graph to represent an image, and segments this graph by using a Min-Cut/Max-Flow algorithm. “GrabCut” also makes use of a matting technique for regions that do not have clear boundaries.

There are several matting techniques which have also been developed. Probabilistic matting uses Gaussians to model colour information along the area to be matted. Once a matte is created no user editing is possible, and so this technique is not interactive.

Poisson matting works directly on the alpha matte, and creates it by using Poisson equations. This approach allows the matte to be edited, and is therefore interactive.

Work relating to the "Grab-Cut" algorithm can be divided into two main categories, namely image segmentation and matting. For this reason the chapter has been divided into two sections reviewing the literature relating to:

1. Image Segmentation, and in particular "Grab-Cut", as well as some of the energy minimization techniques that make use of graph-cut.

2. Matting - The different techniques that can be used to matte an image.

2.1 Image Segmentation

2.1.1 Thresholding

Thresholding is a very simple form of segmentation. Segmentation is performed by defining a threshold, which will most often be an intensity value in the image. Every pixel in the image is compared with this threshold, and if the pixel's intensity is greater than the threshold value it will be marked as foreground, and if it is less than the threshold as background. Other forms of thresholding exist where the threshold is allowed to vary across the image.

Thresholding is a primitive technique, and will only work for very simple segmentation tasks. This is because complex foreground objects will often contain pixels which have intensity values that also lie in the background. Under these circumstances pixels which are part of the background will be incorrectly segmented as part of the foreground, and pixels which should be segmented as foreground will be segmented as background.

Most graphics packages come with some form of thresholding segmentation tool. "Magic Wand" is such a tool, and is included in Adobe Photoshop 7. With this tool the user will select a seed or multiple seed pixels and set a tolerance level. The segmentation is performed by testing all pixels against the seed pixels, and evaluating if they lie within the set tolerance level [INCORP, 2002]. This technique is easy to use, but the results are most often unsatisfactory and finding the correct tolerance level and seed pixels can be cumbersome, and sometimes even impossible [Rother et al., 2004].

2.1.2 Snakes / Energy Minimizing Splines

Snakes are energy minimizing splines that are initialized to roughly follow the border of the object to be segmented in an image. They are designed to be interactive, in that the user must give clues as to where the boundaries of the object are. Snakes are guided by external and internal

constraints. An energy function is defined by these constraints, and Snakes are used to minimize the energy function and so trace the contour or boundary of the object to be segmented [Kass et al., 1987]. Snakes are so called due to the wriggling motion they undergo while minimizing their energy functions.

Snakes work on the assumption that edges are found not only by looking at the local gradient, but also at the long range distribution of the gradient. The gradient information found in an image is incorporated into the external constraints. Internal constraints refer to the curvature of the Snake. These constraints are used to produce a smooth contour that conforms to the object boundary.

The classic implementation of snakes by Kass et al. [1987] allows the problem to be reduced to a matrix form. However this puts constraints on the energy functions. Davison et al. [2000] propose a less complicated form of the energy functions, and energy minimization is carried out by adjusting individual vertexes on the snakes. This allows for a greater range of energy functions, and the addition of internal energy functions like area and symmetry terms without complicating the minimization process as would be the case with the classic implementation.

2.1.3 Live-Wire Boundaries

This technique uses dynamic programming to solve a 2-D graph searching problem and, in doing so, detect boundaries in an image. For this technique a 2-D graph is created, where pixels in the image are represented by nodes in the graph. The edge weights between nodes are defined by a cost function that depend on features found in the image. The goal of the 2-D graph search is to find the minimum cost path between a start and goal node.

1. Mortensen and Barrett [1995] have developed an approach to creating an interactive tool called Intelligent Scissors that uses a live-wire boundary to perform segmentation. When the user moves the mouse near a boundary the live-wire snaps to this boundary. This algorithm greatly reduces sensitivity to noise, and selects the mathematically optimal boundaries nearby. It also incorporates on the fly training so that the live-wire snaps to the current type of edge being followed rather than just the strongest one in the neighbourhood.

2. Mortensen and Barrett wrote another paper which improves on their previous approach. This approach entails over-segmenting the image into regions using a technique called tobogganing. Tobogganing creates a region-based graph rather than a pixel-based graph. This results in a graph with fewer nodes than a pixel-based graph and so provides faster graph searches allowing quicker results and better user interaction. As the user moves the

mouse near boundaries in the image a collection of good segmentations are displayed. This technique allows the user to interactively select the best segmentation as they provide seed points along the image boundary [Mortensen and Barrett, 1999].

2.1.4 Graph Cut Techniques

2.1.4.1 Graph Cuts as a energy minimization technique

The use of graphs to solve energy minimization problems has become more and more popular in the context of low level computer vision [Boykov and Kolmogorov, 2004]. Many problems in computer vision can be reformulated as an energy minimization problem. Energy minimization has in the past been computed by using dynamic programming (which only works in very simple cases) and simulated annealing (which is very slow). Recently there have been many approaches to energy minimization by using graph-cut techniques, and in most cases each graph has had to be specially created for solving a specific energy minimization problem [Kolmogorov and Zabih, 2004]. Kolmogorov and Zabih [2004] look at the types of functions that can be minimized by graph cuts and they give a general construction of the graphs that can solve different classes of problems.

All segmentation techniques using graphs use some form of a graph-cut algorithm to segment the graph into two regions, and in doing so minimize the energy. Boykov and Kolmogorov [2004] provide a comparison between current Min-Cut/Max-Flow algorithms with regards to their efficiency. This paper includes Goldberg-Tarjan style "Push-Relabel" methods and Ford-Fulkerson style "Augmenting Path" methods. It also introduces a new algorithm which they claim works several times faster than all other known methods in most cases. The paper uses energy minimization techniques in the context of image restoration, stereo and image segmentation in order to compare the speeds of the different methods

2.1.4.2 Graph Cuts for image segmentation

- Boykov and Jolly [2001] introduce a segmentation technique that uses a graph to represent the image, and a Min-Cut/Max-Flow algorithm to segment the graph. Pixels in the image are represented by nodes on the graph. The edge weights on the graph are defined by a cost function, which is defined by region and boundary information found in the image. A Min-Cut/Max-Flow algorithm is used to segment the image by minimizing the cost function. This technique uses an intensity histogram to store region information found in the image, and only works on greyscale images.

- Li et al. [2004] introduce an interactive means of segmentation based on graph minimization techniques very similar to Boykov and Jolly [2001]. It involves two major steps. The first step is the object marking stage in which certain pixels are marked as either background or foreground. These are hard constraints which have to be met during the segmentation process. Graph cuts are used together with an over-segmentation technique which greatly increases the speed of segmentation, and can provide segmentation results quickly to the users. The second step is a boundary editing step in which individual vertexes on the segmentation can be moved around until the user is satisfied.

- Interactive segmentation is becoming more and more popular and it is far preferred over fully automatic segmentation methods that are never perfect [Boykov and Jolly, 2001]. There have however been some methods that try to perform automatic segmentation. Blake et al. [2004] describe a method of image segmentation that operates with very few hints from the user. Most image segmentation techniques use parameters set by the user. In this case an algorithm is used to learn parameters from the image data, and then perform the segmentation based on these parameters. The segmentation technique proposed by Boykov and Jolly [2001] is used, but the parameters needed for segmentation are learned by a pseudo-likelihood algorithm. A database of images with the correctly segmented results is used to test the approach. The percentage error using interactive approaches was considerably lower than this approach [Blake et al., 2004].

2.1.4.3 "Grab-Cut"

Rother et al. [2004] present an innovative way of segmenting an image into a foreground and background region. The image to be segmented is represented by a graph which is constructed so that a minimum cost cut on the graph will produce the best segmentation of the image. Pixels in the image are represented by nodes on the graph. The edge weights of the graph are defined by a cost function / energy function. This cost function depends on boundary and region information found in the image. A Min-Cut/Max-Flow technique is used to segment the graph.

"Grab-Cut" uses the segmentation technique described by Boykov and Jolly [2001], but extends the technique to work on colour images and introduces incomplete labeling (the user only has to specify hard constraints for either the background or foreground). "GrabCut" also provides a border matting feature for use in areas of the image that are "fuzzy" and don't have clear boundaries. The algorithm is interactive and allows the user to relabel pixels after the initial segmentation is performed.

Table 2.1: Comparison between segmentation techniques

Segmentation Technique	Information used in Segmentation	Method of Segmentation
Thresholding	Colour	Simple Comparison Between Pixels And The Threshold
Live Wire	Edge	2D Graph Search Using Dynamic Programming
Snakes	Edge	Energy Minimization of Internal and External Functions
Graph cut	Colour and Edge	Max-Flow/Min-Cut Algorithm For Energy Minimization

Boykov and Jolly [2001] use an intensity histogram to model region information contained in the image. The "GrabCut" technique uses Gaussian Mixture Models to model region information found in the image.

2.1.5 Summary

Table 2.1 shows a comparison between the various segmentation techniques with regard to the information they use to perform the segmentation and the method that is employed in order to perform the segmentation.

2.2 Matting

The boundaries of objects in images are sometimes well defined and can easily be traced with some sort of edge detection mechanism. However natural or organic objects often have very intricate boundaries. Hair is very hard to segment and to trace with standard edge detection methods. Water spray, smoke and leaves are also very difficult to segment using any of the standard techniques. The difficulty arises because of the limited resolution of cameras. If a pixel in the camera receives light from more than one source (as will happen with very fine objects) the pixel will take on a colour that is a mixture of the colour from the two objects. The amount that a colour contributes to the overall colour of a pixel is referred to as the alpha value of that colour. Matting techniques segment images by calculating the colour and alpha value of foreground pixels [Ruzon and Tomasi, 2000]. The problem of matting is inherently under constrained, since for each pixel that is mixed, there are an infinite number of combinations of the object's colour and alpha value [Chuang et al., 2001].

Most matting techniques require a tri-map of the image before they can perform matting. The tri-map specifies which regions of the image are background, foreground, and the "unknown region". The unknown region is where matting must take place.

2.2.1 Probabilistic Alpha Estimation Using Colour Statistics

Ruzon and Tomasi [2000] describe a matting technique of using two un-orientated Gaussians distributions to model the colours in the foreground and background region. All pixels in the unknown region are then said to fall within the colour space between these two Gaussians. As the pixels in the unknown region are a mixture of the colours from the foreground and background model, an interpolation of the two Gaussians is necessary. The alpha value of pixels in the unknown region can then be determined by creating an interpolated probability distribution that gives the maximum probability for the value of the unknown pixel.

The technique proposed by Chuang et al. [2001] is very similar to that of Ruzon and Tomasi [2000]. Gaussian probability distributions are also built for the foreground and background. However the technique differs in a number of ways. Orientated Gaussian distributions are used which they claim better models colour distributions. Once colour values for pixels in the unknown region are computed, they are used in calculating un-computed nearby values. This is referred to as a "sliding window" for neighbourhood definitions. They also formulate the problem of calculating the matte values as a Bayesian Framework. This means that the algorithm uses Baye's theorem to calculate the most likely values for a pixels alpha value, and the foreground and background colours of the pixel.

