Segment-Based Image Matting using Inpainting to Resolve Ambiguities

by

Heng Ping Nabil Christopher Moh

S.B. Computer Science and Engineering, M.I.T. (2008)
S.B. Economics, M.I.T. (2008)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2008

Author .....
Department of Electrical Engineering and Computer Science
August 12, 2008

Certified by .....
Frédo Durand
Associate Professor
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Professor of Electrical Engineering
Chairman, Department Committee on Graduate Theses

Segment-Based Image Matting using Inpainting to Resolve Ambiguities

by

Heng Ping Nabil Christopher Moh

Submitted to the Department of Electrical Engineering and Computer Science
on August 12, 2008, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Image Matting and Compositing [6, 25] – the extraction of a foreground element from an image and overlaying it over a different background – are two important operations in digital image manipulation. The extraction of the foreground element and its composition over an existing background is performed using a mask known as an *alpha matte*, which is generated by Image Matting. The problem of Image Matting is inherently ill-posed and has no "correct" solution; however, several matting algorithms have been proposed. This thesis studies the popular *Bayesian Matting* [9] algorithm in detail, and documents several problems with regard to its efficiency and accuracy. Inspired by these problems, this thesis proposes two major ideas: Firstly, a new Segment-Based Matting Algorithm that incorporates shading and has a closed form solution. Secondly, a new general approach that uses Digital Inpainting – the technique of restoring defective areas in digital images – to resolve ambiguous areas in the alpha mattes. This thesis demonstrates that the combination of these ideas improves both the efficiency and accuracy of Image Matting. From the results obtained, this thesis proposes the following idea: The degree of local smoothness enforced in the alpha matte should depend on the local color distribution; the more similar the local foreground and background color distributions are, the greater the amount of smoothness enforced.

Thesis Supervisor: Frédo Durand
Title: Associate Professor

Acknowledgments

This thesis would not have been possible without the help of many people.

First and foremost, my utmost gratitude goes to my thesis supervisor, Frédo Durand, for countless things, including providing me the opportunity to work on this thesis, providing me with research guidance and good ideas when I was stuck, and providing me the necessary resources to work on this problem.

I am extremely grateful to my family for their undying support and their encouragement in helping me get through MIT.

I thank the many people who have provided technical resources – such as code and sample images – that have been useful to me. Notables include Marcelo Bertalmio, Kevin Chen and Anat Levin.

I would like to thank the Singaporean undergraduate community at MIT, and my neighbors in Senior House Second Ware, for being such good friends and a source of community support.

I have grown a lot academically at MIT, and I would like to thank everyone who has contributed to this academic growth, including my academic supervisor, Ronitt Rubinfeld, the Course VI Undergraduate Administration – especially Anne Hunter – and all the professors and TAs who taught the classes I took.

I would also like to thank all my friends at MIT. You have definitely made my life at MIT a lot more meaningful, worthwhile and fun.

Finally, to everyone this thesis owes a debt to but I have not mentioned, I thank you and offer my sincere apologies for omitting you from this list of acknowledgements.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Digital Compositing and Matting are two important operations in Digital Image Processing. In Compositing, a foreground image is overlaid over an existing background. This foreground image is extracted from an existing image using a mask known as an *alpha matte*, which will also be interchangeably referred to as the *alpha channel* or simply the *matte*. Matting, the counterpart of compositing, is the process that generates the matte. An example of Matting and Compositing is shown in Figure 1-1.

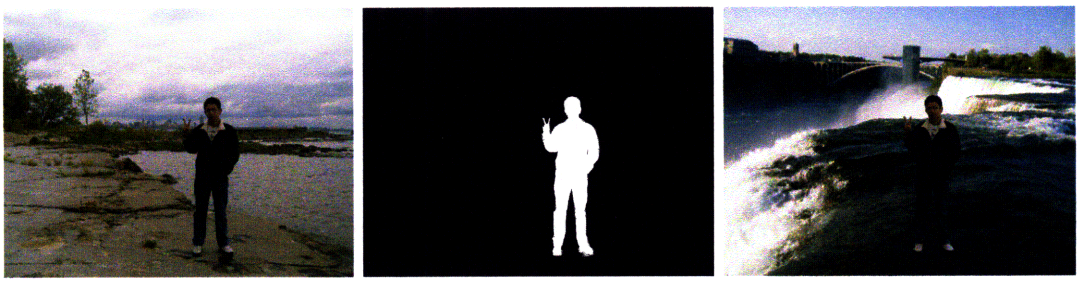


Figure 1-1: How to Walk on Water: An example of Matting and Compositing. From the original image (left), a matte is obtained (center). This matte is then used to composite the foreground onto a different background (right).

Matting has a long history dating from the 19th century. We will only briefly mention some of its history here; a more detailed account can be found in expositions such as [6] and [8].

Optical printers in the early 1900s created composites using an *optical compositing* technique [4] that used three film plates – a foreground plate, a matting plate and a background plate – to create new composite images. The foreground plate contained the foreground image and the background plate contained the background on which the foreground would be composited on. The matting plate helped to combine the foreground plate and the background plate; it was a monochrome plate that helped to control the foreground opacity and ensure that only the desired elements from the foreground plate were composited onto the background. Clearly the matting plate was the most important plate and designing and producing it well was crucial.

Matting has obvious applications in the film and video industry. In the 1960s and 1970s, Petro Vlahos invented *Blue-Screen Matting* through a series of patents that earned him a Scientific and Technical Academic Award of Merit at the 1994 Oscars; his work is formalized by Smith and Blinn [32]. In Blue-Screen Matting, actors are filmed against a constant-colored (usually bright blue or green) screen; the background is removed using a color difference method and the resulting images are composited onto a different (often computer generated) background to obtain special effects. This technique is still used today.

Matting is also used in the print and magazine industry. The front cover of magazines often features images (such as celebrities in fashion magazines or haute cuisine in food magazines). These images are often extracted from a digital picture or photo using matting and then composited onto the magazine cover.

In recent years, with the increasing popularity of image editing software such as CorelDraw and Adobe Photoshop, together with the growing use of digital cameras among the general public, matting has also become more popular with amateur image editors and casual computer users. Cutting out a foreground element from a digital photo and compositing it on a different background, in order to create logos, forum avatars, MySpace and Facebook pictures, etc. has become commonplace; in the Internet jar-

gon, it has become popular to refer to it as a "Photoshop" (as in "I Photoshopped my head onto his body").

Matting is closely related to Image Segmentation, which deals with the problem of separating objects in an image from the background [1]. There are many different techniques for image segmentation, such as thresholding methods [29] to separate objects with different light intensities from the background, or edge detection methods [36] such as Laplacian zero-crossings that try to detect the boundaries between object and background. Although these techniques have been fairly well-studied, we will not encounter them in this thesis; there are important differences between Matting and Image Segmentation. We will list a few:

1. Image segmentation is binary – a pixel either belongs to an object or to the background. However, matting seeks partial coverage information: The alpha matte can take fractional values, especially at boundaries where the foreground blends into the background; these fractional values allow the extracted foreground element to blend better into the background it is composited on.

2. Image segmentation is generally unsupervised and without human intervention. However, matting generally requires human intervention in order to generate an accurate alpha matte.

3. In an image with many objects (such as a photo with many people), image segmentation will attempt to separate each of these objects from the background. In matting, however, in most cases the human user is only interested in obtaining one particular object and removing everything else – including all other objects.

4. Image segmentation is primarily a useful tool in Computer Vision – for example, when a robot needs to identify obstacles to avoid. Matting, on the other hand, is a tool in Image Processing, for images to be viewed by the human eye.

The remainder of this chapter will proceed as follows: Section 1.1 will briefly present

the mathematical framework for Digital Image Compositing and Matting. Section 1.2 will discuss the motivation behind this thesis and the contributions of this thesis, as well as lay out a roadmap for the rest of the thesis.

1.1 Digital Image Compositing and Matting: A Mathematical Framework

A digital image I is a rectangular array of *pixels*. In color images, each pixel is usually a three-dimensional vector, denoting a point in three-dimensional color space. There are many different color spaces, such as RGB (Red-Green-Blue), HSV (Hue-Saturation-Value), Lab (Light and two opponent color spaces); for the purpose of this thesis, we assume RGB color space – so that the three dimensions of each pixel represent the Red, Green and Blue components of its color – but the concepts and algorithms discussed will transfer easily to different color spaces. We will refer to the Red, Green and Blue components of every pixel in the image collectively as the Red, Green and Blue *channels* of the image. For each pixel, each component is 8 bits in size and thus each of its Red, Green and Blue components lie in the set $\{0, 1, 2, \ldots, 255\}$.

In the standard Digital Compositing framework, introduced by Porter and Duff [25], a new composited image I is composited from three elements: A foreground image F, a background image B, and an alpha matte α. F and B are both color images, while α is a grayscale mask such that $0 \leq \alpha \leq 1$ for all pixels in the image. These three elements are related to the image I through the *Compositing Equation*[1], also referred to as the *over* operator:

$$I = \alpha F + (1 - \alpha)B \tag{1.1}$$

Thus, the image I is a linear interpolation between the foreground and background

[1]Although F, B and α are all arrays, the multiplication in equation (1.1) should be interpreted as pointwise multiplication. For example, at each foreground pixel, the Red, Green, and Blue values are all multiplied by the value of α at that pixel.

images F and B, with the degree of interpolation determined by the alpha matte. When $\alpha = 1$, the pixel displayed is that of the foreground; similarly, when $\alpha = 0$, the pixel displayed is that of the background and there is a smooth transition between the two for fractional values of α. Thus, another interpretation of α is of the *foreground opacity* or more precisely – following Porter and Duff [25] – the partial foreground coverage: When α is high, the foreground is relatively opaque and has a high coverage, and when α is low, the foreground is relatively transparent and has a low coverage.

Now that we have formalized the compositing and matting framework, we can state the Digital Compositing and Matting problems:

- **Compositing:** Given foreground F, background B, and alpha matte α, compute the composited image I.
- **Matting:** Given image I, separate it into its components F, B and α.[2]

The compositing problem is trivial and is easily solved by applying the compositing equation (1.1). The matting problem, however, is non-trivial. It is in fact inherently ill-posed and underconstrained and thus has no "correct" solution: There are seven unknowns – the three color channels of each of the foreground and background images and the alpha matte – but only three equations – equation (1.1) for each of the three color channels. As such, further approximations and assumptions will have to be used in order to obtain a solution.

In blue-screen matting, the background is assumed to be a known constant color, and as such there is no need to estimate the three background color channels as they are known. With this assumption, there are now four unknowns and three equations. A further assumption is now required to compute the matting solution. There are many

[2]For most practical purposes, only α and F need to be computed: When compositing a foreground element onto a new background, the background B of the original image is completely discarded. However, when α and F are both known, computing B from the image I is trivial, following equation (1.1).

possible assumptions of various complexity and power; for the sake of brevity, we will only briefly mention Vlahos's first solution from [32].

Let us quickly introduce some notation: Let F_R, F_G and F_B denote the Red, Green and Blue channels of the foreground and B_R, B_G and B_B the corresponding channels for the background. In blue screen matting with a blue background, $B_R = B_G = 0$, and B_B is set by the user (for a pure blue, $B_B = 255$). One assumption that leads to Vlahos's first solution [32] is that $F_G = aF_B$ for a user-tuned setting of a – thus there is a constant ratio of green to blue. As [32] notes, this assumption is quite valid in movie settings as it holds for gray colors (such as spaceships) and flesh colors. With this additional assumption, the matting problem is solvable, and Vlahos's solution gives

$$\alpha = 1 - \frac{1}{B_B}\left(F_B - \frac{1}{a}F_G\right)$$

where as stated before B_B and a are user-defined parameters. Since the above formula may not lead to a value of $\alpha \in [0, 1]$, the resulting solution is clamped to $[0, 1]$. Different assumptions about the colors will provide different matting solutions in blue-screen matting; for a fuller overview, consult [32].

A different technique introduced in [32] – but is quite implausible in practice – is to obtain two pictures of the foreground element against two different known backgrounds, assuming that the alpha matte is the same in these two pictures. By putting the foreground elements against two different known backgrounds, additional information and constraints are provided which makes the matting problem feasible.

1.1.1 Natural Image Matting

However, in many settings – especially non-industrial and amateur settings – the background is not known. Furthermore, the background may not be a constant color;

in the case of photos, for example, the background is often multi-colored. In addition, the user may not want to extract all the objects in the image; if, for example, the user wishes to extract a single face from a group photograph, all the other people in the photograph also become part of the "background".

All the above facts mean that the techniques of blue-screen matting will not be applicable in the general case where we are presented with a *single image* and wish to obtain an alpha matte to extract a particular foreground element. This problem, which is the main focus of this thesis, is known as **natural image matting**. Natural image matting algorithms generally require guidance from the user to help generate the alpha matte; usually the user helps classify certain regions as foreground and background, so that the algorithm can "learn" to distinguish which areas belong to the foreground elements and which belong to the background.

There has been much prior work in natural image matting (a brief summary is provided in chapter 2), and the algorithms differentiate themselves in different ways. One way in which the algorithms distinguish themselves is the user input that is provided to them other than the image itself. There are two main types of user input: Trimaps and Scribbles.

In a trimap, the user supplies a separate grayscale bitmap that separates the original image into three sub-images: An area that is known to be entirely foreground, an area that is known to be entirely background, and an unknown region. The algorithm is generally constrained to set $\alpha = 0$ in the background region, $\alpha = 1$ in the foreground region, and its main goal is to infer α in the unknown region, using the knowledge about the known foreground and background pixels. An example of a trimap is shown in figure 1-2.

In scribbles, the user simply marks certain areas on the original image as foreground or background – with the rest of the image unknown – and the algorithm uses this

Figure 1-2: Example of a trimap. From left to right: Original Image, Trimap, Alpha Matte generated using Bayesian Matting [9]. In the trimap, black represents background, white foreground, and gray unknown.

limited information to compute an alpha matte. While there are many similarities between trimaps and scribbles – indeed, a trimap can be used as a scribble, albeit a very detailed one – we distinguish the two because trimaps are generally far tighter and provide much more information than scribbles; the area of the unknown region is far larger in scribbles than in a trimap. Figure 1-3 shows an example of a scribble input.

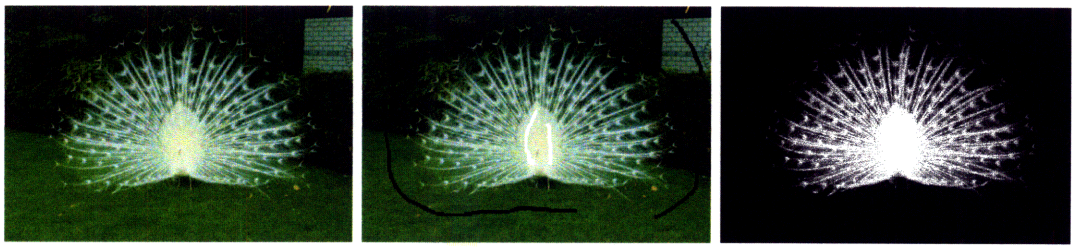


Figure 1-3: Example of scribbles. From left to right: Original Image, Scribbles on Image, Alpha Matte generated using Levin et al's Closed Form Matting Solution [16]. In the scribbles, black denotes background and white denotes foreground.

A related important distinction – which we will allude to over the course of this thesis – differentiating matting algorithms involves their formulation and use of *data* and *connectivity* (or smoothness). Data involves the use of color-based information, such as the local foreground and background color distributions near a pixel. Connectivity involves the use of smoothness constraints that try to enforce local smoothness in the alpha matte. As a general guideline – although of course this is not true in all cases – trimap-based algorithms generally emphasize the data term, while scribble-

based algorithms favor the connectivity term and use propagation mechanisms with smoothness constraints to estimate the alpha matte. We will see some examples of these in chapter 2.

One advantage of trimap-based algorithms is that they are designed to use the additional information provided, and thus can be more accurate, especially in regions of sharp foreground/background boundaries where α jumps from 0 to 1. In contrast, scribble-based algorithms often use propagation mechanisms that tend to prefer a smooth transition for α from 0 to 1 and thus can be less accurate in such regions. Scribble-based algorithms are not designed to use the additional information provided by a trimap efficiently: While they tend to be more accurate for extremely loose trimaps, as the trimap becomes tighter eventually the trimap-based algorithm becomes more accurate (see [35] for an example). Moreover, in regions where the foreground/background transition is sharp, the additional effort required to generate a tight trimap, as compared to mere scribbles, is relatively low.

However, in images where the foreground/background boundary is blurred and extremely textured, such as in figures 1-2 and 1-3, the additional user effort needed to create a tight trimap may become significant. Furthermore, in these images, it is more natural for α to transition smoothly from 0 to 1 at the foreground/background boundary; in these cases the scribble-based algorithms can be more efficient and appear more natural.

This thesis will introduce Segment-Based Matting – a new trimap-based algorithm motivated by the problems of the Bayesian Matting [9] algorithm (which is also trimap-based). The next section will briefly discuss the motivations behind this thesis as well as its contributions.

1.2 Thesis Motivation and Contributions

Bayesian Matting [9] is a popular trimap-based natural image matting algorithm that estimates each pixel separately; for each pixel, it builds local foreground and background color distributions from nearby foreground and background pixels and estimates a solution to the matting problem using a Bayesian approach. Although Bayesian Matting performs reasonably well in practice, it has some problems. We provide here a brief informal summary of some of its problems; details can be found in section 3.2:

1. The run-time can be fairly long, because its numerical optimization procedure can have an empirically slow rate of convergence.

2. Bayesian Matting can assign fractional alphas to pixels that should clearly be foreground or background pixels and thus have $\alpha = 1$ or $\alpha = 0$; thus the mattes generated are less sharp than optimal and their accuracy is reduced. One reason for this is that it tends to favor the mean of the foreground and background distributions and thus does not consider the effect of shading.

3. Bayesian Matting considers each pixel independently and thus can generate a fairly discontinuous alpha matte, with patches of high alpha interspersed in regions of low alpha. This is especially true in regions where the local foreground and background color distributions are very similar.

This thesis contributes with regard to this aspect in two ways:

1. We perform an in-depth study of many of the *details* of Bayesian Matting that result in such problems.

2. Motivated by Bayesian Matting and its problems, we propose Segment-Based Matting – a *new* color-based statistical algorithm. Segment-Based Matting solves many of the problems of Bayesian Matting. Firstly, it has a closed form solution which improves runtime efficiency. Secondly, it models color distributions as lines and *does not favor the means of these line distributions*; this

models shading and helps increase the sharpness and accuracy of the alpha matte. Empirically, it also reduces the discontinuity of the alpha matte.

A problem many matting algorithms have is distinguishing the foreground and background when the foreground and background color distributions are very similar. Since these algorithms rely in part on the foreground and background having different color distributions, the resulting matte can be fairly arbitrary and ambiguous. Propagation-based algorithms that try to enforce smoothness on the alpha matte can reduce the effect of this problem, but the resulting matte can still be incorrect, especially with regard to the contours of the foreground and background boundaries.

This thesis contributes to this aspect in two ways:

1. We propose a new approach where matting algorithms will first attempt to classify pixels as either foreground, background, or ambiguous – being both in the foreground and background – before performing estimation. This has a few benefits: It sharpens the matte by ensuring most pixels have $\alpha = 1$ or $\alpha = 0$, it reduces the computation effort needed by identifying "obvious" cases, and it allows us to identify the areas the algorithm will have a problem on.

2. We use a different image processing technique to resolve ambiguities and fill in the ambiguous areas. Such techniques include inpainting [3] – the restoration of defective regions in digital images – and texture synthesis [11, 12].

Informally, the above approach first tries to use color information to estimate the matte. When estimation based on color is inaccurate and ambiguous, it uses a different kind of digital image processing technique that is less reliant on color and more on other visual aspects such as contour continuation and texture similarities to fill in the ambiguous areas.

Finally, we will demonstrate that Segment-Based Matting and our new approach of using inpainting to resolve ambiguous areas in alpha mattes compares favorably to

some existing matting approaches on a variety of images. From our results, we propose the following important concept: Many existing matting algorithms generally use two principles, which we have previously described as data and connectivity: Data – which we will refer to as color – suggests that the global or local color distribution of foreground and background should help determine the matte. Connectivity – which we will refer to as smoothness – suggests that the the alpha matte should be smooth in some sense. Some algorithms – such as Bayesian Matting – use only color, some – such as Levin et al's Closed-Form laplacian matting solution – use only smoothness, while some – such as Wang and Cohen's Robust Matting [35] – try to combine color and smoothness. In this thesis, we claim that none of these approaches are entirely optimal: For example, the Bayesian Matting solution can result in disjointed mattes, while the Closed-Form solution can result in mattes that are overly smooth at foreground/background boundaries, which is inaccurate if the foreground/background boundary is sharp. We propose a two-fold approach that emphasizes the use of color over smoothness: Smoothness only becomes important if the color distribution is ambiguous. While it is possible to adjust the extent we enforce smoothness based on the ambiguity of the color distribution – and we will note as such in the conclusion of this thesis – the approach we propose here uses the color distribution to determine the matte, and *only if the color distribution is ambiguous* do we try to use smoothness. This allows the matte to be sharp when the foreground/background boundary is sharp, and generate a smoother matte when there is a smoother transition from the foreground to the background.

Remark 1.1. As we have briefly mentioned in our introduction to matting, one important industrial application of matting is in the film industry, where matting is performed on a series of film frames, and not on a single static image frame. **Video matting** is also the topic of a large amount of research, and many existing matting algorithms – including Bayesian Matting – have been adapted for video matting. However, video matting is beyond the scope of this thesis; we will only discuss our algorithms on static image matting.

The remainder of this thesis will proceed as follows. Chapter 2 will briefly summarize some prior work in natural image matting. Chapter 3 will summarize Bayesian Matting as well as present our in-depth study of some of its problems. Chapter 4 presents Segment-Based Matting, our color-based statistical matting algorithm. Chapter 5 discusses our approach for using techniques such as inpainting and texture synthesis to resolve color ambiguities in matting. Chapter 6 presents the results of running our Segment-Based Matting algorithm from chapter 4 and incorporating our approach from chapter 5, and discusses possible implications of these results. Finally, chapter 7 will conclude and propose some future work.

We have also provided three appendices that lay the mathematical groundwork – assuming a working knowledge of first-year college calculus – for our work. Appendix A provides a brief introduction to elementary linear algebra, including a section on basic transformations in two dimensions in section A.4 and a section on the application of homogenous coordinates to line intersection in A.5. Appendix B provides a brief introduction to basic statistics and estimation procedures, including a very brief overview of Markov Random Fields (section B.6) as it pertains to this thesis. Finally, appendix C builds on some of the material introduced in appendices A and B to briefly introduce the technique of Principal Component Analysis, which is used in the Segment-Based Matting algorithm.

Chapter 2

Some Prior Work in Natural Image Matting

In this chapter, we will briefly introduce some prior algorithms in Natural Image Matting. We first introduce the approach by Ruzon and Tomasi in section 2.1; this is an approach that has inspired many trimap-based statistical approaches. We next briefly mention Bayesian Matting in section 2.2; Bayesian Matting is the subject of chapter 3 and its problems motivate the development of our Segment-Based Matting algorithm in chapter 4. We introduce Levin et al's closed form laplacian matting algorithm in section 2.3. Section 2.4 introduces the graph cut and grab cut algorithms. Following that, we briefly mention more complicated hybrid approaches such as robust matting in section 2.5 and the iterative optimization approach in section 2.6; these approaches use elements from some of the other algorithms introduced. We then briefly introduce geodesic matting in section 2.7; this algorithm uses a different approach and follows a geodesic framework. Finally, we will list some problems of natural image matting in section 2.8.

2.1 Ruzon-Tomasi

The trimap and color-based method introduced by Ruzon and Tomasi [28] is a statistical approach that has inspired many other statistical sampling-based algorithms

such as Bayesian Matting.

Each unknown pixel is estimated independently: For each unknown pixel, the nearby (in a spatial sense) foreground and background pixels (as labelled in the trimap) are sampled and several isotropic (spherical) Gaussians are fitted to these foreground and background distributions based on different criteria. The number of foreground and background distributions are identical and paired up, so that there are n (foreground, background) distribution pairs. For the jth pair of Gaussian distributions, $1 \leq j \leq n$, we denote the foreground distribution as $N(\mu_{f,j}, \sigma_{f,j}^2)$ and the background distribution as $N(\mu_{b,j}, \sigma_{b,j}^2)$. The jth foreground-background pair is also given a confidence level a_j, which is normalized such that $\sum_{j=1}^{n} a_j = 1$. Note that while $\mu_{f,j}$ and $\mu_{b,j}$ are vectors in three-dimensional color space, $\sigma_{f,j}$ and $\sigma_{b,j}$ are scalars as the fitted Gaussians are isotropic.

The solution of α for pixel I_i in image I is then given as

$$\alpha = \arg\max_{\alpha} \sum_{j=1}^{n} a_j f_{normal}(I_i; \alpha\mu_{f,j} + (1-\alpha)\mu_{b,j}, \alpha\sigma_{f,j}^2 + (1-\alpha)\sigma_{b,j}^2)$$

where f_{normal} is the multivariate normal probability density function. The above equation is solved numerically. To gain some intuition into the above formulation, consider the case when $n = 1$, so that there is only one foreground/background pair. In this case, the solution for α is obtained by "projecting" the pixel I_i into a line connecting the foreground and background distributions.

2.2 Bayesian Matting

Bayesian Matting is a trimap and color-based statistical algorithm that models the image, foreground, background and alpha matte using a joint probability distribution and solves the matting problem using a Bayesian approach. Bayesian Matting is studied in detail in chapter 3, so we will only give a very brief overview here.

Like in the Ruzon-Tomasi approach described in the previous section, Bayesian Matting estimates each unknown pixel independently and samples the nearby foreground and background pixels. It then partitions the foreground (background) pixels into multiple clusters and fits a Gaussian to each cluster; unlike in Ruzon-Tomasi, the Gaussians fitted are anisotropic, so they are not necessarily spherical. For each foreground/background pair of clusters, Bayesian Matting performs Maximum A Posteriori estimation to solve the matting problem; this gives multiple possible solutions, one for each foreground/background pair of clusters. Bayesian Matting then chooses the solution that has the highest Maximum A Posteriori likelihood.

2.3 Closed Form Matting

The closed form laplacian matting algorithm introduced by Levin et al [16] is a scribble and smoothness-based algorithm that uses a propagation approach based on smoothness assumptions to solve for the alpha matte.

From the compositing equation (1.1), we obtain $I_j = \alpha_j F_j + (1 - \alpha_j) B_j$ for pixel j. This can be rewritten to give a solution for α_j as

$$\alpha_j = a_j I_j + b_j \tag{2.1}$$

where a_j and b_j are constants that depend on the values of F_j and B_j.

To provide an understanding of this approach, we will start – as in [16] – by assuming that the image I is grayscale. The key assumption in the grayscale case is a local smoothness property: In a small window around any unknown pixel I_j, the foreground and background are essentially constant and equal to F_j and B_j respectively. Therefore, any discontinuities in the image must be caused by discontinuities in α.

Under this assumption, for all pixels i in a window w_j around pixel I_j, $a_i = a_j$ and $b_i = b_j$ since the values of a and b are dependent on F and B, which are constant in this window by assumption. Hence, following equation (2.1), we obtain

$$\alpha_i = a_j I_i + b_j \tag{2.2}$$

for all pixels $i \in w_j$. Naturally, this assumption cannot hold exactly over the entire image, but we wish to find α for all pixels such that this assumption is as accurate as possible. Thus we wish to find α, a and b (over the entire image I) that minimizes the cost function

$$J(\alpha, a, b) = \sum_{j \in I} \left(\sum_{i \in w_j} (\alpha_i - a_j I_i - b_j)^2 + \epsilon a_j^2 \right) \tag{2.3}$$

In the above formulation, the regularization term ϵa_j^2 has been added. A higher value of ϵ penalizes a non-zero value of a_j more. Following equation (2.1), a value of $a_j = 0$ implies that $\alpha_j = b_j$ is independent of the observed image value I_j, and this enforces smoothness of α under the assumption that $F_j = F_i$ and $B_j = B_i$ – and thus $\alpha_j = b_j = b_i = \alpha_i$ – for pixels j and i in each other's local window. Hence, a higher value of the ϵ parameter makes the generated alpha matte smoother.