Rother et al. [2004] use a technique very similar to Ruzon and Tomasi [2000] in order to perform probabilistic matting. A smooth alpha profile is assumed, and dynamic programming is used to minimize an energy function and so calculate the alpha profile. Gaussian parameters for the foreground and background are estimated by using near-by known values. Once the colour of a foreground pixel is calculated, the closest match to that colour is chosen from the corresponding foreground Gaussian Mixture Model and used. This is used to reduce colours bleeding in from the background.

2.2.2 Poisson Matting

Ruzon and Tomasi [2000] and Chuang et al. [2001] use colour samples from the background and foreground regions to perform the matting using statistical information about these colour samples. These approaches are not interactive, and once the tri-map is specified no further editing of the matte can be performed. The Poisson matting approach by Sun et al. [2004] works directly on the gradient of the matte. This solves the effects of the wrong classification of colours in complex scenes in the statistical methods. The gradient of the matte is estimated in the image, and then recreated by solving Poisson equations. A smooth gradient is assumed between the

Table 2.2: Comparison between matting techniques

Matting Technique	Method Used	Interactive
Probabilistic Alpha Estimation	Colour Statistics are used to create Gaussians	No
Poisson Matting	The Alpha matte is modeled using Poisson equations	Yes

foreground and background. This technique solves for a global matte gradient, and then allows the user to edit the gradient with various tools in a local manner. This technique produces better results than Bayesian techniques.

Perez et al. [2003] discuss how images can be seamlessly blended together, as well as how portions of an image can be deformed and seamlessly integrated, using Poisson equations.

2.2.3 Summary

Table 2.2 show a comparison between the different matting techniques with regards to the method used to perform the matting, and if the technique is interactive.

2.3 Conclusion

There are many different techniques that perform segmentation. The main categories being Thresholding, Snakes, Live-Wire Boundaries and Graph cut.

Thresholding is a very simple method and not suitable for complex segmentation.

Snakes and Live-Wire boundaries use edge information in the image to perform segmentation. They are both interactive, but do not work well for objects which don't have well defined boundaries. Objects that have intricate borders are very time consuming to segment using purely edge detection methods. This is because so many clues have to be given.

Graph cut techniques use colour and edge information to perform segmentation. This allows intricate borders to be segmented far simply than with tools like intelligent scissors.

With the new Max-Flow/Min-Cut algorithm the energy minimization process is fast enough to provide an interactive method of segmentation using graph cuts. "Grab-Cut" is an efficient and powerful graph cut technique, and includes a border matting feature for badly defined object boundaries.

Chapter 3

Design

This chapter describes the way image segmentation and matting work using "GrabCut". It also provides three different ways in which region information can be modeled.

Section 3.1 gives an overview of how the "GrabCut" algorithm works and how graph cuts are used to perform image segmentation. This section is followed by section 3.2 which describes three methods that are used to model region information contained in images. The first method is for greyscale images and the second and third are for colour images. Each method section contains a description of the method, the way region data is modeled, and the formulae needed to set up edge weights in the graph. Section 3.3 provides an explanation of border matting and how an energy function is minimized in order to produce the best matte.

3.1 How "GrabCut" Works

"GrabCut" is an image segmentation technique that uses graph cuts to perform segmentation. Like most segmentation techniques "GrabCut" uses information encapsulated in the image. Most segmentation techniques make use of either edge information or region information in the image. "GrabCut" makes use of both edge and region information. This information is used to create an energy function which, when minimized, produces the best segmentation.

The "GrabCut" segmentation process can be broken down into four stages. Figure 3.1 outlines this process.

1. In order for segmentation to take place the "GrabCut" algorithm needs to have some idea as to which parts of the image are foreground and which parts are background. This can be done by labelling pixels as either foreground or background. Image 1 in figure 3.1

shows how the top left pixel has been labelled as background and the bottom right pixel as foreground. This first stage of labelling pixels is known as the clue marking stage. All pixels labelled in this stage may not change their labelling later on during segmentation, and so these labels are hard constraints.

2. A graph is created, where nodes in the graph represent pixels in the image. In addition to all the pixel nodes, two special nodes are also created. These are the Sink and Source nodes. Every pixel node in the graph is connected to the Source and Sink node. The Source node represents the foreground of the image, and the Sink node the background. In image 2 in 3.1 the F node is the foreground or Source node, and the B node is the background or Sink node. The weights between pixel nodes in the graph are determined by edge information in the image. Thus a strong indication of an edge between two pixels (a large difference in pixel colour) results in a very small weight between two pixel nodes. The pixels labelled as either foreground or background in the clue marking stage are used to create a foreground and background region model. These models determine the weights between pixel nodes and the Source and Sink nodes. These weights are calculated by determining the probability of the pixel node being part of the background or foreground model.

3. The Source and Sink nodes must be separated in order to perform segmentation. This is done by cutting the links between pixel nodes and the Source and Sink nodes. An energy function E is created which is defined by the sum of all the edge weights that are cut. The Min-Cut/Max-Flow algorithm created by Boykov and Kolmogorov [2004] is used to minimize this energy function, and in so doing segment the graph. This algorithm determines the minimum cost cut that will separate the Source and Sink nodes. The cost of the cut is determined by the sum of all the weights of the links that are cut.

4. After stage 3 is complete all pixel nodes will only be connected to either the Source or Sink node. The purpose of this stage is to transfer the graph information back to the image. All pixel nodes in the graph which are connected to the Source node cause their representative pixels in the image to become part of the foreground. All pixel nodes which are connected to the Sink node cause their representative pixels in the image to become part of the background.

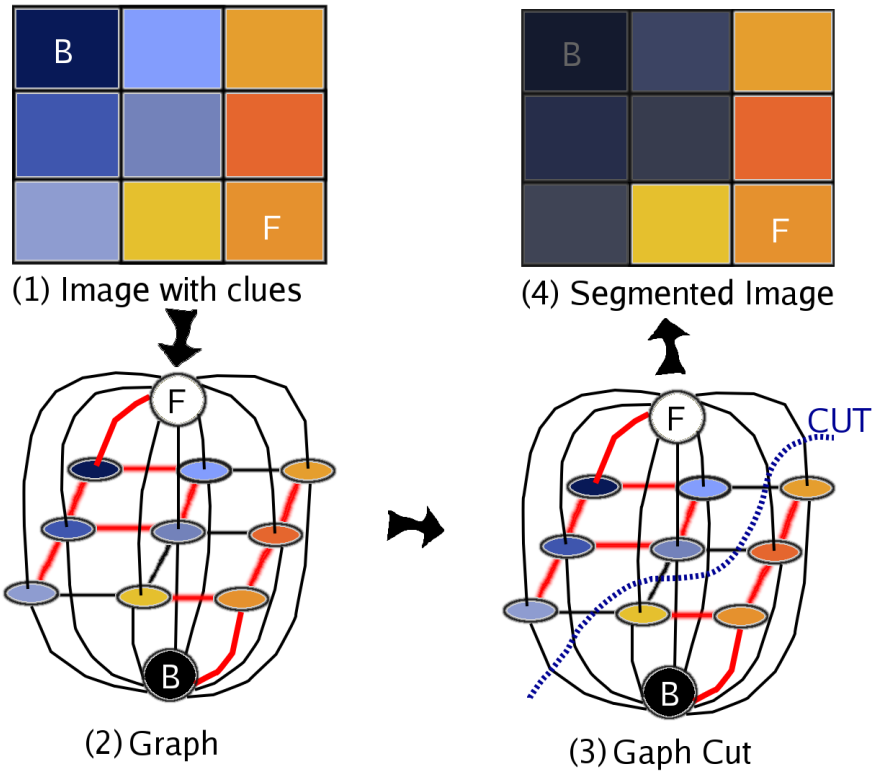


Figure 3.1: The “GrabCut” segmentation process

3.2 Image segmentation using the "GrabCut" Technique

During stage 3 in the "GrabCut" segmentation process (see section 3.1) an energy function is created which is defined by the sum of all the edge weights in the graph which are cut. This energy function is created by extracting region and boundary information from the image. The formula for energy function is as follows:

$$E = U + \gamma V \tag{3.1}$$

In equation 3.1, U is the regional term, which contains region information, and V the boundary term, which contains edge information. The constant γ is used to weight the region and boundary terms. Rother et al. [2004] use a value of 50 for this constant.

Boundary information is always gained by noting the difference in intensity or colour between neighbouring pixels. There are different methods to model region information. We have used three different methods for this purpose.

1. First a method which only works on greyscale images was implemented. This simplifies the modeling of image data and allows the basic structure of the plug-in to be designed. For greyscale images boundary information is obtained by taking the difference in intensity values between neighbouring pixels, and a histogram is used to model region information. This tool followed the suggestions in the paper by Boykov and Jolly [2001].

2. Next a method which worked on colour images was designed. This method follows the "GrabCut" paper by Rother et al. [2004]. Gaussian Mixture Models and an Expectation Maximization clustering algorithm are used to model region information in colour images, and the boundary information is obtained by taking the difference in colour values between neighbouring pixels. The EM clustering algorithm that is used is an implementation based on the suggestions in a book by Kung et al. [2004] in the section entitled Expectation-Maximization Theory. The results using this tool are unsatisfactory and the segmentation process is very slow (see section 5 for a description of these results).

3. Due to the poor results using Gaussian Mixture Models, we have developed a new method of modeling region data. This method uses colour histograms to model region data. A histogram is created for each colour channel in order to model region information, and the boundary information is obtained by taking the difference in colour values between neighbouring pixels.

The following sections 3.2.1, 3.2.2 and 3.2.3 describe these three different approaches to creating region models in more detail.

3.2.1 Region modelling using greyscale histograms

Pixels marked as either foreground or background during the clue marking stage are used to model region information by building up foreground and background histograms. Histograms are built up by noting the frequency with which a pixel occurs within a group of pixels. For greyscale images the intensity of each pixel can range between 0 and 255. A histogram with 256 bins is created for greyscale images, so that each bin holds the frequency at which pixels with the intensity of the bin number occur in an image. Figure 3.2 shows how foreground and background information is used to create histograms in a greyscale image.

The weights between a pixel node and the Source node are determined by the probability that the pixel node lies inside the foreground histogram. Similarly the weight between a pixel node and the Sink node is determined by the probability that it lies inside the background histogram.

3.2.1.1 The graph weights

The edge weights of the graph are defined as follows:

Edge	Weight	Applied to
Between nodes	γV	All neighbouring pixels
Between node and Source terminal	$U(Background)$	Pixels with no hard constraints
	K	Foreground constraints
	0	Background constraints
Between node and Sink terminal	$U(Foreground)$	Pixels with no hard constraints
	0	Background constraints
	K	Foreground constraints

K is a constant that is set to a large value so that edges between nodes that are labelled as foreground or background during the clue marking stage are never cut.

The constant γ is defined in equation 3.1.

U returns the probability that a pixel intensity lies in either the foreground or background, and is defined as follows:

$$U(Background) = -\gamma \times \log(HistogramBack(Pixel)) \tag{3.2}$$

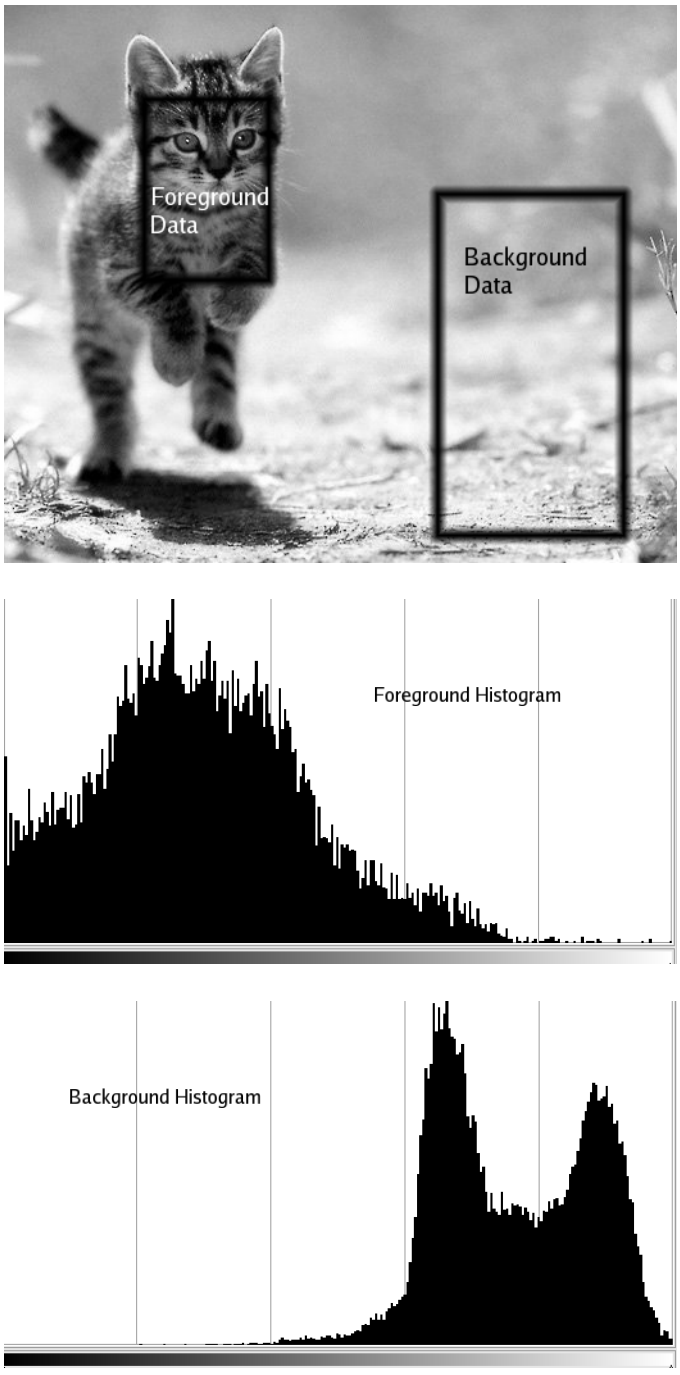


Figure 3.2: Histogram models in a greyscale image

$$U(Foreground) = -\gamma \times \log(HistogramFore(Pixel)) \tag{3.3}$$

In equation 3.2 $HistogramBack$, returns the probability that a pixel lies inside the background histogram. In equation 3.3 $HistogramFore$, returns the probability that a pixel lies inside the foreground histogram.

V measures the difference in intensity between neighbouring pixels and is defined as follows:

$$V = \exp(-B(I_{pixel1} - I_{pixel2})^2) \times \frac{1}{distance} \tag{3.4}$$

In equation 3.4, B is a constant, I_{pixel} refers to the intensity of a pixel, and $distance$ is the distance between the two neighbouring pixels

Setting the weights in this way causes the following:

- The larger the difference in intensity between neighbouring pixels, the smaller the cost, and the greater the chance of the edge being cut. Thus edges are more easily cut than regions of intensity coherence.
- Pixels which are labelled during the clue marking stage are never relabelled, this is due to the large constant K which ensures large edge weights.
- The region costs of unlabelled pixels between the Source and Sink nodes are defined by the probability that their intensity lies in the foreground and background histogram. A high probability results in large cost, thus region coherence is established.

In figure 3.2 pixels with a high intensity value will have a higher probability of occurring in the background histogram than in the foreground histogram, and so the edge weight between the pixel node and the Sink node will be stronger than the edge weight between the pixel node and the Source node. Pixels with a low intensity value will have a higher probability of occurring in the foreground histogram than in the background histogram, and so the edge weight between the pixel node and the Source node will be stronger than the edge weight between the pixel node and the Sink node.

3.2.2 Region modelling using Gaussian Mixture Models

In this design pixels marked as either foreground or background during the clue marking stage are used to model region information by building up Gaussian Mixture Models. A Gaussian Mixture Model is the weighted sum of several Gaussians. To understand Gaussian Mixture Models

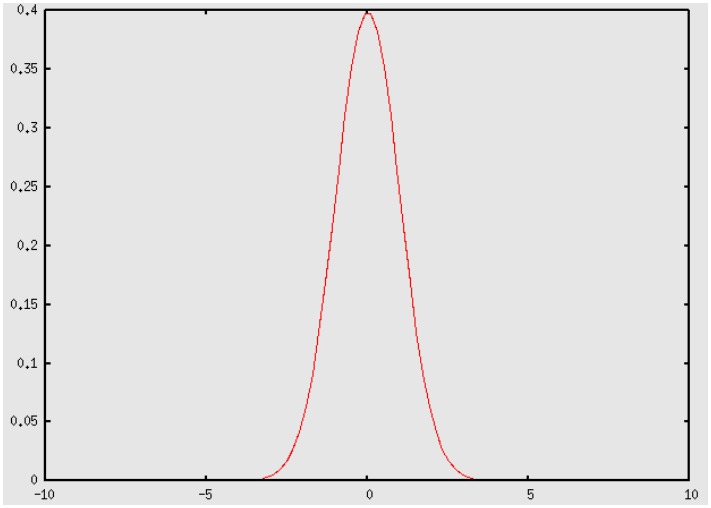


Figure 3.3: A Gaussian probability function

the concept of a Gaussian first has to be understood. Section 3.2.2.1 provides a description of Gaussians, and section 3.2.2.2 describes Gaussian Mixture Models.

3.2.2.1 Gaussians

A Gaussian is similar to a histogram in that it also returns a probability. Given a value x, the Gaussian probability density function returns the probability that x occurs within the distribution. The formula for the basic 1D Gaussian probability density function is as follows:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp(-(x-\mu)^2/2\sigma^2) \tag{3.5}$$

In equation 3.5, σ is the standard deviation and μ the average. This formula produces figure 3.3. In this case $\sigma = 1$ and $\mu = 0$.

The 1D Gaussian probability density function can be extended to 3D, and is as follows:

$$f(x) = \frac{\exp(-\frac{1}{2}(\bar{\mathbf{x}} - \bar{\mu})'\Sigma^{-1}(\bar{\mathbf{x}} - \bar{\mu}))}{\sqrt{(2\pi)^3|\Sigma|}} \tag{3.6}$$

In formula 3.6 Σ is the covariance matrix, and $|\Sigma|$ the determinant of the covariance matrix. The variables $\bar{\mathbf{x}}$ and $\bar{\mu}$ are vectors of 3 values each. For the purpose of modelling colour, these vectors store red, green and blue colour values. This formula will accept a colour, which is made

up of a red, green and blue value, and returns the probability that that colour occurs in the density distribution. This is very useful as a 3D Gaussian probability function is able to encapsulate 3 values into the single concept of a colour. Histograms are only able to encapsulate 1 value. Section 3.2.3 uses 3 individual histograms to model region data, and the concept of individual colours is lost.

3.2.2.2 Gaussian Mixture Models

$$GMM = \Sigma_{i=0}^{n} \pi_i G_i \tag{3.7}$$

A Gaussian Mixture Model is the weighted sum of several Gaussians. Equation 3.7 defines how the Gaussian Mixture Model, GMM, is built up by summing i Gaussians, where each Gaussian, G_i, is weighted by a factor of π_i. Rother et al. [2004] use 5 Gaussians to model the foreground region and 5 to model the background region.

The 5 Gaussians which make up the foreground region are each centred around a different colour found in the foreground region. These colours are the most common colours found in the foreground. The same occurs for the 5 Gaussians making up the background. The weighting of each Gaussian depends on how common the colour is within the foreground or background region.

The process of selecting the most common colour from a foreground or background region in an image is performed by an EM clustering algorithm. The EM clustering algorithm clusters the Gaussians around the most common colours and in doing so determines the means, covariance matrices and mixture parameters of the Gaussians.

Figure 3.4 shows an example 1D Gaussian Mixture Model which is made up of 5 Gaussians. A 1D Gaussian Mixture Model has been used as it employs the same concept as 3D Gaussian Mixture Models, but it easy to visualize and draw. Each peak in this diagram is centered around a common colour in the region being modelled.

3.2.2.3 Clue marking method

The foreground and background clue marking method differs from the greyscale case. In the greyscale case pixels could be marked as either foreground or background. In this case the user drags a rectangle around the foreground object, and the inverse of the selection is set as hard background. Unlike the greyscale implementation, this implementation iteratively solves for the best segmentation, and the pixels inside the rectangle are labelled as unknown, and allowed to change between foreground and background on each iteration. Figure 3.5 shows how a rectangle

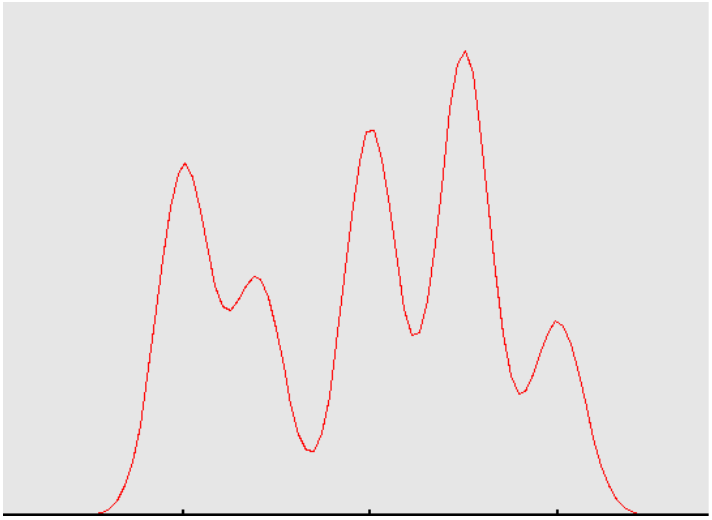


Figure 3.4: A 1D Gaussian Mixture Model

is drawn around the object of interest, in this case an eagle. All pixels inside this rectangle are set as unknown, and all pixels outside the rectangle are set as hard background.