The key insight of the closed form laplacian matting algorithm is that we can eliminate the terms a and b in equation (2.3) and rewrite it as

$$J(\alpha) = \alpha^T L \alpha \tag{2.4}$$

where α is a vector of all the unknown alpha values and L, the *matting laplacian*, can be entirely determined by the observed image, the window size, and the ϵ parameter; in other words, it is a constant matrix. The problem of finding α that minimizes the cost function (2.4), subject to the constraints imposed by the scribbles, can be solved in closed form and this is the gist of the algorithm.

The above description applied to grayscale images; Levin et al extend it to color images by observing that under the assumption that the foreground (background) color distribution lies on a straight line in color space (this is the color line model, also briefly discussed in section 4.2), analogues to (2.3) and (2.4) can be obtained, and the analogous problem can be solved to find α.

2.3.1 Spectral Matting

The Spectral Matting approach in [17] performs further analysis of the matting laplacian L. First of all, they draw an analogue between the matting laplacian and *node affinities* in graph segmentation models. In these graph segmentation models, edges between nodes represent *affinities* between the nodes and the goal is to segment the graph by cutting edges – subject to certain criteria – such that as far as possible, nodes with a high affinity with each other remain connected. An affinity matrix A can be built for such a graph, such that $A_{i,j}$ corresponds to the affinity between nodes i and j. The Spectral Matting approach considers the pixels as nodes and the matting laplacian as the affinity matrix.

One well-known technique for cutting a graph, given an affinity matrix, is the normalized cuts technique given in [31], which uses the eigenvectors of the affinity matrix. In a similar fashion, the Spectral Matting approach tries to utilize the eigenvectors of the matting laplacian, and generate an alpha matte from a linear combination of these eigenvectors. Details can be found in [17].

2.4 Graph Cut

The Graph Cut [5] algorithm is an approach that combines color and smoothness. It generates a hard segmentation of grayscale images – color images are first converted to grayscale – so that the resulting alpha matte only contains $\alpha = 1$ and $\alpha = 0$ values. The algorithm views the image as a lattice graph – where pixels are connected

to adjacent pixels – and separates the pixels into *foreground* and *background* using a *minimum cut* approach.

Two nodes are added to the lattice graph: A foreground node F and a background node B. Both of these nodes are connected to every pixel node in the graph, but not to each other. The algorithm finds the minimum cut in the graph that separates F and B – a subset of edges that disconnect F and B such that the total weight of all these edges is minimal. After removing the cut, the set of vertices that are connected to F are given $\alpha = 1$ and the set of vertices that are connected to B are given $\alpha = 0$. Minimum cut is a well-studied problem that has efficient algorithms [10]; hence all that remains is to describe the edge weights such that the resulting cut provides a desirable result.

In the lattice graph, the weight between adjacent pixels i and j is given by $W(i,j) \propto \exp\left(-\frac{(I_i - I_j)^2}{2\sigma^2}\right)$ where I_i and I_j are the scalar (grayscale) intensity values of pixels i and j in the image and σ is a user-defined parameter. This formulation gives higher weight to edges connecting adjacent pixels with similar intensities; therefore it is more likely that they will belong to the same component (foreground or background) in the resulting cut.

For any unknown pixel, the weight between pixel i and the foreground node F is given by $-\log P(I_i|B)$, where $P(I_i|B)$ is the probability that a pixel with intensity I_i belongs to the background, and similarly the weight between pixel i and the background node B is given by $-\log P(I_i|F)$. The probabilities $P(I_i|F)$ and $P(I_i|B)$ are obtained by tabulating a *global histogram* of the intensity values of the pixels that were marked as foreground and background by the user. Since the image is grayscale, the number of possible intensity values is small, and hence if the number of marked pixels is large – or equivalently if the number of unknown pixels are small – the histograms provide a good approximation to the probabilities.

Any pixel marked as foreground has an extremely large weight with F and zero weight with B, and similarly any pixel marked as background has an extremely large weight with B and zero weight with F. The exact magnitudes of these weights – discussed in [5] – ensure that in the resulting minimum cut, all pixels marked as foreground will remain connected with F and all pixels marked as background will remain connected with B, thus preserving user-defined constraints.

If pixel i remains connected to the foreground, it is cut from the background with cut edge weight $-\log P(I_i|F)$, and similarly, if it remains connected to the background, it is cut from the foreground with cut edge weight $-\log P(I_i|B)$. Hence, the minimum cut problem attempts to solve

$$\alpha = \arg\min_{\alpha} \sum_{i \text{ in foreground, } j \text{ in background}} W(i,j) - \sum_i \log P(I_i|\alpha_i)$$

where $P(I_i|\alpha_i = 0) = P(I_i|B)$ and $P(I_i|\alpha_i = 1) = P(I_i|F)$ (recall that graph cut provides a hard segmentation and thus $\alpha_i \in \{0, 1\}$). The first term $W(i,j)$ can be viewed as an interaction term and the second term is a likelihood term; hence we can interpret graph cut as maximizing the likelihood of the foreground/background assignment adjusted by a correction term for interaction.

2.4.1 Grab Cut

The Grab Cut algorithm [27] extends the Graph Cut algorithm to color images.

In color images, the space of possible pixel values is much larger than in grayscale and tabulating histograms no longer forms a good approximation to probabilities. To calculate the assignments of pixels to foreground or background, the Grab Cut algorithm performs the following two steps repeatedly:

1. Calculate probabilities $P(I_i|F)$ and $P(I_i|B)$ using previously obtained assignments of α. In the first iteration, α is obtained using the trimap, and unknown

pixels are arbitrarily assigned $\alpha = 1$.

2. Perform Graph Cut as described in the previous section to solve for α using the probabilities derived in the first step.

The first step fixes the values of α and solves for the probabilities; the second step fixes the probabilities and solves for α. To estimate the required probabilities in the first step, [27] models the foreground and background as a mixture of anisotropic Gaussian Distributions, and estimates the distributions using a variant of hard Expectation-Maximization (EM) in this Gaussian Mixture Model [18].

Once the estimation process is done, [27] then performs a further regularization step to smooth the matte and allow fractional values of α.

2.5 Robust Matting

The robust matting approach [35] is a trimap-based approach, combining color and smoothness, that uses a graphical model similar to graph cut, with pixels forming a lattice graph and additional foreground and background nodes (F and B) that are each connected to all pixel nodes. There are two major differences:

1. The edge weights in the graph and the way they are calculated.

2. The problem to solve in this graph.

As in the case of graph cut, we need to define two types of edge weights in the graph: The interaction weights between neighboring pixels – which control the smoothness of the matte – and the weights between the foreground (background) node and the pixel nodes.

For the interaction weights between neighboring pixels, [35] uses the affinity as given by the matting laplacian from Levin et al's closed form laplacian matting solution (see section 2.3 and [16]), so that $W(i,j) = L_{i,j}$ where L is the matting laplacian

matrix.

The weights between the foreground (background) node and the pixel nodes are calculated separately for each pixel. For each unknown image pixel, nearby foreground and background samples (from the trimap) are obtained. Instead of building a distribution over these samples, [35] tries every (foreground, background) pixel pair from these samples. For each (foreground, background) pair, an alpha value and a confidence level is calculated – the alpha value is obtained simply by projecting the unknown image pixel onto the line formed by the (foreground, background) pixel pair, while the confidence level takes into account, among other factors, both the foreground–background distance as well as the distance of the unknown image pixel from its projection.

The three (foreground, background) pairs with the highest confidences are then chosen, and the overall alpha estimate α and overall confidence level f of this pixel are taken as averages of the values from these pairs. The weight between the unknown image pixel and the foreground and background nodes are then given by

$$W(F) = \gamma[f\alpha + (1-f)\delta(\alpha > 0.5)]$$
$$W(B) = \gamma[f(1-\alpha) + (1-f)\delta(\alpha < 0.5)]$$

where δ is a boolean function that returns 0 or 1. γ is a parameter that adjusts the relative importance of the weight with the foreground (background) node; the higher γ is, the more important this weight is, and the less important the interaction weights are; thus the obtained alpha matte will be less smooth with higher γ.

The problem that [35] solves in this graph is not a graph cut, but a *random walk* question: Suppose we start a random walker at the node corresponding to a given pixel, and at each step he has a probability of transitioning to any neighbor that he shares an edge with; the probability of transitioning to a given neighbor is propor-

tional to the weight of the corresponding edge. What is the probability that he enters the foreground node before he enters the background node?

This probability is higher for pixels that are more likely to be foreground, and at the same time it can take any real value between 0 and 1; [35] hence uses the solution for this question as the values of α. This problem can be solved as a linear system, using a graph laplacian matrix; details can be found in [14].

2.6 Iterative Optimization

The iterative optimization approach [34] is a scribble-based algorithm, utilizing both color and smoothness, that gradually propagates the alpha matte outwards from the user-provided scribbles, using a Markov Random Field (MRF) formulation (a brief introduction to MRFs can be found in Appendix B.6).

A brief overview of the approach is as follows: A set U_c of all pixels that have been previously estimated – including the pixels marked by the scribbles – is kept, and each of these pixels is given a confidence value, indicating how confident the algorithm is of its estimate for that pixel. Initially, U_c consists only of the marked pixels, and they are all given a confidence level of 1, indicating that their confidence is perfect.

The optimization approach proceeds as follows. The following steps are repeated until the set U_c remains constant and the confidence level of pixels no longer increases:

1. Add any pixels that are spatially near the set U_c to U_c if they were not previously in U_c. Each of the newly added pixels has a confidence of 0.

2. Take all pixels in U_c – including pixels that were previously estimated – that have a sufficiently low confidence level. This allows pixels that were estimated in previous steps to be re-estimated, if new pixels that were added can provide new information about the alpha matte. We denote this set U_k.

3. Re-estimate α, F and B and update the confidence of the estimated pixels in U_k.

In the estimation step, the pixels are modelled as a lattice graph as previously, but instead of solving a graph cut or a random walk problem as in previous sections, the graph is viewed as a Markov Random Field with corresponding estimation problem (see appendix B.6)

$$\alpha = \arg\max_{\alpha} \prod_{\text{All pixels } i \in U_k} \Psi_{\alpha_i}(\alpha_i) \prod_{(i,j) \text{ are neighboring pixels, } i,j \in U_k} \Psi_{\alpha_i,\alpha_j}(\alpha_i, \alpha_j)$$

where $\Psi_{\alpha_i}(\alpha_i)$ is the *data term* and $\Psi_{\alpha_i,\alpha_j}(\alpha_i, \alpha_j)$ is the *connectivity term.* Once the data and connectivity terms are specified, the MRF problem can be solved using loopy belief propagation (see appendix B.6) to obtain α; hence all that remains is to specify the data and connectivity terms.

The data term is obtained as follows: For pixel i, sample the foreground and background pixels in its immediate neighborhood, including pixels that were previously estimated; pixel j in the neighborhood of pixel i is considered foreground if α_j is greater than the previously estimated value of α_i and similarly pixel j is considered background if α_j is less than the previously estimated value of α_i. Each foreground (background) pixel j is given a weight w_j which is a combination of the spatial distance of j from i as well as the confidence of the estimate of α_j. Next, the likelihood of a given value of α_i is computed as

$$L(\alpha_i) \propto \sum_{j \in \text{ foreground sample}} \sum_{k \in \text{ background sample}} w_j w_k \exp\left(-\frac{||I_i - \alpha_i F_j - (1-\alpha_i)B_k||^2}{2\sigma_{\alpha_i}^2}\right)$$

where I_i is the color of pixel i and F_j and B_k are the foreground and background values of the foreground pixel j and the background pixel k. The covariance σ_{α_i} is calculated from interpolating the covariance of the foreground and background distributions: $\sigma_{\alpha_i} = \alpha_i \sigma_f + (1-\alpha_i)\sigma_b$, where σ_f and σ_b are the foreground and background covariances. The likelihood is calculated for K different values of $\alpha_i \in$

$\{\alpha_{i,1}, \ldots, \alpha_{i,K}\}$. Finally, the data term is given by

$$\Psi_{\alpha_i}(\alpha_i) = \frac{L(\alpha_i)}{\sum_{j=1}^{K} L(\alpha_{i,j})}, \alpha_i \in \{\alpha_{i,1}, \ldots, \alpha_{i,K}\}$$

The connectivity term is simply given by

$$\Psi_{\alpha_i,\alpha_j}(\alpha_i, \alpha_j) = \exp\left(-\frac{(\alpha_i - \alpha_j)^2}{2\sigma_c^2}\right)$$

where σ_c is a user-defined parameter. This connectivity term tries to enforce smoothness of the alpha matte; the greater σ_c is, the lower the degree of smoothness enforced.

Once the values of α are estimated using the MRF, the values of F_i and B_i for each pixel $i \in U_k$ are estimated as follows: Pick the pair of pixels j in the foreground sample and k in the background sample from the neighborhood of pixel i such that $||I_i - \alpha_i F_j - (1 - \alpha_i)B_k||$ is minimized, and pick $F_i = F_j$ and $B_i = B_k$. Finally, the confidence of the pixel i is updated as being equal to $\sqrt{w_j w_k}$, where j and k are the foreground and background pixels that were picked to form F_i and B_i.

2.7 Geodesic Matting

Geodesic Matting [2] is a different color and smoothness-based approach to scribble-based matting that uses shortest distance calculations to classify pixels.

From the scribbles, *global* foreground and background probability densities P_F and P_B are calculated using a non-parametric kernel density estimation procedure. For a given pixel i with intensity I_i, its likelihood of being in the foreground is given as $L_F(I_i) = \frac{P_F(I_i)}{P_F(I_i)+P_B(I_i)}$ and its likelihood of being in the background is given as $L_B(I_i) = 1 - L_F(I_i)$. Now define the *path weight* between two adjacent pixels as ∇L_F, the gradient of the foreground likelihood. This defines a weighted graph with edge weights. For a given pixel i, the foreground distance $D_F(i)$ is then the shortest distance in this graph from i to any pixel marked in the scribble as foreground, and

similarly the background distance $D_B(i)$ is the shortest distance from i to any pixel marked in the scribble as background.

From these values, α_i for pixel i is calculated as:

$$\omega_F(i) = D_F(i)^{-r} L_F(I_i)$$
$$\omega_B(i) = D_B(i)^{-r} L_B(I_i)$$
$$\alpha_i = \frac{\omega_F(i)}{\omega_F(i) + \omega_B(i)}$$

where r is a user-specified parameter. The higher r is, the greater the distance from the foreground or background matters; when $r = 0$, the calculated α value is simply its foreground likelihood L_F.

To obtain F_i and B_i for pixel i, nearby foreground and background pixels (as marked by the scribbles) are sampled, and the (foreground, background) pair F and B that minimizes $||I_i - \alpha_i F - (1 - \alpha_i)B||$ are used as the values of F_i and B_i.

2.8 General Problems with Matting

Natural Image Matting is an area with many potential problems, because users do not have complete control over the circumstances the image was obtained in. We will briefly list some of these potential problems. Most matting algorithms try to attack a subset of these problems, but none – as far as we know of, and including the algorithm described in this thesis – solves all these problems adequately. The problems we will mention are:

1. **Smoothness of the image matte.** One fundamental assumption behind many matting algorithms is that the alpha matte should be somewhat smooth. There is always a trade off between smoothness and accuracy of the matte at sharp foreground/background boundaries: The alpha values should change rapidly at these boundaries, but this is not a smooth change. It is arguable

that in these cases it is far more desirable to entirely discard smoothness, to allow for a sharp boundary instead of a smooth transition. Furthermore, if the foreground element contains small holes, it is a challenge to prevent the matting algorithm from smoothing over these holes in the alpha matte.

2. **Sudden Changes in foreground and background color distributions in a local area**. The foreground and background distributions are not homogenous throughout the entire image. In an extremely colorful image, these can change extremely rapidly in a local area. Matting algorithms must be able to handle such changes.

3. **Similar foreground and background color distributions**. Most matting algorithms use foreground and background color distributions in a local area to solve for the alpha matte, using the underlying assumption that these distributions are distinct in some way. What if these distributions are very similar?

4. **Shading**. Often, especially at a foreground/background boundary, different shades of the same color can be found; these can be caused by uneven lighting throughout the image. Shading can cause problems for matting algorithms; an example can be found in chapter 3 when we examine Bayesian Matting.

5. **Shadows**. Light shadows caused by background lighting or camera flashes can distort images and cause dark shades throughout the foreground element; this can cause discontinuities if the background is dark and the matting algorithm estimates these pixels as belonging to the background.

6. **Compression Artifacts**. Many digital cameras compress images and store them in lossy image formats such as JPEG; these can cause local artifacts in the image and can cause problems for matting algorithms that assume the image is generally flawless.

We postulate that one reason why many of these problems have not been explicitly examined in the literature is that enforcing smoothness of the alpha matte can help

mitigate the effect of many of these problems, such as similar foreground and background distributions, shading and shadows. However, as we have noted here and we will see in chapter 6, enforcing smoothness can lead to its own problems. In this thesis, we propose a different approach that tries to solve the matting problem using color information and only enforces smoothness when we detect some of the problems noted here; this allows sharp mattes at sharp foreground/background boundaries.

Chapter 3

Bayesian Image Matting

In this chapter, we will perform a detailed study of the popular Bayesian Matting [9] algorithm, focusing on its problems. Bayesian Matting is a sampling-based algorithm, utilizing a trimap input by the user, that uses Maximum A Posteriori (MAP) estimation in a Bayesian Framework (see Appendix B.5) to estimate α, F and B.

Section 3.1 will describe the Bayesian Matting Algorithm, as well as provide a brief interpretation of the algorithm. Next, section 3.2 will perform an in-depth study of some problems of the Bayesian Matting algorithm. Finally, section 3.3 will discuss a few minor changes to Bayesian Matting that will address some of these problems; this modified Bayesian Matting algorithm will form part of the basis of our comparisons in chapter 6.

3.1 A Description of Bayesian Matting

In Bayesian Matting, the image I, foreground element F, background element B, and alpha matte α are linked through a joint probabilistic distribution; each pixel is considered to be independent of the rest and hence every pixel is estimated separately. Using this joint probabilistic distribution, the algorithm uses Bayes's law to form an a posteriori probability distribution of the unknown parameters F, B and α and perform MAP estimation.

To be precise, letting $P(A)$ denote the probability of variable A and $P(A|B)$ denote the probability of variables A conditioned on the values of variables B, we may use Bayes's law to obtain:

$$P(F, B, \alpha|I) = \frac{P(I|F, B, \alpha)P(F, B, \alpha)}{P(I)} = \frac{P(I|F, B, \alpha)P(F)P(B)P(\alpha)}{P(I)}$$

where in the Bayesian matting framework F, B and α are independent of each other and hence the second equality. Taking logs, and letting $L(.) = \log P(.)$, we obtain

$$L(F, B, \alpha|I) = L(I|F, B, \alpha) + L(F) + L(B) + L(\alpha) - L(I)$$

In the matting framework, I is observed and fixed and is not a parameter to optimize over, hence $L(I)$ is a constant. In the Bayesian Matting framework, $L(\alpha)$ is a constant over all α – in the sense that each possible α is considered uniformly likely. Hence in the MAP framework, we may discard both $L(\alpha)$ and $L(I)$. The MAP problem to solve is then

$$(F, B, \alpha) = \arg\max_{F,B,\alpha} L(I|F, B, \alpha) + L(F) + L(B) \tag{3.1}$$

The Bayesian Matting Algorithm models each of the three separate terms in (3.1) as Gaussian distributions. The likelihood $L(I|F, B, \alpha)$ is modelled as a univariate Gaussian with mean given by the compositing equation (1.1):

$$L(I|F, B, \alpha) = -\frac{||I - \alpha F - (1 - \alpha)B||^2}{\sigma_C^2} \tag{3.2}$$

where σ_C^2 is the *measurement error variance* and is a user-specified parameter. The $L(F)$ and $L(B)$ terms are computed via color sampling, which we briefly describe as follows:

Pixels from a circular spatial region around pixel I are collected to form the foreground

and background color distributions. The collected pixels include both user-marked pixels from the trimap, as well as *pixels for which α was previously estimated.* Each pixel from this sample is then given a weight; the weight of the ith sample is given as $w_i = S(d)K(\alpha)$: $S(d) = \frac{1}{\sqrt{2\pi}\sigma_d} \exp(-d^2/2\sigma_d^2)$ is a Gaussian spatial distance falloff with d the distance of the sample pixel from pixel I and σ_d a user-specified falloff parameter, and $K(\alpha) = \alpha^2$ for the foreground distribution and $K(\alpha) = (1-\alpha)^2$ for the background distribution measures the relevance of this pixel to the given foreground or background distribution. Thus the $K(\alpha)$ term ensures that foreground pixels contribute no weight to the background distributions and vice versa.

To ensure that there exists a good number of sample pixels, the Bayesian Matting algorithm estimates pixels in *onion-peel* order, marching inward from the boundaries of the user-specified foreground and background regions. This ensures that even when there are few pixels from the trimap in the sample, there will exist some previously computed pixels to sample from.

The foreground and background pixels may come from a wide variety of color distributions. Hence, these foreground and background pixels are *clustered* into a fixed number of clusters, partitioning them into groups. The clustering technique used is by Orchard and Bouman [22] and is briefly described in section 3.1.1. For each *pair* of foreground and background clusters, the likelihoods $L(F)$ and $L(B)$ are modelled for the foreground and background clusters respectively and the MAP estimate given by (3.1) is calculated for that foreground/background pair. From all these pairs, the final F, B and α chosen will correspond to the F, B and α from the pair that has the highest likelihood as defined by (3.1). Hence, for ease of exposition we will simply describe how $L(F)$ and $L(B)$ are modelled for a single foreground or background distribution.

For a given set of pixels in the foreground distribution, each with an RGB value (in 3-dimensional space) F_i and foreground weight w_i, the foreground likelihood is simply

modelled by a multivariate Gaussian with mean and covariance matrix given by the weighted means and covariance matrices of the pixels:

$$\begin{aligned} L(F) &= -(F-\mu_F)^T\Sigma_F^{-1}(F-\mu_F) \\ \mu_F &= \frac{1}{\sum_i w_i}\sum_i w_i F_i \\ \Sigma_F &= \frac{1}{\sum_i w_i}\sum_{i=1} w_i(F_i-\mu_F)(F_i-\mu_F)^T \end{aligned} \tag{3.3}$$

Similarly, for a given set of pixels in the background distribution with background values B_i and background weights w_i, the background likelihood is given by

$$\begin{aligned} L(B) &= -(B-\mu_B)^T\Sigma_B^{-1}(B-\mu_B) \\ \mu_B &= \frac{1}{\sum_i w_i}\sum_i w_i B_i \\ \Sigma_B &= \frac{1}{\sum_i w_i}\sum_{i=1} w_i(B_i-\mu_B)(B_i-\mu_B)^T \end{aligned} \tag{3.4}$$

Combining equations (3.1), (3.2), (3.3) and (3.4) completes the specification of the Bayesian Matting estimation problem. It is possible to take the resulting first order condition and solve for F, B and α, although because of the multiplication between unknowns in (3.2), the resulting problem becomes fairly difficult to solve. The Bayesian Matting solution uses an iterative numerical optimization approach as follows:

First assume α is constant. The resulting problem (3.1) becomes quadratic in both F and B (because there is no longer the multiplication of unknowns in (3.2)) and thus there is an easy closed form linear equation solution. Next, using the values of F and B calculated, fix F and B as constant and solve (3.1) for α: In this scenario, the $L(F)$ and $L(B)$ terms are inconsequential and only the $L(I|F,B,\alpha)$ term in (3.2) is used; the solution for α is obtained by projecting I onto the $F-B$ line in RGB color space, and taking the distance of the projection from B normalized by the length of the $F-B$ line. This procedure is repeated: Fix α constant and solve for F and B, then use the solutions for F and B to solve for α, and use this α to solve for F and

B again, etc. in an iterative procedure.

This concludes our description of the Bayesian Matting Algorithm.

3.1.1 Clustering pixels

An important step in the Bayesian Matting algorithm concerns the partitioning of foreground or background pixels, each with a given weight, into clusters. The algorithm used for this is by Orchard and Bouman [22], and we will briefly describe it here without going into too many specifics.

We first start by describing how to split a single cluster into two. The algorithm first finds the direction of greatest variance among all the points, and splits the points into two clusters separated by the hyperplane perpendicular to this direction passing through the mean. An example is shown in figure 3-1.

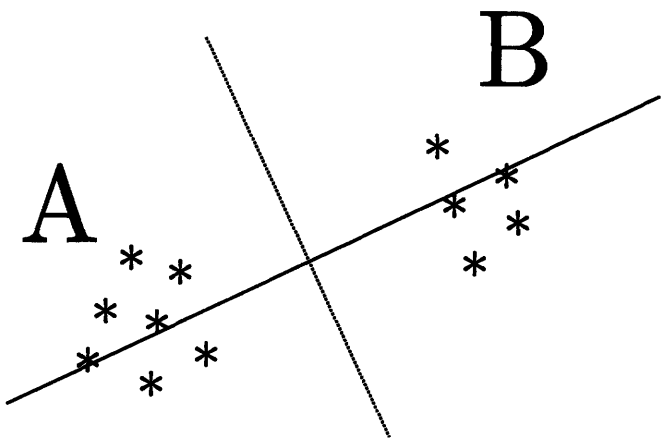


Figure 3-1: One cluster of points is split into two. First the direction of largest variance (solid line) is calculated, and then the points are divided into two clusters, separated by the hyperplane perpendicular to the direction of largest variance passing through the mean (dotted line). In this example, the points are divided into two clusters A (on the left) and B (on the right).

To find the direction of greatest variance, first the weighted covariance matrix Σ is calculated using a formula similar to equation (3.3) (see Appendix B.1.2). From this, the direction of greatest variance is given by the eigenvector corresponding to the

largest eigenvalue of Σ and the variance in this direction is given by the corresponding eigenvalue (see appendix C for a brief discussion of the derivation of this fact).

To split one cluster into multiple clusters, we perform an iterative greedy procedure. First we split one cluster into two smaller clusters. We then choose the cluster with the larger variance along its direction of greatest variance – given by the largest eigenvalue of its covariance matrix – and split that into two. This process is repeated until we have obtained the desired number of clusters: From all the clusters, pick the cluster that has the largest eigenvalue of its covariance matrix, and split that cluster into two.

One important reason for choosing this algorithm is that it is fast – there exists a method to quickly update the covariance matrices of the split clusters in constant time without using the formula for the covariance matrix, which requires going through all the points. Details can be found in [22].

3.1.2 An Interpretation of Bayesian Matting

The estimation framework given by equation (3.1) may be somewhat abstract; in this subsection we will quickly outline an intuitive geometric interpretation that will be helpful in our later discussion.

Consider figure 3-2. The lightly-shaded oval represents the foreground distribution and the darker-shaded oval represents the background distribution as described in (3.3) and (3.4) respectively. The small circle around the pixel I represents the measurement error as described in (3.2). The calculated α represents the relative distance of the projection of I on the $F - B$ line from the estimated B.

Using figure 3-2 as a guide, we may interpret the estimation problem (3.1) as follows: The Bayesian Matting Algorithm attempts to find F and B such that the sum of the following three distances is minimized:

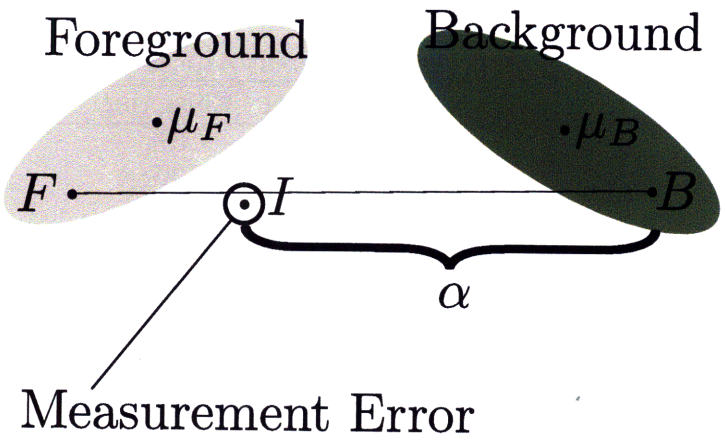


Figure 3-2: An geometric interpretation of Bayesian Matting.