Segmentation of an image is performed as follows:

- A background Gaussian Mixture Model is created from the hard labelled pixels in the clue marking stage.
- A foreground Gaussian Mixture Model is created from pixels inside the user drawn rectangle.
- The following steps are then repeated until the energy function converges:

1. Estimate the means, covariance matrices and mixing parameters of the background and foreground Gaussian Mixture Models based on the current segmentation. The EM algorithm is used for this purpose
2. Assign to every pixel a Gaussian component from the foreground and background model. Each pixel is assigned a background and foreground Gaussian. This is done by determining which Gaussian in the foreground Gaussian Mixture Model produces the highest probability given the pixels colour, and likewise for the background Gaussian Mixture Model.

Figure 3.5: Pixel labelling when using Gaussian Mixture Models

3. Create the graph and segment it, marking pixels as either foreground or background. Region weights in the graph are determined by the Gaussians assigned to each pixel. The weights between a pixel node and the Source node are determined by the probability that the pixel node lies inside the foreground Gaussian. Similarly the weight between a pixel node and the Sink node is determined by the probability that it lies inside the background Gaussian.

3.2.2.4 The graph weights

The edge weights between nodes in the graph are defined as follows:

Edge	Weight	Applied to
Between nodes	γV	All neighbouring pixels
Between node and Source terminal	0	Pixels with hard background constraints
	$U(Foreground)$	Temporary foreground constraints
	$U(Foreground)$	Temporary background constraints
Between node and Sink terminal	K	Pixels with hard background constraints
	$U(Background)$	Temporary background constraints
	$U(Background)$	Temporary foreground constraints

The only hard constraints while working with Gaussians is the initial background labelling. These constraints are enforced by the large constant K. During each iteration the remaining pixels may change their labelling. The weights between pixel and the Source and Sink nodes are determined by the Gaussian Mixture Models. Weights between pixels and the Source terminal are determined by the foreground Gaussian Mixture Model and weights between pixels and the Sink terminal by the background Gaussian Mixture Model. The probability that a pixel lies in a Gaussian Mixture Model is taken as the maximum probability of the pixel lying in any of the Gaussians making up the Gaussian Mixture Model. Thus even though there are 5 Gaussians making up each mixture model, only one of the Gaussians in each model will be used to set the region weights for a pixel node in the graph. The weight terms are defined as follows:

$$U(Foreground) = -\log(\pi_F) + \log(|\Sigma|_F)/2 + f(pixel)/2 \tag{3.8}$$

$$U(Background) = -\log(\pi_B) + \log(|\Sigma|_B)/2 + f(pixel)/2 \tag{3.9}$$

$$V = \gamma \times \exp(-B(pixel1 - pixel2)^2) \times \frac{1}{distance}$$

$$(pixel1 - pixel2)^2 = (pixel1_{Red} - pixel2_{Red})^2 + (pixel_{Green} - pixel2_{Green})^2 + (pixel1_{Blue} - pixel2_{Blue})^2 \tag{3.10}$$

Equation 3.10 determines the difference in colour between two pixels, and is taken as the Euclidean distance in RGB colour space.

In equations 3.8 and 3.9 $f(pixel)$ refers to the probability that the pixel lies in the foreground or background Gaussian Mixture Model.

3.2.3 Region modelling using colour histograms

This design is the same as the greyscale version, except here histograms are built up for each colour channel. The red, green and blue channels in colour images have a possible range of 0 to 255, so histograms are built up in exactly the same way as in greyscale images. The probability that a pixel lies in the foreground region is calculated by the sum of the probabilities that its red value lies in the red foreground histogram, its blue value in the blue foreground histogram and its green value in the green foreground histogram. The weights between a pixel node and the Source node are determined by the probability that the pixel node lies inside the foreground region.

Similarly the weight between a pixel node and the Sink node is determined by the probability that it lies inside the background region.

Colour histograms do not fully represent colour regions in an image. This is because the histograms are separate, and have no concept that a colour is made up of 3 RGB values. Gaussian Mixture Models are more appropriate for representing colour regions as they cluster actual colours, not individual RGB values.

3.2.3.1 The graph weights

This design is an extension of the greyscale case, and all the edge weights are defined as in the greyscale case. In this design three histograms are used to model region data: a histogram for each of the red, green and blue colour channels. The U and V terms of the energy function are defined as follows:

$$\begin{aligned} U(Foreground) &= -\gamma \times (\log(RedHistogramFore(Pixel_{Red})) \\ &\quad -\gamma \times (\log(GreenHistogramFore(Pixel_{Green})) \\ &\quad -\gamma \times (\log(BlueHistogramFore(Pixel_{Blue})) \end{aligned}$$

$$\begin{aligned} U(Background) &= -\gamma \times (\log(RedHistogramBack(Pixel_{Red})) \\ &\quad -\gamma \times (\log(GreenHistogramBack(Pixel_{Green})) \\ &\quad -\gamma \times (\log(BlueHistogramBack(Pixel_{Blue})) \end{aligned}$$

$$V = \exp(-B(pixel1 - pixel2)^2) \times \frac{1}{distance}$$

$$(pixel1 - pixel2)^2 = (pixel1_{Red} - pixel2_{Red})^2 + (pixel_{Green} - pixel2_{Green})^2 + (pixel1_{Blue} - pixel2_{Blue})^2$$

3.3 Matting using the “GrabCut” technique

Often segmentation produces jagged edges and unnatural looking boundaries. This occurs mainly in objects that have fuzzy boundaries like hair or smoke. In order to make the boundaries of these

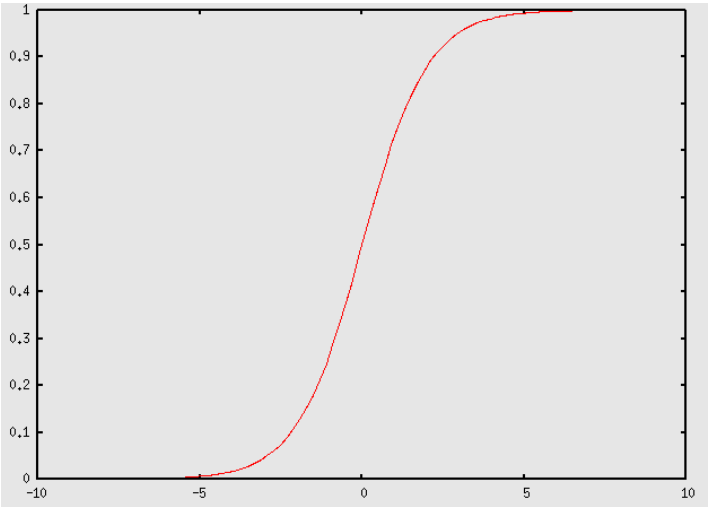


Figure 3.6: The sigmoid function

segmented objects appear more natural Rother et al. [2004] suggest a border matting technique. This technique only performs matting in a narrow strip around the objects boundary and not over the whole image.

Rother et al. [2004] use a technique very similar to Ruzon and Tomasi [2000] in order to perform matting. A smooth profile is assumed for the alpha matte. The alpha profile is set by the sigmoid function, which is defined as follows:

$$S(r, \Delta, \sigma) = \frac{1}{1 + \exp(\frac{-(r-\Delta)}{\sigma})} \tag{3.11}$$

In equation 3.11, Δ determines the centre of the sigmoid function (the value of r that will return a alpha value of 0.5), and σ determines the width of the sigmoid function (the smaller the width, the steeper the transition from 0 to 1). The value returned by the sigmoid function is an alpha value that can range between 0 and 1. Figure 3.6 shows the sigmoid function with Δ = 0 and σ = 1.

The purpose of border matting is to determine the values of σ and Δ which will produce the best alpha profile for the boundary of an object. As the boundaries of segmented objects are not constant, the alpha profile is allowed to vary along the boundary. In order to solve for the best matte, alpha profiles are created for small regions along the boundary of the object. These regions are known as windows, and the process of moving the window along the boundary of the

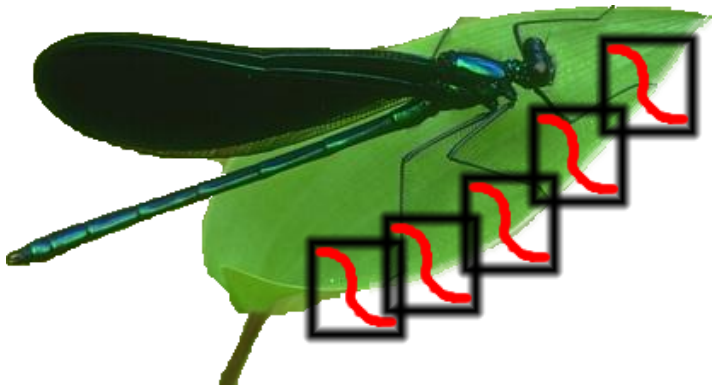


Figure 3.7: Matting windows along an objects boundary

object is known as sliding the window. The optimal global matte is determined by minimizing a cost function which is created from all the individual alpha profiles. Figure 3.7 shows how windows, which are represented by black squares, are created along the edge of a leaf which is being matted. Each window contains its own sigmoid function with different values of Δ and σ. The sigmoid functions are represented by red curves.

In order for matting to work a tri-map needs to be provided. A tri-map is a labeling of pixels as either background, foreground or unknown. This plug-in is designed so that the tri-map is obtained from the segmentation results after an image is segmented. The edge between the foreground and background in the image is expanded to a width of $2r_{Max}$ pixels wide. These pixels make up the unknown region, and are assigned r values according to their distance from the original boundary. Figure 3.8 shows an example matting window with its tri-map. The distance from the boundary to pixel p is r. The distance ranges from $-r_{Max}$ at pixels bordering the background to $+r_{Max}$ at pixels bordering the foreground. The alpha value of pixel p is determined by the sigmoid function for that window.

3.3.1 Energy minimization for border matting

Each matting window has its own sigmoid function which defines the alpha profile for that window. In order to solve for the best overall matte, an energy function is created which incorporates all the sigmoid functions into one equation. The energy function is created so that minimizing it will result in the best overall matte. The energy function is defined as follows:

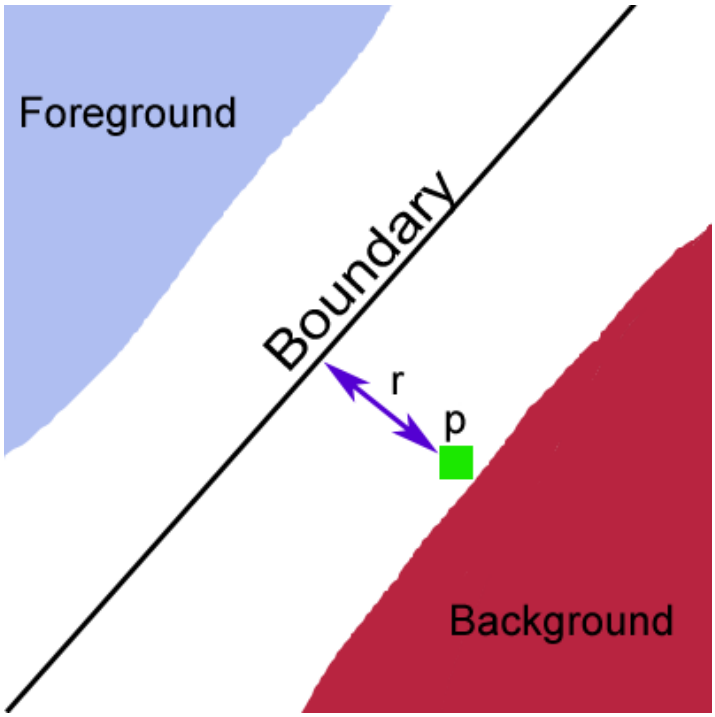


Figure 3.8: The border matting window

$$E = \Sigma_{UPixels} D + \Sigma_{AllWindows} V \tag{3.12}$$

In equation 3.12 $\Sigma_{UPixels}$ refers to the sum of D over all pixels in the unknown region. In this case the unknown region is the sum of all the unknown regions in every matting window.

The purpose of the energy function is to produce a good matte in each window, and maintain smoothness in the alpha profiles between adjacent windows. V is the smoothness term which maintains continuity, and D the data term which produces a good matte for each individual window. The smoothness term is defined as follows:

$$V = \lambda_1(\Delta - \Delta')^2 + \lambda_2(\sigma - \sigma')^2 \tag{3.13}$$

In equation 3.13, λ_1 and λ_2 are constants, and after experimenting with different values, Rother et al. [2004] suggest using a value of 50 for λ_1 and a value of 1000 for λ_2. This term is used to keep the difference between Δ and σ in adjacent windows as small as possible, thus ensuring a smoothly varying alpha profile. The data term is used to ensure the best alpha profile for a particular window and is defined as follows:

$$D = -\log N(Pixel) \tag{3.14}$$

In equation 3.14 N is a 3D Gaussian probability density function. Every window along the objects boundary has a different N function. There are two Gaussians which are used to create N. One Gaussian is created from pixels in the foreground region in the window, and one from pixels in the background region of the window. The Gaussians are added together to produce N. Thus the average matrix, μ_N, and covariance matrix, Σ_N, for Gaussian N, are a sum of the average and covariance matrices from the foreground and background Gaussians. Equations 3.15 and 3.16 show how this summing occurs.

$$\mu_N = (1 - \alpha)\mu_B + \alpha\mu_F \tag{3.15}$$

$$\Sigma_N = (1 - \alpha)\Sigma_B + \alpha\Sigma_F \tag{3.16}$$

The matrices μ_B and μ_F in equation 3.15 are the average matrices from the background and foreground Gaussians. The matrices Σ_B and Σ_F in equation 3.16 are the covariance matrices from the background and foreground Gaussians. The α value used in these equations is defined by the sigmoid function, and depends on the r value of each pixel in each matting window.

For a pixel with a r value near $+r_{Max}$ the α value will be close to one. In this case μ_F and Σ_F will almost solely be used to create μ_N and Σ_N. This is because colour values in the unknown region, in this case, are expected to be from the foreground region, and so the foreground Gaussian distribution is almost solely used. Vice versa for r values near $-r_{Max}$.

Chapter 4

Implementation

This chapter details the way in which "GrabCut" has been implemented as a plug-in for the GIMP.

Section 4.1 describes the general plug-in code that is needed when implementing a plug-in for the GIMP. It also describes the main, render and interface classes that have been created. Section 4.2 gives details of the most common image manipulation functions available in the GIMP that have been used in this plug-in. Section 4.3 provides a UML diagram for the plug-in, and describes the classes which are used. This section also describes the flow of the "GrabCut" plug-in. Section 4.4 describes how matting has been implemented.

4.1 Implementing a plug-in for the GIMP

In order to implement a plug-in for the GIMP a plug-in template is used, which is available on the GIMP developer website [Neary, 2005]. Details on implementing a plug-in for the GIMP can also be found on the GIMP developer website.

In order for a plug-in to be available within the GIMP, it has to be registered in the procedural database (PDB). This allows plug-ins to specify what parameters they accept and the parameters that they produce, as well as their menu hierarchy. When the GIMP is started the PDB is queried, and all the plug-ins are registered. The GIMP allows plug-ins to communicate by a transparent transport of parameters between plug-ins, through the PDB.

The "GrabCut" plug-in is separated into three classes:

1. A class which handles the communication between the "GrabCut" plug-in and the GIMP. This is the main class, and it handles the general management of the plug-in. This involves registering the plug-in in the PDB and calling the main and interface classes.

2. A class that allows pixels to be labelled as either foreground or background. This is the interface class, and is where all the GUI functions are stored. The interface class uses the GIMP Development Toolkit (GTK) to draw a basic user interface.

3. A class which performs all the image manipulation functions, and where the “GrabCut” algorithm is implemented. This is the render class, and is where the majority of the processing is performed.

4.1.1 The main class

There are four methods that all plug-ins have, that will be called at predefined stages during the lifetime of the plug-in by the GIMP. These methods are declared as follows in the main method:

```
GimpPlugInInfo PLUG_IN_INFO = { init, quit, query, run };
```

The init and quit methods are optional, and have not been used in this plug-in. The query method is called the first time the plug-in is present, and every time that it changes. It is here that the plug-in can be registered in the PDB by calling the gimp_install_procedure method.

```
gimp_install_procedure (PROCEDURE_NAME, "GrabCut, interactive foreground
extraction using iterative graph cuts",
"Cut out the foreground of an image",
"Matthew Marsh <g02m1682@campus.ru.ac.za>",
"Matthew Marsh <g02m1682@campus.ru.ac.za>",
"2005",
N_("GrabCut"),
"RGB*, GRAY*, INDEXED*",
GIMP_PLUGIN,
G_N_ELEMENTS (args), 0, args, NULL);
```

The run function is called whenever the plug-in is asked to run. There are three modes in which a plug-in can run, namely: "GIMP_RUN_INTERACTIVE", "GIMP_RUN_NONINTERACTIVE" or "GIMP_RUN_WITH_LAST_VALS". The non-interactive mode is used for running from a script or batch file, and the last vals option is when the “Filters->Repeat Last" shortcut is run. As this plug-in is inherently interactive, only the “GIMP_RUN_INTERACTIVE” mode is considered. If it is run in the interactive mode, the interface class is called to produce a user interface, and then the render class is called to perform the segmentation.

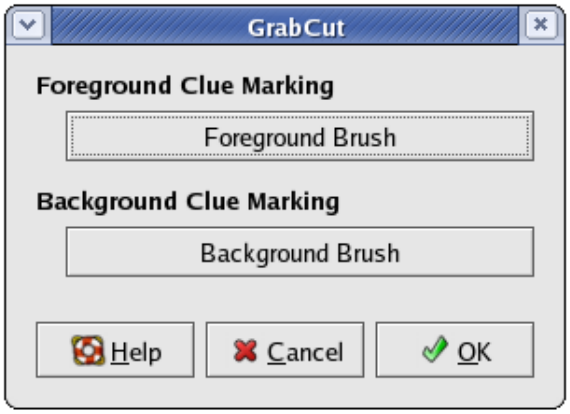


Figure 4.1: The user interface

4.1.2 The interface class

The GIMP development toolkit is used to create a basic graphical interface that allows the selection of a foreground and background brush to label pixels in the clue marking stage. The user interface for the "GrabCut" plug-in is very simple. It allows the user to choose between a background and foreground brush, confirm the clues or cancel. The foreground and background brush options change the current colour of the drawing tools. White represents foreground clues, and black the background clues. As long as the user draws onto the image with these colours all the clues will be accepted. For this reason any of the gimp drawing tools can be used, brush size changed, and any misplaced brush strokes can be undone by using the standard undo feature in the GIMP. Figure 4.1 shows the basic user interface

All the clues are captured into an array and sent to the render class.

4.1.3 The render class

The render class is where all the image manipulation takes place. The render class is provided with all the user clues. It creates histograms or Gaussian mixture models, depending on the implementation, from these clues. All the cost functions are defined in this class, and it is here that a graph is created. The Min-Cut/Max-Flow class by Boykov and Kolmogorov [2004] is used for the segmentation of this graph. Once the graph is segmented, the image can be segmented and displayed to the user.

In order to implement Gaussian Mixture Models various matrix manipulation functions are

included in this class. These include functions to obtain the inverse, transpose and determinant of a matrix, as well as a function which calculates the covariance matrix from an array of data. These functions are necessary in order to determine equation 3.6.

4.2 General image manipulation functions

As this plug-in is so closely related to images and image information, it is imperative to be able to access image data, manipulate this data and write data back to the image. The GIMP provides an excellent set of image manipulation functions. The GIMP API was used extensively in the creation of this plug-in, and provides a list of all built in methods, the actions they perform, and the parameters that they need.

The most important structure within the GIMP for working on image data is the "drawable". A drawable is any object from which image data can be obtained, and sometimes written to. Drawables are the way in which the GIMP allows programmers to get access to image data.

Many objects within the GIMP are drawable. Layers and selections, among others, are drawables. All drawables have a unique drawable id, which can be obtained by using the gimp_drawable_get method. The GIMP uses a structure which allows direct access to the data in a drawable. This is the GimpPixelRgn structure. The GimpPixelRgn structure can be initialized with a drawable id as follows:

```
    gimp_pixel_rgn_init(&dest, AlphaLayer, 0, 0, width, height, TRUE, TRUE); //
The drawable "AlphaLayer" is initialized for writing
    gimp_pixel_rgn_init(&source, AlphaLayer, 0, 0, width, height, FALSE, FALSE);
// The drawable is initialized for reading
```

Once a pixel region has been initialised there are numerous methods for reading and writing data from and to the image. These include individual pixel, row, column, and block read and writes. This plug-in uses the row method of reading and writing. The following code snippet reads drawable data line by line into an array, modifies the data and writes the data back to the drawable. Image information is stored in the one dimensional data array in the form of Red, Green, Blue and Alpha value for each pixel. Thus the first 4 values in the array are for the first pixel, the second 4 for the second and so on. The following code modifies every fourth element in the array, thus it only modifies the alpha value of every pixel in the image. Variations on this code are used throughout the plug-in:

```
    for (loop = 0; loop < height; loop ++) {
      gimp_pixel_rgn_get_row(sourcepr, data, 0, loop, width); //Read in every row
from the drawable
      for (loop2 = 0; loop2 < width; loop2 ++)
        data[(loop2 * 4) + 3] = opacity[loop][loop2] * 255; //Modify the alpha value
of every pixel
      }
      gimp_pixel_rgn_set_row(destpr, data, 0, loop, width); //Write the data back to
the drawable
    }
```

4.3 The "GrabCut" image segmentation implementation

4.3.1 The "GrabCut" UML diagram

The relationship between all the classes implemented for plug-in can be best understood by a UML diagram. This diagram includes the main, interface and render classes which have already been discussed in section 4.1, as well as other classes which are used to perform image segmentation. These are the graph, neighbours, histogram and cov classes. Figure 4.3 shows the basic layout of the plug-in. The purpose of these classes is as follows:

- The graph class is used to create the graph. It has methods to set up the edge weights between nodes in the graph, and uses the Min-Cut/Max-Flow method to segment the graph. This class is an implementation by Boykov and Kolmogorov [2004] and has not been modified in any way by the authors of this project.