1. Squared distance from F to μ_F, normalized by Σ_F.

2. Squared distance from B to μ_B, normalized by Σ_B.

3. Squared distance from I to its projection on the $F - B$ line, normalized by σ_c^2.

From the estimated F and B, the estimated α is obtained through a projection of I onto the $F - B$ line.

3.2 Problems with Bayesian Matting

In this section, we will study some of the problems associated with the Bayesian Matting Algorithm described so far, both in terms of running efficiency as well as accuracy.

3.2.1 Efficiency

With any computer algorithm, efficiency is important. One way to improve efficiency is to identify runtime bottlenecks in implementations and optimize them. Table 3.1 contains approximate percentage times spent in four different areas in our Bayesian Matting implementation, where we have fixed the number of foreground and background clusters at 3 each.

Table 3.1: Relative Running Times of Different Aspects of Bayesian Matting

Number of Iterations	Miscellaneous	Sampling	Clustering	Numerical Optimization
20	5%	12%	3%	80%
30	3%	9%	2%	87%
40	2%	7%	2%	89%
50	2%	5%	2%	91%
60	2%	5%	1%	92%

In the table, the leftmost column denotes the number of iterations we perform in the numerical optimization step to solve for F, B and α. The miscellaneous column denotes the relative time spent ordering pixels in onion-peel order and pre-computing means and covariance matrices for the different clusters. The sampling time denotes the relative time spent sampling nearby foreground and background pixels, and the clustering time is the relative time spent to order these samples into clusters. Finally, the last column denotes the relative amount of time spent in numerical optimization.

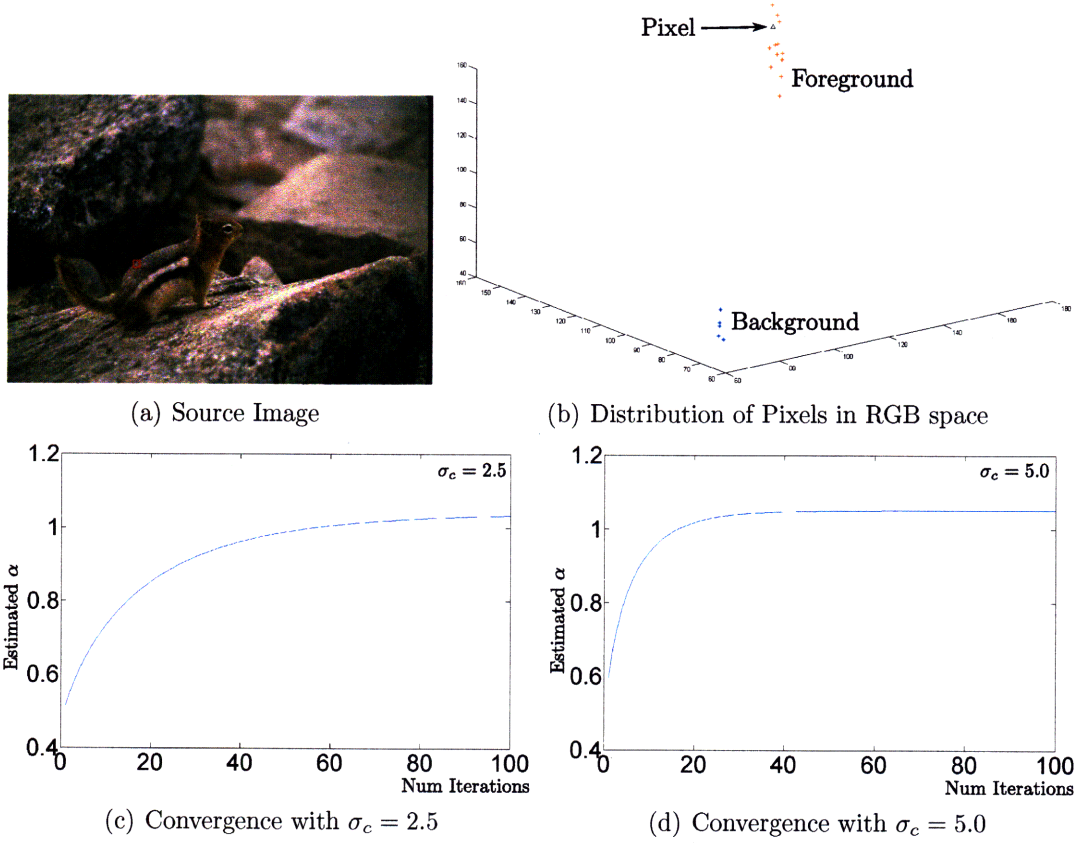
We thus see that the numerical optimization step dominates the run-time. Next, we ask how badly the run-time is affected by increasing the number of iterations. Table 3.2 demonstrates the relative running time after increasing the number of iterations in numerical optimization.

Table 3.2: Relative Total Running Time of Bayesian Matting

Number of Iterations	Total Running Time (Relative)
20	100%
30	136%
40	174%
50	212%
60	250%

From table 3.2, we see that increasing the number of iterations will significantly increase the running time. The final and most important question to ask, then, is: How many iterations are required? Figure 3-3 shows a representative example. From figure 3-3(b), we see that the pixel belongs to the foreground, and thus should have

an α close to 1 – if we follow the interpretation of figure 3-2, the α value should be slightly greater than 1 as the pixel is away from the foreground mean. This is indeed the solution in figures 3-3(c) and 3-3(d), which shows the numerical optimization procedure for two different values of σ_c[1], the measurement error standard deviation.



(a) Source Image

(b) Distribution of Pixels in RGB space

(c) Convergence with $\sigma_c = 2.5$

(d) Convergence with $\sigma_c = 5.0$

Figure 3-3: A representative example demonstrating the convergence rate of the Numerical Optimization technique. Figure 3-3(a) shows the source image, taken from the Berkeley Segmentation Dataset [19]; the small red box shows the area we are sampling from. Figure 3-3(b) shows the foreground and background distribution as well as the pixel being examined. Figure 3-3(c) and 3-3(d) show the convergence of the numerical optimization procedure for two different values of σ_c.

We see that the greater σ_c is, the faster the rate of convergence. We can think of each iteration of the numerical optimization procedure as "moving" F and B such that $\alpha F + (1 - \alpha)B$ is sufficiently close to I, as allowed by the measurement error.

[1]In understanding the magnitude of σ_c, recall that each color channel takes an integer value in the range $[0, 255]$.

Intuitively, the greater σ_c is, the greater our allowance for measurement error, and thus the faster the convergence of F, B and α to an acceptable solution. If we refer to figure 3-2, this intuition translates as follows – the greater σ_c is, the larger the circle representing the measurement error, and thus the less F and B have to move away from their means in order for the line between F and B to pass through the circle.

Figure 3-3 demonstrates that the rate of convergence is relatively slow, and therefore about 40 iterations is required in order to achieve sufficient convergence. From table 3.2, this translates to about a 75% slowdown over an approach with 20 iterations. In fact, even with 20 iterations, Bayesian Matting is already fairly slow, and thus with 40 iterations Bayesian Matting is rather inefficient. It therefore follows that the inefficiency of Bayesian Matting represents a problem with it.

3.2.2 Accuracy I – Non-Sparse Alpha due to Mean Preference

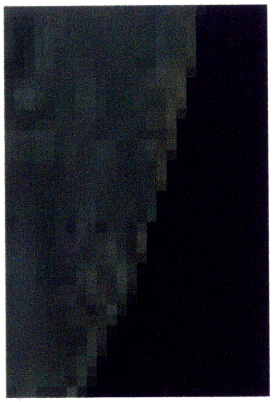
In this section, we discuss what we believe is the most important accuracy problem with Bayesian Matting.

We believe that one important feature of alpha mattes is that *it should be sharp when the foreground/background boundary is sharp*, which implies that in images with sharp boundaries α should primarily take the values of 0 and 1. This desirable property of α is known as *sparseness.*

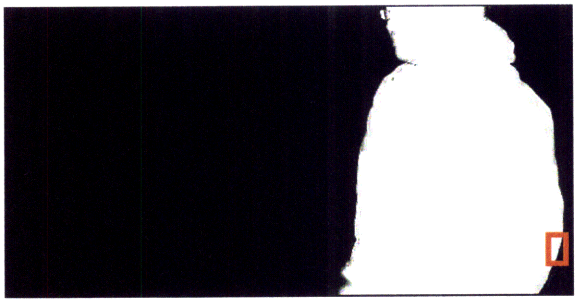
Bayesian Matting fails this criteria fairly often. An example can be found in figure 3-4, which demonstrates a matte created by Bayesian Matting which is not sharp at the boundary, even though the foreground-background boundary in the actual image is relatively sharp.

Why is this so? We believe the main reason is the propensity of Bayesian Matting to

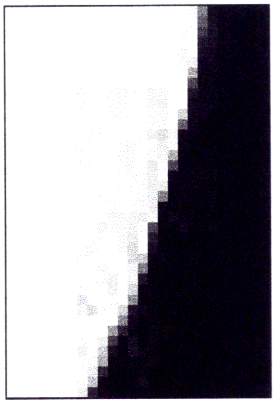
(a) Source Image



(b) Closer look at marked region



(c) Alpha Matte generated with Bayesian Matting



(d) Closer look at marked region

Figure 3-4: An example demonstrating how Bayesian Matting does not give sharp mattes. Figures 3-4(b) and 3-4(d) show close-up views of the regions marked in red in figures 3-4(a) and 3-4(c) respectively. We see that even though the foreground-background boundary is relatively sharp, the matte generated is not.

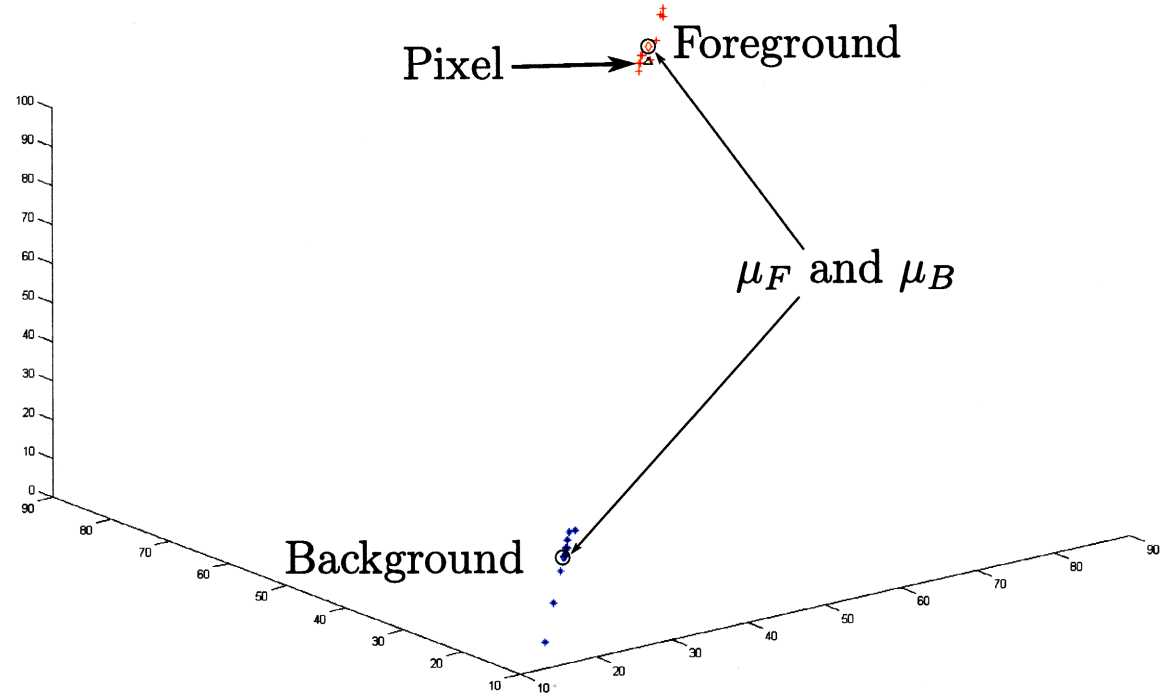
prefer means: This means that Bayesian Matting prefers F to be near μ_F and B to be near μ_B, as well as the measurement error to be near 0. The penalty for deviating is large; for example, from our discussion in section 3.1.2, we know that the penalty scales with the *square* of the distance from F to μ_F. This causes problems in two ways:

1. It assigns fractional values of α to pixels that clearly belong to the foreground or background clusters, as long as these pixels deviate sufficiently far from the relevant foreground or background mean.

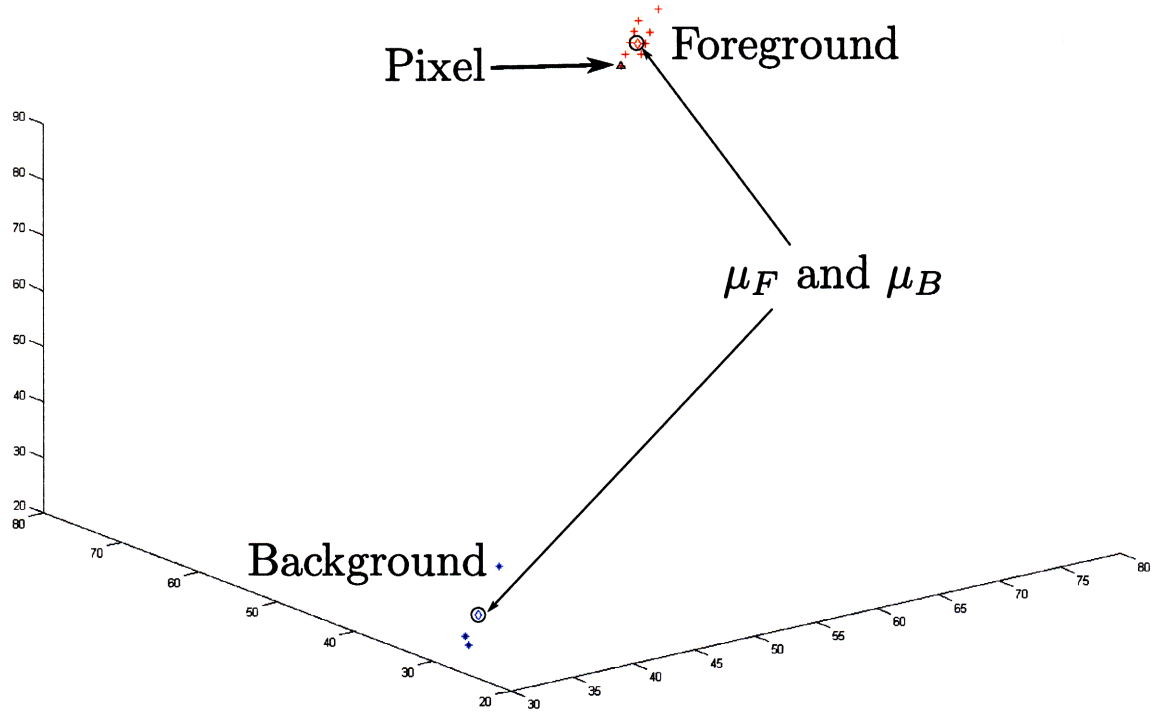
2. When deciding which foreground/background pair of clusters to pick, it can often pick the wrong one.