- The neighbours class is used to store the neighbouring relationship between pixels in the image. During the evaluation of the weights between pixel nodes in the graph, information about neighbouring pixels nodes is needed. This includes the distance between the pixels, and their x and y offsets relative to their neighbours. The neighbours class stores all the relationships between pixels, and simplifies the process of evaluating edge weights. A neighbouring labelling system is adopted, in which neighbours are labelled in a clockwise fashion. Thus the neighbour directly above a pixel will be neighbour 0. Figure 4.2 shows this relationship between pixel x and its neighbours. The neighbours class also ensures that pixels along the border of images are given the correct neighbours, and not allowed to have neighbours which extend beyond the boundaries of the image.

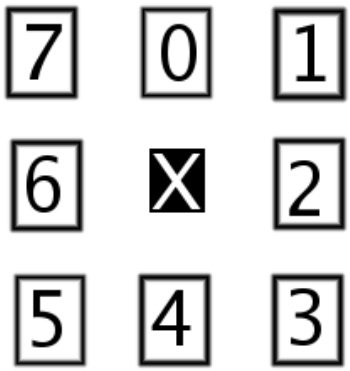


Figure 4.2: The neighbourhood labelling system

- The histogram class is used to model region data in the greyscale and colour histogram implementation. This class builds up histograms from an array of values, and has methods to normalize the histogram and evaluate the probability of an intensity value lying in the histogram. The histogram has 256 bins, and each bin stores the percentage that that intensity occurs in the image. For the greyscale implementation one histogram is uses for each region model. For the colour implementation three histograms are used for each region model.

- The cov class is used to implement the Expectation Maximization data-clustering algorithm suggested by Kung et al. [2004], and is only used in the Gaussian Mixture Model implementation. This algorithm is used to iteratively determine a specified number of Gaussian clusters from an array of values. For the purpose of this implementation the suggestions by Rother et al. [2004] are followed, and 5 Gaussians are chosen for each Gaussian Mixture Model. The purpose of the EM clustering algorithm is to determine the means, covariance matrices and the weights of each Gaussian in the Gaussian Mixture Model.

4.3.2 The flow of the "GrabCut" plug-in

The flow of the "GrabCut" plug-in is as follows:

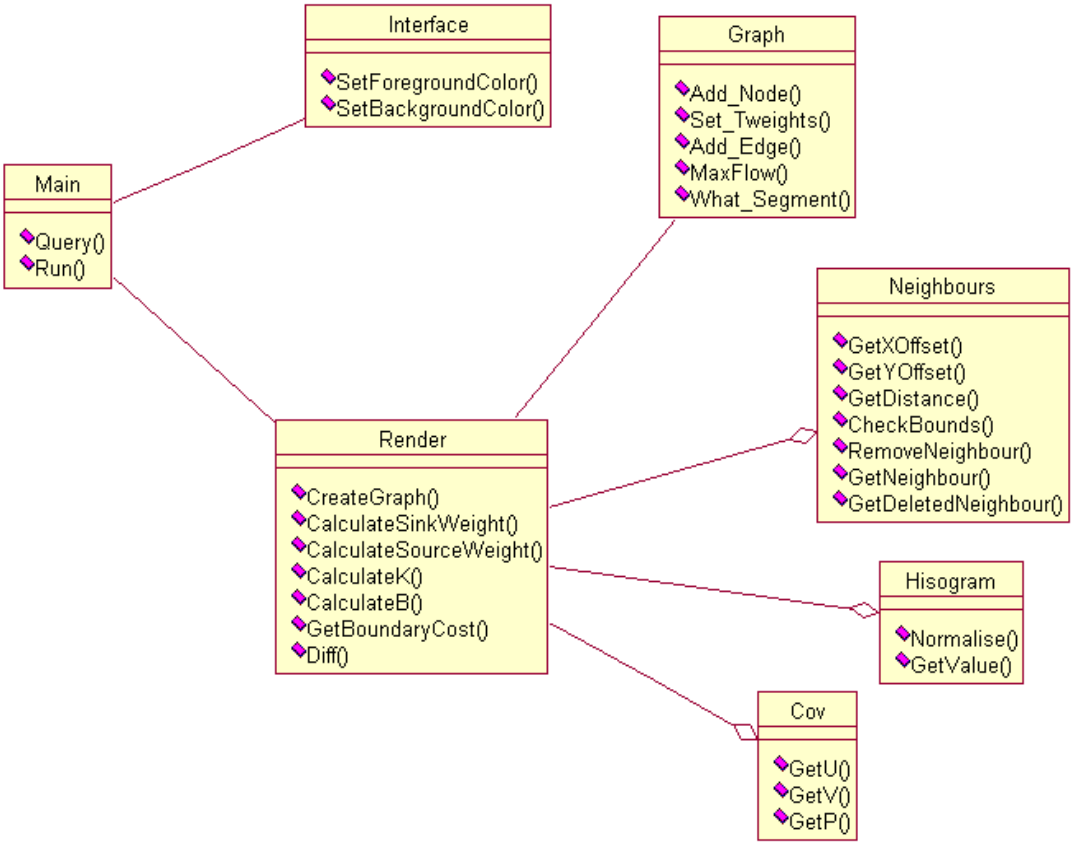


Figure 4.3: A UML diagram for the plug-in

1. The user interface is displayed, and clues are drawn onto a transparent layer. The plug-in creates a transparent layer above the image in the layer stack. This allows the brush strokes to be captured, while retaining the integrity of the original image.

2. The clue information is read back from the image and stored in an array. This is a 2D array which labels pixels as either foreground, background or an unknown region. This array is then passed to the render class.

3. The render class uses the array holding the labelling to build up foreground and background region models from the data. These models will either be histograms or Gaussian Mixture Models depending on the implementation.

4. The K and B constants used in all the cost functions are calculated.

5. A graph is created by evaluating the boundary and region costs for each pixel.

6. The Min-Cut/Max-Flow algorithm is used to segment the graph. This segmentation is a labelling of pixels as either foreground or background and doesn't affect the image.

7. If Gaussian Mixture Models are used this process is iterative, and further steps are performed. See section 3.2.2.3 for the full details.

8. The segmentation results are transfered to the image drawable by editing the pixel alpha values in the image. Pixels in the background are set to have an alpha value of 0, and those in the foreground an alpha value of 255. The GIMP's built in feature of being able to convert an alpha layer to a selection is used to select just the pixels belonging to the foreground region.

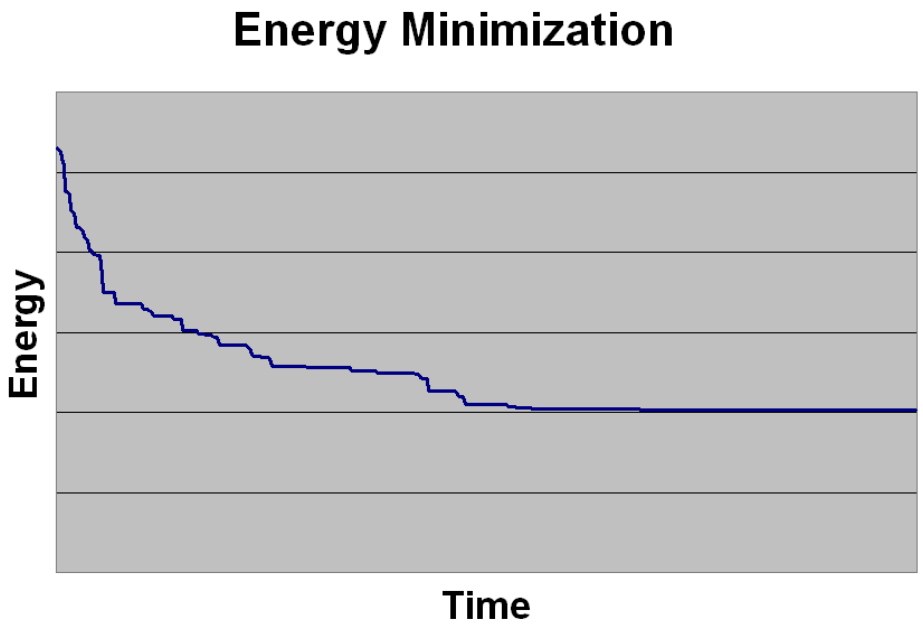
4.4 The "GrabCut" matting implementation

4.4.1 Energy minimization

In order to minimize E, which is defined in section 3.3.1, Rother et al. [2004] use a Dynamic Programming technique. Due to time constraints a less complicated method of minimising energy functions is used. This technique uses a genetic algorithm to interactively solve for a minimum value of E.

10 genes are used, and each gene holds values of Δ and σ for every window along the objects boundary. The genes are initialized with Δ values of 0 and σ values of 1 for each window. This

Figure 4.4: Energy minimization using a genetic algorithm



ensures a good initial sigmoid function for each window. See figure 3.6 for a diagram of this function. The following process is then iterated until convergence:

1. The algorithm determines which gene produces the minimum value of E
2. New genes are bred from the best gene.

New genes are bred from the best gene by using the best genes values of Δ and σ, and adding on small amounts of variance to Δ and σ in randomly selected windows. Figure 4.4 shows how the energy function is rapidly minimized by using a genetic algorithm.

4.4.2 Border matting in the GIMP

The GIMP provides some useful features which are used in the matting implementation. The ability to create paths in the GIMP is used to define the border region to be matted. Paths can be created by allowing the user to specify key points along the path, and once the path is created,

points along the path can be calculated by supplying a built in function with a distance along the path.

The GIMP allows selections to be converted into a border selection (a selection 1 pixel wide which follows the boundary of a selection). It also allows selections to be grown. These functions were used to assign r values to pixels in the unknown region as follows:

1. Convert the selection obtained from segmentation into a border selection
2. Grow the selection
3. Pixels within the selection which are labelled as foreground are assigned a positive value of r according to the number of times the selection has been grown
4. Pixels within the selection which are labelled as background are assigned a negative value of r according to the number of times the selection has been grown
5. Repeat from step 2 until the selection is grown 6 times

This labels all pixels 6 or less pixels away from the original section according to their distance from the boundary.

Chapter 5

Results

This chapter discusses the results obtained using the "GrabCut" plug-in in th e GIMP. All results are all obtained using Linux on a 3 Ghz Pentium 4 PC with 1 Gig of RAM.

This chapter is divided into 5 sections. Sections 5.1, 5.2 and 5.3 discuss the results using the three different region models, namely greyscale histograms, Gaussian Mixture Models, and colour histograms. These sections are followed by section 5.4 which compares the GIMP's built in segmentation tools with the colour histogram implementation of "GrabCut". Section 5.5 details the results found using the matting technique suggested by Rother et al. [2004]. The segmentation results will be discussed in terms of processing time, and the rather subjective topic of appearance.

5.1 Segmentation results using greyscale histograms

In order to test the segmentation results 5 images where each scaled to 3 different sizes, producing a test set of 15 images. Each of these images were segmented, and the segmentation process timed. Table 5.1 shows the average segmentation time for the three different sizes. From this table it can be seen that for even very large images the processing time is quick.

The user interaction process is simple, and it takes about 10 seconds to draw foreground and background clues. The current greyscale implementation does not allow further clues to be drawn once segmentation has taken place. This is frustrating as the user has to remember where previous clues were placed, and where to place future clues in order to obtain the best segmentation by re-segmenting the original image. Good results can however still be obtained by using this approach. Figure 5.1 shows how a complex image with fuzzy boundaries has been segmented well using the "GrabCut" technique.

Figure 5.1: Greyscale segmentation results

(a) Original image (b) Clues drawn (c) Segmentation result

Image Size	540 x 340	800 x 600	1024 x 768
Segmentation Time	3.5 sec	11 sec	15 sec

Table 5.1: Greyscale segmentation times

5.2 Segmentation results using Gaussian Mixture Models

In order for a Gaussian Mixture Model to be created individual Gaussians have to be created by detecting colour clusters. The process of detecting these clusters is performed by an EM clustering algorithm. The EM clustering algorithm as currently implemented, tends to be extremely sensitive to local maxima. This means that once a cluster is detected it will be used as one of the five clusters, even though better solutions may exist. It also tends to create very large Gaussians encompassing all the data points, instead of creating 5 small Gaussians. The process of determining the Gaussians is also very slow. The segmentation results were also heavily dependent on the starting Gaussian Mixture Models. The problems with the EM clustering algorithm made it impossible to test the segmentation process.

In order to test the rest of the process some manually chosen starting parameters for the Gaussians are hard coded in for each image. Testing in this manner is a very tedious task as each Gaussian has to be initialized with good values of $\bar{\mu}$ and Σ. For this reason only a few images were tested.

Even with hard coded in values the segmentation results were not satisfactory. In some cases the images were correctly segmented, but often the results were very bad.

As it is currently implemented, this method of region modeling is very slow and does not

Image Size	540 x 340	800 x 600	1024 x 768
Segmentation Time	4.5 sec	14 sec	21.5 sec

Table 5.2: Colour histogram segmentation times

consistently produce good segmentation results. Segmenting an image using this technique takes at least 30 seconds to complete.

5.3 Segmentation results using colour histograms

As in the greyscale case, the segmentation results were tested by using 5 images which where scaled to 3 different sizes. Each of these images were segmented, and the segmentation process timed. Table 5.2 shows the average segmentation time for the three different sizes. From this table it can be seen that for even very large images the processing time is quick. The processing time has increased from the greyscale case by a few seconds for the three image sizes, but this increase in time is not significant.

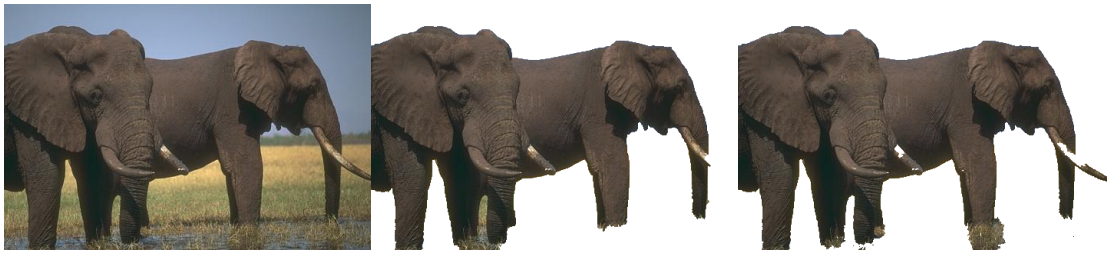
The user interaction process is the same as in the greyscale case, and it takes about 10 seconds to draw foreground and background clues.

This implementation allows further clues to be drawn once the algorithm has completed. This makes the user interaction process very simple, and the best segmentation can usually be obtained within 2 segmentations.

The Colour implementation using histograms produces very good results, and is still quick. The results using this technique are at least as good, and often better than the much slower Gaussian method. Figure 5.2 shows a comparison between segmentation results using these two methods, and the time it took to perform the segmentation. Figure 5.3 shows some segmentation results obtained by using the colour histogram implementation. All these results are completed without in one step (clues were only provided once).

5.4 Comparing “GrabCut” to the Segmentation Tools Built Into GIMP

It would be pointless to add a segmentation tool to the GIMP if it already contains better tools. The purpose of this section is to compare the “GrabCut” segmentation technique to these built in tools. The built in tools are the Magic Wand tool and Intelligent Scissors tool.



(d) Original image

(e) Image segmented using colour histgrams (3s to segment)

(f) Image segmented using Gaussian Mixture Models (35s to segment)

Figure 5.2: A comparison between Gaussian Mixture Models and colour histograms

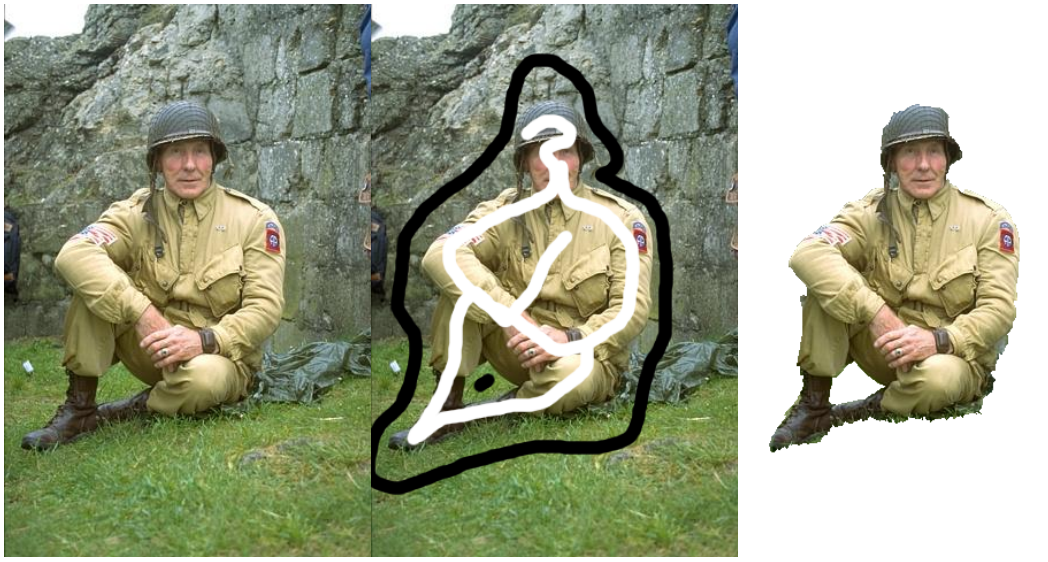
The Magic Wand tool is easy to use; however it only produces good segmentation results on very simple images. Figure 5.4 shows how "GrabCut" produces better results than the Magic Wand tool on a very simple image. The Intelligent Scissors tool is powerful, and the results are comparable to the results achieved with "GrabCut", however it can be very cumbersome. It places a large work load on users if they attempt to segment images that are highly textured. "GrabCut" places a relatively low work load on users for all images. Figure 5.5 shows how a highly textured image is easily segmented using "GrabCut". Few clues need to be given, and the actual segmentation processing took 3 seconds. Segmenting this image using Intelligent Scissors took 2 minutes and produced poor results.

From the results of this section it can be seen that the "GrabCut" plug-in performs as well or better than the other segmentation tools built into the GIMP.

5.5 Border matting results

Due to time constraints the Dynamic Programming approach of minimizing energy suggested by Rother et al. [2004] is not adopted. Instead a simple genetic algorithm method is used to minimize the energy function. This means that the matting results are produced fairly slowly, however it has allowed the matting technique suggested by Rother et al. [2004] to be implemented and tested.

The method produces good results under most circumstances. One situation has been identified where matting results using this technique are not satisfactory. This occurs if the boundary of an object in an image contains fine textures that extend beyond the boundaries of the sliding



(a) Original image (b) Clue marking stage (c) Segmentation result

(d) Original image (e) Clues drawn (f) Segmentation result

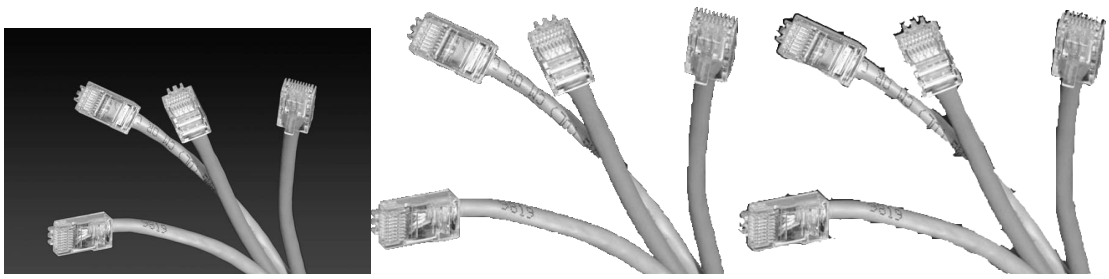
Figure 5.3: Segmentation results using colour histograms

(a) Original image

(b) Segmentation using "GrabCut"

(c) Segmentation using Magic Wand

Figure 5.4: A comparison between the Magic Wand tool and “GrabCut”



(a) Original image

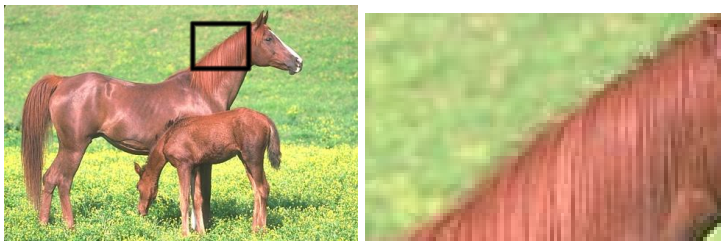
(b) Fine textures are segmented quickly and easily using "GrabCut"

(c) Segmentation results using Intelligent Scissors

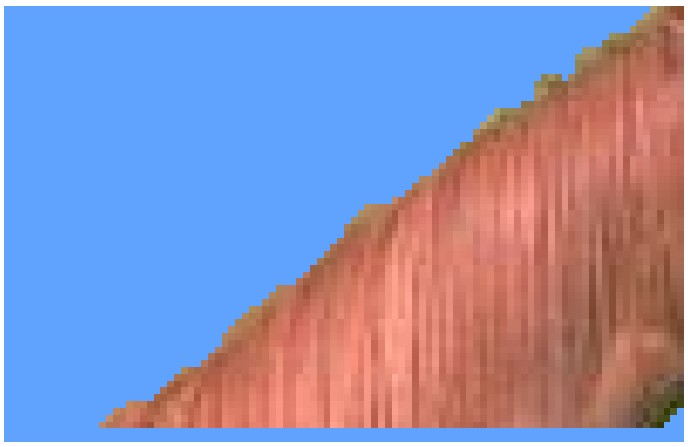
Figure 5.5: A comparison between Intelligent Scissors and “GrabCut”

window. Matting is only performed inside the window, and so images containing long strands of hair, for example, are not matted well. Figure 5.7 shows how long strands of hair are removed when border matting is performed. Border matting on other images significantly improves the overall appearance of the final segmentation. Figure 5.6 shows how matting improves the appearance of the segmentation of a horses mane.