We will briefly study each of these problems in turn. First, we consider the assignment of fractional values of α. Figure 3-5 shows, for two pixels from the problem region labelled in figure 3-4, a representative foreground/background cluster pair for each pixel. Even though both pixels clearly lie in the foreground distribution and should be given $\alpha = 1.0$, they are given fractional values of α, because they lie away from μ_F.

It may be argued that fractional values of α that are close to 0 and 1 – such as those in figure 3-5 – are very similar to $\alpha = 0$ or $\alpha = 1$; however, this problem is exacerbated if the foreground or background distribution has high variance and the pixels lie on the edge of the distribution. In this case, the calculated values of α are no longer near 0 or 1.



(a) Within the foreground cluster, but assigned $\alpha = 0.97$.

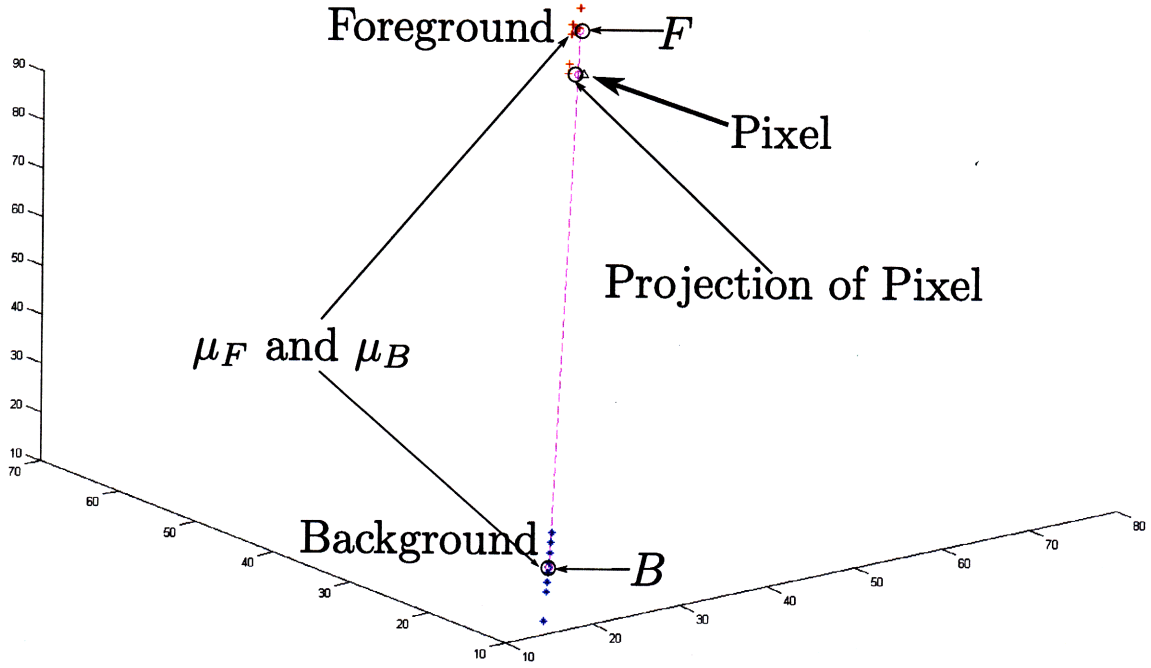


(b) Within the foreground cluster, but assigned $\alpha = 0.96$.

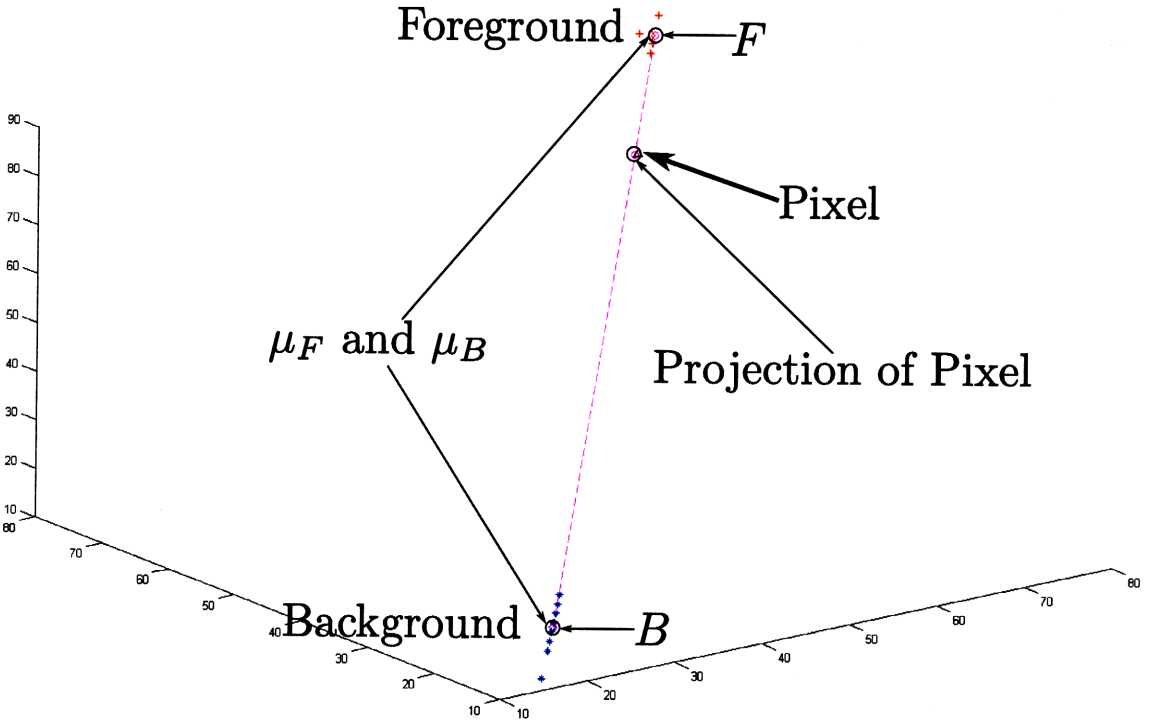
Figure 3-5: The above two plots show Bayesian Matting assigning a fractional value of α to two pixels that are both in the foreground distribution and should be assigned $\alpha = 1$. The reason is that the pixel (marked with a triangle) lies away from μ_F (marked with a diamond and in a circle).

The second problem created by mean preference is that it can cause Bayesian Matting to pick the wrong foreground/background pair of clusters. Recall that Bayesian Matting tries to solve for F, B and α for every pair of clusters and then picks the pair that minimizes the sum of normalized squared distances from their mean, as described in section 3.1.2. This means that a pair of clusters that results in a sparser and more correct value of α may be overlooked for an inferior pair of clusters.

Figure 3-6 shows an example for one pixel in the problem region labelled in figure 3-4, and two different pairs of foreground/background clusters. The pair of clusters in figure 3-6(a) results in a more accurate value of α – though not entirely accurate (the correct $\alpha = 1$) because of the first problem caused by mean preference – but the pair of clusters in figure 3-6(b) is chosen instead because it has F much closer to μ_F and the measurement error – the distance from a pixel to its projection on the $F - B$ line – is smaller, *even though these distances are already small in figure 3-6(a), because of the squared distance penalty*! As a result, the value of α chosen is more incorrect.



(a) Better Pair of Clusters, inferred $\alpha = 0.92$.



(b) Inferior Pair of Clusters, inferred $\alpha = 0.79$. This is the pair chosen by Bayesian Matting.

Figure 3-6: Figures 3-6(a) and 3-6(b) show two foreground/background pairs of clusters for the same pixel. The pixel is marked with a triangle, μ_F and μ_B are marked with diamonds, and the chosen F and B are marked with circles, as well as the projection of α onto the $F - B$ line, which is shown dotted. The pair of clusters in figure 3-6(b) is chosen because of the smaller $F - \mu_F$ and projection distances.

Failure to Recognize Shading

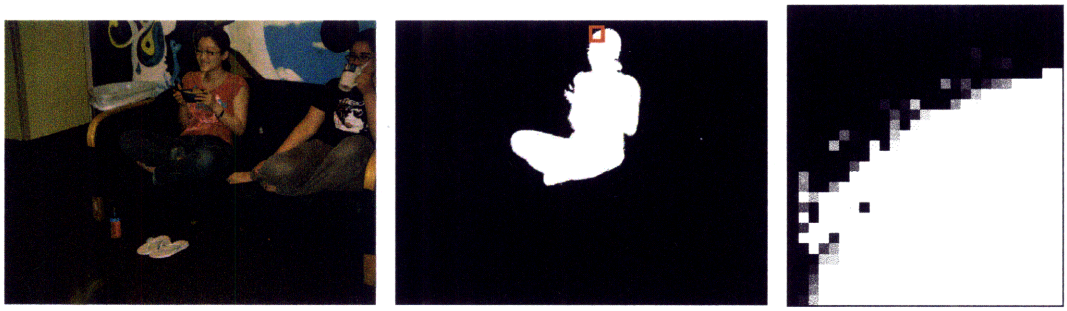
A more important reason why mean preference is a problem is that it *ignores shading effects.* The foreground-background boundary often contains shading of the foreground or background; this means that the colors on the boundaries are of slightly different shades – even though they may be of the same color – as the interior regions, which are used to build the foreground and background distributions. Even with shading, the foreground/background boundary is often still very distinct and thus α should be sparse; however, by using mean preference, Bayesian Matting ignores the effect of shading and automatically assumes that the shades on the extreme regions of the foreground or background distributions should have fractional alphas.

3.2.3 Accuracy II – Discontinuity due to similar foreground and background distributions

Another property of α is that if the foreground is continuous, the region where α is high should be continuous. Bayesian Matting does not explicitly factor in this constraint as each pixel is estimated independently; however, in most cases, this should not cause discontinuity as long as the local foreground and background color distributions are sufficiently distinct.

However, in regions where the local foreground and background color distributions are very similar, or even overlap, this can cause large fluctuations in the estimated α in the region and result in discontinuous regions of α. An example can be shown in figure 3-7 in the middle-upper area of figure 3-7(c). This is because in a small spatial region, the image pixels move a small amount in RGB space, but a small movement can result in very different values of α being estimated when the foreground and background color distributions are very close to each other or overlap.

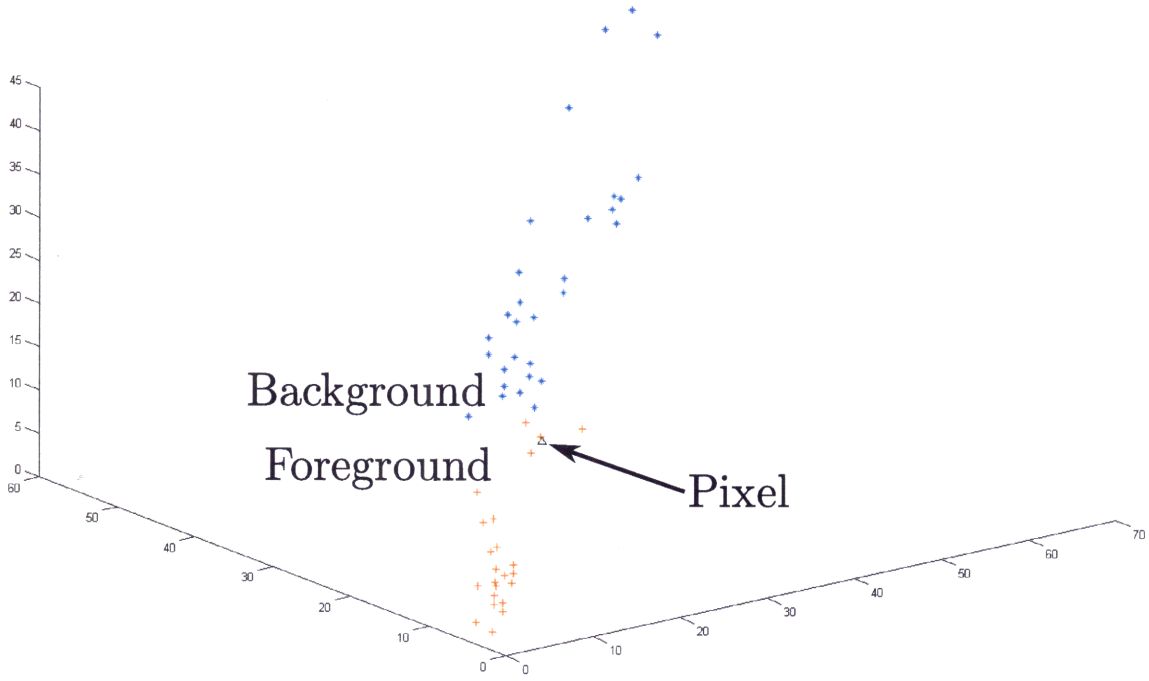
Remark 3.1. As noted in section 2.8, areas where the foreground and background have similar color distributions are a problem faced by all matting algorithms.



(a) Source Image

(b) Alpha Matte generated with Bayesian Matting

(c) Closer look at Marked Region



(d) A plot of the foreground and background pixels around one problem pixel.

Figure 3-7: Discontinuity caused by similar foreground and background distributions: Figure 3-7(c) shows a close up of the region marked in red in figure 3-7(b); there are noticeable discontinuities at the middle-upper area of the image. Figure 3-7(d) shows the foreground and background distributions around one of the problem pixels; the foreground and background distributions are very close to each other and the pixel is on the boundary of the foreground/background distribution.

3.2.4 Accuracy III – Miscellaneous Issues that Affect Accuracy

This section discusses some relatively minor issues that can affect the accuracy of Bayesian Matting in a detrimental way. Unlike the problems of the previous two sections, most of these issues can be rectified within Bayesian Matting easily – see section 3.3 for our modifications.

Number of Clusters

Bayesian Matting partitions the foreground and background distributions into a fixed number of clusters. However, setting a fixed arbitrary number of clusters is not ideal: The number of clusters may depend on the variation of foreground and background color in an image, and it should also be different in different areas of the image.

The goal of having "clusters" is to ensure that each foreground or background group of similar colors gets its own cluster. Setting a fixed number of clusters create two opposite problems:

1. The fixed number of clusters may be too small: In this case, there may be clusters that span multiple different colors.

2. The fixed number of clusters may be too large: In this case, there exist different clusters that contain very similar colors and it would be more optimal to merge these clusters.

In general, these problems – especially the first problem – will cause errors in the estimated μ and Σ of each cluster, and thus result in incorrect estimation.

Remark 3.2. Because we have to perform numerical optimization for every pair of clusters, and numerical optimization is the runtime bottleneck of Bayesian Matting (see section 3.2.1), having too many clusters will slow down the runtime greatly.

Choosing Pixels

Recall that Bayesian Matting also includes the previously inferred foreground and background pixels in the set of sampled foreground or background pixels. This can cause problems, especially when combined with the mean preference tendencies of Bayesian Matting, as described in section 3.2.2.

The first problem is that the sampled pixels may have been estimated incorrectly: We have seen in section 3.2.2 that mean preference can result in the wrong set of foreground and background clusters being chosen, and this will result not only in inaccurate α values, but also in incorrect values of F and B being estimated (see figure 3-6 for an example). Thus, the sampled values of F will be incorrect.

The second problem is that because of mean preference, the sampled foreground or background pixels that were previously estimated will be close to the mean of the distribution they came from, and from mean preference the estimated foreground or background value of the *current* pixel will also be close to this mean, and thus subsequent future pixels, which sample from this pixel, will be biased towards this mean.

The effect of the problems associated with choosing previously estimated pixels can result in incorrectly estimated μ and Σ.

Weighting Pixels

Recall that Bayesian Matting weights each pixel by a multiplicative combination of a spatial distance term and a relevance term. We will concentrate on the spatial distance term, which is a Gaussian falloff. We have found in practice that this falloff – which is exponential – decays too rapidly. This results in two problems:

1. Pixels that are slightly far away from the pixel to be estimated are essentially worthless. When combined with sampling from previously inferred foreground and background pixels that can be inaccurate (see previous section), this means

that greater and greater weight is placed on possibly incorrect foreground and background pixels.

2. In a practical sense, there are many numerical difficulties associated with performing clustering and inference on pixels that have such a great variety of weights.

A more troubling aspect with pixel weighting is that Bayesian Matting *fails to normalize weights* before calculating covariance matrices. Since nearby pixels are given higher weights, the covariances of clusters that were sampled near the estimated pixel is much greater than the covariances of clusters that were sampled further away from the estimated pixel. This can cause problems in at least two ways:

1. The higher the covariance, the more we allow deviation away from the mean. It should be the case that for clusters that are further away spatially, we should allow greater deviation away from the mean – especially when considering shading. However, the approach used by Bayesian Matting does the *opposite*!

2. The (relatively) unequal covariances can cause problems when choosing the correct pairs of clusters to use for estimation, similar to the problems caused by mean preference (see figure 3-6 for an example). For example, consider an image where the trimap specified is tighter on the background side (meaning that most of the unknown pixels are foreground). For the pixels on the background boundary, the sampled background distribution has a much higher covariance than the sampled foreground distribution. This means that the distance between F and μ_F is much more important than the distance between B and μ_B. In this case, for a *background* pixel, the correct pair of clusters determined by Bayesian matting is primarily determined by the *foreground* estimates! This problem is not rectified by onion-peel ordering of pixels to estimate, because this pixel – a background pixel close to the edge of the trimap – will be sampled early in the order.

3.2.5 A Remark on Measurement Error

From section 3.2.1, we have seen that the measurement error can affect the rate of convergence of the numerical optimization procedure and hence the efficiency of Bayesian Matting. We have also seen in section 3.2.2 that it can affect the choice of the foreground and background cluster pair to use for estimation, as it normalizes the distance penalty of the image pixel to its projection on the estimated $F - B$ line; because the measurement error is generally small considered to the foreground and background covariances, it could play a very important role in choosing the correct pair of clusters.

Why should measurement error play such a large role? In other words, why should the projection distance be normalized this way? We feel that it should make no difference whether the projection distance is 5 or 6 units in RGB space; however, if the measurement variance is sufficiently small, such a difference can be huge. It can be argued that measurement error is really an artificial construct in order to make the estimation problem more numerically tractable.

A more reasonable model would have the measurement error penalty be zero for small values up to a threshold and then increase exponentially above that threshold. However, this model may be a lot more difficult to estimate and solve in a practical sense.

3.3 A Slightly Modified Bayesian Matting Algorithm to Form a Baseline

For our experiments and comparisons in chapter 6, we modify the Bayesian Matting algorithm to fix some of the issues discussed in section 3.2.4, as follows:

1. We only sample from pixels that are marked either as foreground or background in the trimap; to compensate for the lack of sampling from previously estimated

pixels, we use tighter trimaps.

2. We weight sample pixels using a $1/d$ falloff, where d is the spatial distance from the pixel to estimate, instead of a Gaussian falloff.

3. We normalize weights – so that the total weight of all pixels is 1 – before calculating covariance matrices.

We fix the number of clusters at 3. While this does not address the issue of a fixed number of clusters – and we certainly could have used the improved clustering algorithm discussed in 4.4 – we have found that this proves adequate for comparison purposes.

Finally, as discussed in section 3.2.1 on convergence, we fix the number of iterations of numerical optimization at 40 to allow for sufficient convergence.

Chapter 4

Segment-Based Matting: A Closed Form Color-Based Statistical Matting Algorithm

In this chapter, we present Segment-Based Matting – a closed form trimap and color-based statistical matting algorithm that addresses the problems of Bayesian Matting we discussed in section 3.2.

Section 4.1 presents the motivation behind Segment-Based Matting and presents a quick overview and summary. Section 4.2 presents an overview of the simple color line model that we use to model the foreground and background distributions, section 4.3 provides a comprehensive description of our Segment-Based Matting algorithm, and finally section 4.4 describes an improvement and adaptation of the Orchard-Bouman clustering algorithm (see [22] and also section 3.1.1) that we use to partition sampled pixels into clusters.

4.1 Motivation and Overview

We have seen in the previous chapter that Bayesian Matting has problems with efficiency and accuracy – in particular with regard to issues of shading and non-sparse

alphas, resulting in mattes that are far less sharp than optimal. In this chapter, we propose a new color-based statistical matting algorithm – Segment-Based Matting – that fixes these problems. In this section, we provide a quick overview of how Segment-Based Matting addresses accuracy and efficiency, and finally provide a brief summary of the approach.

Segment-Based Matting distinguishes itself from other approaches in a few ways. In contrast to Bayesian Matting – which also uses color-based statistics – it models foreground and background distributions as lines in three-dimensional RGB space. This provides a model for shading and also allows a closed-form solution. In contrast to approaches such as Levin et al's closed form laplacian matting which explicitly enforce smoothness of the alpha matte, Segment-Based Matting is purely color-based and does not enforce smoothness directly.

4.1.1 Accuracy

In section 3.2.2, we found that a major cause of Bayesian Matting's inaccuracy was that it did not generate sharp alpha mattes in many cases – especially in regions with shading – because of its bias towards means. Segment-Based Matting models distributions as lines or line segments in RGB space, and shading can be incorporated easily by allowing foreground and background pixels to lie along these lines. We eliminate mean bias by modelling the probability of foreground and background colors with a flat segment distribution. An empirical justification for modelling the distributions as lines can be found in a simple *color line model* which we will briefly discuss in section 4.2. There are at least two ways to incorporate shading using lines in a reasonable way, which we will also discuss in section 4.2.

Another way to improve the sharpness of the alpha matte is to ensure that for most pixels, $\alpha = 0$ or $\alpha = 1$. This is justified because most pixels are obviously – to the human eye – entirely foreground ($\alpha = 1$) or background ($\alpha = 0$). Following this intuition, we use a hypothesis testing approach (see Appendix B.3 for an overview)

to test if pixels belong to the foreground or background distributions.

Remark 4.1. We do not directly address the issue of the continuity of the alpha matte – such as discontinuities caused when the foreground and background distributions are similar – that was discussed in section 3.2.3. However, we have found empirically – see chapter 6 for examples – that Segment-Based Matting alleviates many of the problems associated with discontinuity of α. Chapter 5 also discusses a new approach for using inpainting and texture synthesis to resolve ambiguities in matting – easily applicable to Segment-Based Matting – that explicitly addresses the problem of similar foreground and background distributions.

4.1.2 Efficiency

In section 3.2.1, we saw that the major efficiency cost of Bayesian Matting was the number of iterations required for convergence during numerical optimization. The reason numerical optimization was required was that there was no closed form solution.

In Bayesian Matting, there is no closed form solution because we have to optimize over three-dimensional Gaussians. However, if the foreground and background distributions were *one-dimensional* lines in RGB space – as in Segment-Based Matting – a closed form solution might exist. In our Segment-Based Matting approach to model shading, a closed form solution does exist and improves the running time significantly. For the Bayesian case, there is an obvious solution if the foreground and background color distributions lie on the same line, since the problem becomes one-dimensional. We have not closely studied solving the Bayesian Matting MAP estimator in the general case using the line distribution model – as we feel our Segment-Based Matting approach is superior – but it is quite possible that it has a closed form solution.

Our approach to test if a given pixel was obviously foreground or background also

improves efficiency. If we could rapidly classify these obvious cases based on color distributions, we would not have to solve for these pixels. If the number of obvious cases are large, this would reduce the running time significantly.

4.1.3 Summary of Approach

We will briefly provide a summary of Segment-Based Matting; section 4.3 will reiterate this summary, provide more details, and complete the description of the algorithm. Segment-Based Matting estimates each pixel independently. For each pixel, it performs the following steps:

1. Sample the local foreground and background pixels, as labelled in the trimap. Partition the foreground and background pixels into multiple clusters.

2. For every cluster, we fit a one-dimensional line or line segment in RGB space representing its distribution.

3. We test if the pixel belongs to exactly one of the foreground or the background. If it does, we assign $\alpha = 1$ or $\alpha = 0$ respectively and we skip the next step, otherwise we proceed with the next step.

4. For each pair of clusters, estimate α. This gives multiple solutions of α, one from each pair of clusters. We choose the solution of α that is best according to a scoring system that is described in section 4.3.8. This is the key step in the algorithm.

5. Once we find α, we solve for F and B.

4.2 The Color Line Model

In the color line model, we assume that after clustering, the foreground and background clusters form straight lines in RGB space. This model arises from the empirical observation that in the local area around a pixel where we collect samples, the pixels

tend to form straight lines. An example can be found in figure 4-1.

The color line model has some empirical justification. Omer and Werman [21] have noted that pixel histograms of an *entire image* tend to form elongated lines in RGB space, and have used this to create a different image-specific color space based on color lines. Levin et al [16] have also noted that in situations such as shading and edges between background and foreground regions, the color line model captures much of the local pixel information. The color line model is also used in some image matting algorithms – it is the core assumption behind the closed form laplacian matting [16] and spectral matting [17] approaches.

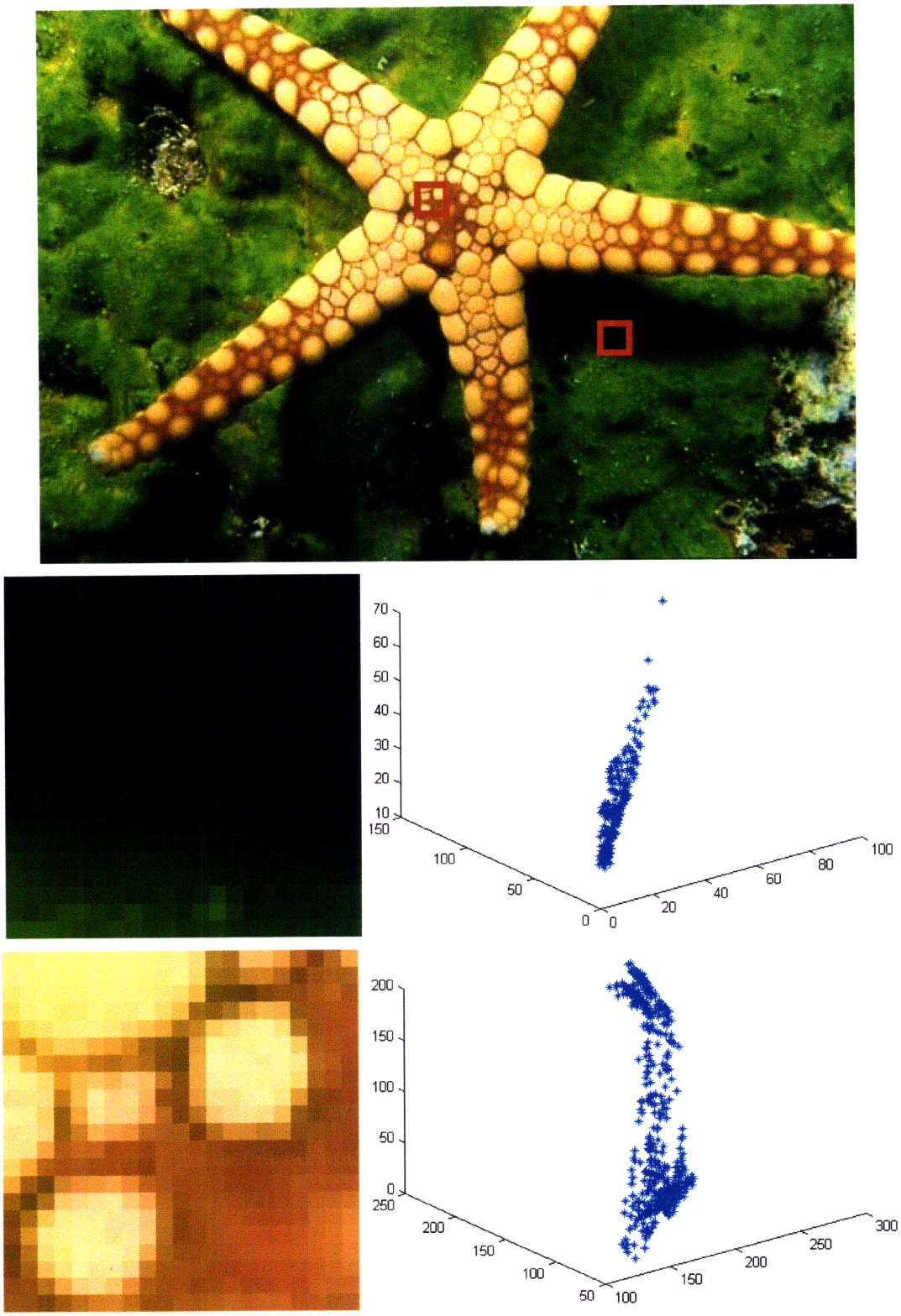


Figure 4-1: The Color Line Model as demonstrated from the starfish image from the Berkeley Segmentation Dataset [19]. We have provided a close-up view as well as plots of the colors in RGB space of the areas marked in red in the original image.