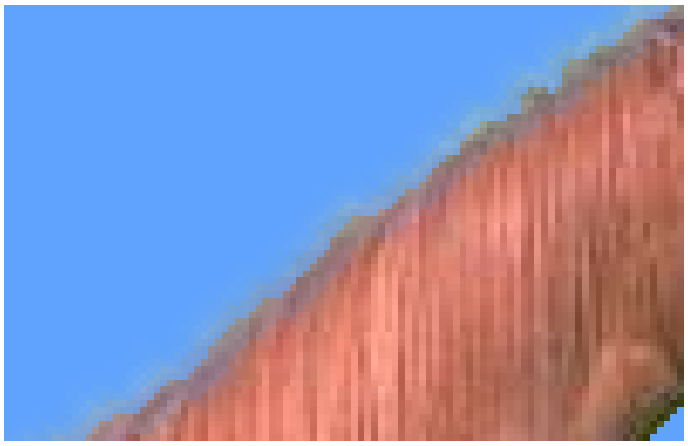
Figure 5.6: Border matting results for fine textures



Original image

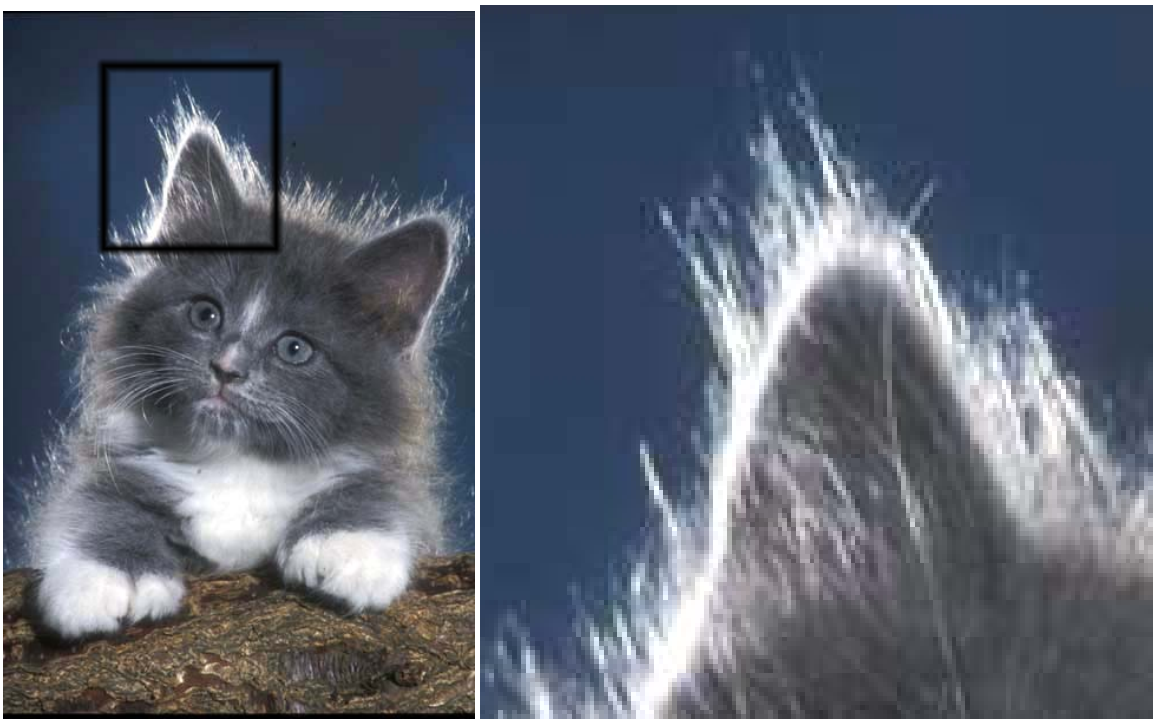


No border matting



With border matting

Figure 5.7: Border matting results for long, fine textures



Original image

Matting results

Chapter 6

Conclusion

Various versions of the "GrabCut" technique have been implemented as plug-ins for the GIMP:

- A greyscale implementation which uses intensity histograms to model region data
- A colour implementation which uses Gaussian Mixture Models to model region data
- A colour implementation which uses a histogram for each colour channel to model region data

The greyscale implementation of "GrabCut" is fast and easy to use. Intensity histograms are sufficient for modeling region data in greyscale images. It performs as well or better than the other segmentation tools built into the GIMP on greyscale images.

The plug-in which uses Gaussian Mixture Models and the EM clustering algorithm is slow and the segmentation results depend heavily on the starting Gaussian parameters. When hard coded starting parameters are used, the segmentation results are sometimes comparable to the results from the histogram implementation, but often worse. The method of only marking background pixels provides the algorithm with less information to work with and so increases the computational time.

The colour histogram implementation retains the speed of the black and white version, and the results are still good, even though some information is lost in the modelling of regions (see section 3.2.3). The method of drawing onto the image with a background or foreground brush provides more information to the algorithm and is more intuitive to users in cases where the foreground region is not rectangular.

The border matting technique for "GrabCut" significantly improves the segmentation results on most images. One case has been identified in which the results are not satisfactory. This

occurs if the boundary of an object in an image contains fine textures that extend beyond the boundaries of the sliding window.

6.1 Contributions

This thesis is a thorough investigation into the implementation of the segmentation technique proposed by Rother et al. [2004]. A greyscale and two colour versions of "GrabCut" have been implemented for the GIMP. The GIMP requires a fast, easy to use segmentation technique, and this is provided to the open source community in the form of the "GrabCut" plug-in.

A novel approach of extending the method of representing region information with histograms in greyscale images to colour images has been implemented by creating a histogram for each colour channel. This method produces good results.

The border matting technique suggested by Rother et al. [2004] has been implemented and tested, and produces good results.

6.2 Future Work

Further work needs to be conducted on the EM clustering algorithm to ensure that it is implemented correctly. The matting technique suggested by Rother et al. [2004] is minimized by using Dynamic Programing. Future works needs to be done so that this is implemented. This will allow for much faster processing of the border matte. Rother et al. [2004] suggest a method of pixel stealing which reduces background colours bleeding into the foreground matte. This also needs to be implemented in the future.

Bibliography

Dana H. Ballard and Christopher M. Brown. *Computer Vision*. Prentice-Hall Inc., New Jersey, 1982.

Andrew Blake, Carsten Rother, M. Brown, Patrick Perez, and Philip Torr. Interactive image segmentation using an adaptive GMMRF model. In *Computer Vision - ECCV 2004: 8th European Conference on Computer Vision, Prague, Czech Republic*, pages 428 – 441, 2004.

Yuri Boykov and Marie-Pierre Jolly. Interactive graph cuts for optimal boundary & region segmentation of objects in N-D images. In *Proceedings of the International Conference on Computer Vision (ICCV)*, volume I, pages 105–112, 2001.

Yuri Boykov and Vladimir Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(9):1124–1137, 2004. Available at: http://www.csd.uwo.ca/faculty/yuri/Abstracts/pami04-abs.html. Last Accessed: 07 November 2005.

Yung-Yu Chuang, Brian Curless, David H. Salesin, and Richard Szeliski. A Bayesian approach to digital matting. In *Proceedings of IEEE Computer Vision and Pattern Recognition (CVPR 2001)*, volume II, pages 264–271, December 2001.

N. E. Davison, H. Eviatarc, and R. L. Somorjaic. Snakes simplified. *Pattern Recognition*, 33 (10):1651–1664, October 2000.

ADOBE SYSTEMS INCORP. *Adobe Photoshop User Guide*. 2002.

Michael Kass, Andrew Witkin, and Demetri Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, 1(4):321–331, 1987.

Vladimir Kolmogorov and Ramin Zabih. What energy functions can be minimized via graph cuts? *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26 NO. 2, 2004.

Sun Yuan Kung, Man Wai Mak, and Shang Hung Lin. *Biometric Authentification: A Machine Learning Approach*. Prentice Hall PTR, Unknown, September 2004.

Yin Li, Jian Sun, Chi-Keung Tang, and Heung-Yeung Shum. Lazy snapping. *ACM Transactions on Graphics*, 23(3):303–308, 2004.

E. N. Mortensen and W. A. Barrett. Toboggan-based intelligent scissors with a four parameter edge model. In *Proceedings of IEEE Computer Vision and Pattern Recognition (CVPR '99)*, volume II, pages 452–458, Fort Collinsw CO, June 1999.

Eric N. Mortensen and William A. Barrett. Intelligent scissors for image composition. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 191–198. ACM Press, September 1995.

Dave Neary. How to write a gimp plug-in, 2005. Available at: http://developer.gimp.org/. Last Accessed: 07 November 2005.

Patrick Perez, Michel Gangnet, and Andrew Blake. Poisson image editing. *ACM Transactions on Graphics*, 22(3):313–318, 2003.

Carsten Rother, Vladimir Kolmogorov, and Andrew Blake. "GrabCut": interactive foreground extraction using iterated graph cuts. *ACM Transactions on Graphics*, 23(3):309–314, 2004.

Mark A. Ruzon and Carlo Tomasi. Alpha estimation in natural images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume I, pages 18–25, Hilton Head Island, SC, June 2000.

Jian Sun, Jiaya Jia, Chi-Keung Tang, and Heung-Yeung Shum. Poisson matting. *ACM Transactions on Graphics*, 23(3):315–321, 2004.