4.2.1 Applying the Color Line Model in Our Framework: The Full Line Model vs. The Line Segment Model

In our matting framework, we use the color line model to represent shading: Each foreground (background) distribution can be represented by a line in RGB space, and this line represents the possible shades that a typical foreground (background) pixel from this distribution can take. Given a set of foreground (background) pixel samples that lie on a line, there are at least two different methods for incorporating this interpretation:

- **The Full Line Model.** We solve for the foreground (background) pixel by allowing them to lie anywhere on the line. This approach fully extrapolates the shading for the foreground (background).

- **The Line Segment Model.** From the points on the line, we generate a line segment on this line: The line segment may be generated in many ways, such as taking the two extreme points, or taking the mean of the samples plus or minus a few sample standard deviations; in our approach, we use the two extreme points of the line. We then solve for the foreground (background) pixel by ensuring that it lies on on the foreground (background) line segment; if this is impossible, we attempt to extend the line segment, but by as little as possible. While this approach still models shading along the line segment, it restricts the amount of shading desired.

Figure 4-2 demonstrates the difference between the two models. We note that in both models, we still evaluate the quality of the estimation by deviation from the mean, with a different penalty system to prevent the bias towards the mean associated with Bayesian Matting (see section 4.3.8 for details); the difference in the models comes solely from the way the foreground (background) pixels – and hence α – is solved for. Details of the solution techniques can be found in sections 4.3.3 through 4.3.7.

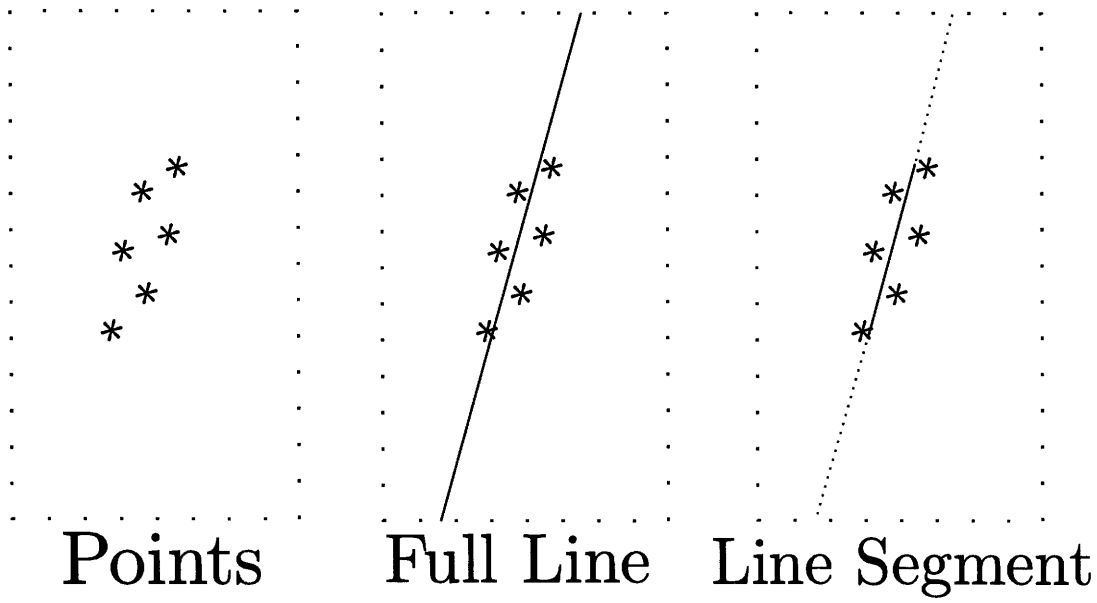


Figure 4-2: A comparison of the full line and line segment models. The left-most image shows the original points, the middle image shows fitting a full line to the points, while the right-most image shows fitting a line segment to those points. In the right-most image, the dotted lines signify possible extensions to the line segment – but only if really necessary – in order to obtain a solution to the matting problem.

The full line model is more appropriate when the range of shading is large and the trimap is not very tight – so when evaluating pixels far from the borders of the trimap, we can fully extrapolate the amount of shading required. The line segment model is more appropriate when the trimap is tighter and we wish to obtain a more precise solution, and in some sense it is more principled because it does not extend the shading indefinitely, but rather only within reasonable limits.

4.3 Our Approach

Segment-Based Matting is a color-based approach that like Bayesian Matting, samples the local foreground and background pixels, partitions them into foreground and background clusters, and solves for each unknown pixel independently. The key difference is in how F, B and α are solved in the matting problem. The steps in the Segment-Based Matting Algorithm for each unknown pixel are as follows:

1. Sample the local foreground and background pixels and assign each sample a weight.

2. Partition the samples into multiple foreground and background clusters; we use a modified Orchard-Bouman clustering method, which will be described at the end of the chapter in section 4.4.

3. For each cluster, fit a line in RGB space, following the color line model. We do this via principal component analysis on the points (see Appendix C) using their weighted covariance matrix and taking the first principal component (in the language of Appendix C, we try to reduce the data to 1 dimension).

4. Determine if the pixel is unambiguously foreground or background. If so, set $\alpha = 1$ or $\alpha = 0$ respectively and skip to step 7; if the pixel is neither foreground nor background or *both* foreground and background, we proceed with the next few steps.

5. For each pair of foreground/background lines, we solve for α. This is the key step in the algorithm.

6. We choose the foreground/background pair that returns the best solution, as determined by a scoring system, and we use its associated solution of α.

7. Once α is solved, we will solve for F and B as follows: If $\alpha = 1$, we set F to be equal to the pixel and B to be zero. Similarly, if $\alpha = 0$, we set F to zero and B to be equal to the pixel. Otherwise we follow an approach similar to [2] and [34]: We sample from the nearby foreground and background pixels (the same set we used to solve for α), and find the pair of foreground and background pixels that minimize $||I - \alpha F - (1 - \alpha)B||$, where I is the pixel value and F and B are the values of the foreground and background pixels. The values of this pair then become the values of F and B for that pixel.

The next few sections will fill in the details of the above algorithm – sections 4.3.1 through 4.3.8 will briefly discuss the key techniques, while section 4.3.9 will add

details about important optimizations and special cases. Let us quickly define some terminology: If $\mathbf{x}$ is a point in RGB space, and $\hat{\mathbf{x}}$ is its projection onto a plane or a line, we define the following:

- The **projection value**, which is defined only if $\hat{\mathbf{x}}$ is a projection on a line, is the distance from $\hat{\mathbf{x}}$ to an arbitrary point on the line; in general, it does not matter which point on the line is the reference point, as long as it is kept constant for all points projected to that line.
- The **projection distance** is the orthogonal distance $||\mathbf{x} - \hat{\mathbf{x}}||$.

For more information on linear projections, refer to Appendix A.2.

4.3.1 Sampling and Weighting pixels

We follow a similar sampling and weighting technique as our modified Bayesian Matting Algorithm described in section 3.3. Samples are collected from a circular region around the pixel to infer from, and only from pixels marked as foreground or background in the trimap – no previously estimated pixels are used. The area of sampling is initially a circle of radius five around the pixel; we expand the radius until at least twenty samples are collected. We only collect foreground samples for the foreground distribution and background samples for the background distribution.

We weight samples using a $1/d$ falloff, where d is the Euclidean distance of the pixel from the sample. After collecting the sample, we normalize the weights such that they sum to 1.

4.3.2 Foreground and Background Detection

For ease of exposition, we first assume that there exists a single cluster of foreground points, with fitted foreground line, and we wish to test if the pixel belongs to the cluster. We project all the foreground points to the line, and replace each point with its projection value; hence the points now become uni-dimensional. We then calculate

the sample mean and variance of the projection values and fit a univariate normal distribution to the sample mean and variance.

The pixel needs to satisfy two tests in order to be considered part of the foreground cluster:

1. It should belong to the foreground line.

2. If it belongs on the foreground line, its projection value should belong to the univariate normal distribution with mean and variance given by the mean and variance of the sample projection values.

For the first test, we use an ad-hoc method: The pixel belongs on the foreground line if its projection distance is at most twice the maximum projection distance of the sample foreground points. A justification for this simple test is that since the projection distance measures the distance orthogonal to the principal component (the line itself), and most of the information – according to the color line model – is contained along the line, the projection distance is simply a noise factor that does not convey much information about whether the point belongs to the distribution. We simply want to determine if the pixel is sufficiently close to the line.

Remark 4.2. It is possible – especially if the number of pixels that make up the cluster is few – that the maximum projection distance of the sample points can be very small and insignificant in RGB space. As an additional heuristic test, we will also allow a pixel to lie on the foreground line if its projection distance is less than a threshold value; the threshold value we use is 5 units in RGB space.

The second test is more important, since it is a test along the principal component, which contains most of the information about the distribution. Thus we use a more principled approach. We use a hypothesis testing framework (see appendix B.3) to

test between the following two hypothesis:

H_0 : The pixel belongs to the foreground cluster.

H_A : The pixel does NOT belong to the foreground cluster.

Similar to many classical statistical tests, we fix the probability α of incorrectly rejecting H_0[1]. We fix $\alpha = 0.05$. Under these assumptions, and fitting a univariate normal distribution to the mean and variance of the sample projection values, the test *rejects* H_0 if the projection value of the pixel is greater than 1.96 standard deviations (where the standard deviation is the square root of the sample variance) away from the mean of the sample projection values; see appendix B.3 for details. Equivalently, we will claim the pixel passes the second test if its projection value is less than 1.96 standard deviations away from the mean of the sample projection values.

Remark 4.3. Again, if the number of pixels is very small, the sample variance can be small; thus, as an ad-hoc rule, we ensure that the sample standard deviation is at least a small value; we use the value of 1.5 units in RGB space.

We can summarize the above discussion – modulo the remarks at the end of each case – as follows: The pixel belongs to the foreground line if its projection distance is less than twice the greatest sample projection distance and its projection value is less than 1.96 sample standard deviations away from the mean of the sample projection values.

The extension to multiple clusters is simple: With multiple foreground clusters, the pixel belongs to the foreground if it passes the above described series of tests for at least one of the clusters. The sequence of tests above also extend to the background; we test for foreground and background independently. Thus, at the end of this series of tests, we can tell if a pixel belongs to the foreground, the background, both, or

[1]This is known as type 1 error. See appendix B.3.

neither.

4.3.3 Solving for α: An Overview

Suppose the pixel belongs neither to the foreground nor the background, or belongs to both the foreground and the background. We then estimate α using pairs of foreground/background lines. Since each of these lines lie in three-dimensional RGB space, there are three possible cases for every pair of lines:

1. They are collinear.
2. They are coplanar. This is the case, for example, if the lines intersect, or have the same slope.
3. They are neither collinear nor coplanar.

The following sections discuss how we detect the cases and solve for each possible case. We actually do not solve for the third case, as its solution is very sensitive to noise – hence the description in section 4.3.7 is primarily for reference – and therefore we will treat the lines as coplanar if we detect the third case; details can be found in section 4.3.4.

Remark 4.4. The case we solve for does not rely on the location of the input pixel in RGB space, because it is very difficult to incorporate it directly without introducing significant additional complexity. If the case chosen does not fit the input pixel well – for example, if we use the collinear case and the projection distance of the input pixel on the line is large – the scoring system in 4.3.8 will heavily penalize it and thus reduce the possibility of choosing a bad case.

4.3.4 Solving for α: Detecting the Cases and Additional Transformations

It is fairly trivial to detect if lines are collinear or coplanar if we only allow *exact* collinearity and coplanarity. However, there is always some noise associated with

sample observations, and therefore we need to allow inexact collinearity and coplanarity; an example can be found in figure 4-3, where the lines are not exactly collinear, but it is in our interests to detect them as such.

Figure 4-3: An Example of inexact collinearity: The lines are not exactly collinear, but we should probably detect them as such.

We can define a line as two points on the line; the choice of such points will be important because of the way we detect inexact collinearity and coplanarity. Like in the line segment model, we can choose these two points as the extreme sample points on the line – the points with the greatest projection values – or we can simply take the projection mean plus or minus a few standard deviations; in our approach, we choose the two extreme points.

Collinearity Detection

Two lines are exactly collinear if they satisfy two properties:

1. The angle between the two lines is zero or π. Equivalently, the absolute value of the cosine of this angle is one.

2. Each line can be defined with two points, for a total of four points. Pick any of these two points, and project all the points on the line defined by these two points. The projection distance of all points should be zero.

For inexact collinearity, we relax the above two conditions a little:

1. The absolute value of the cosine of the angle between the two points is sufficiently high.

2. Out of the four points, take the two points that are the furthest away from each other, and project all the points on the line defined by these two points. The projection distance of all points should be sufficiently small.

We will consider two lines as being collinear if both tests above pass. For our purposes, we take the cosine angle threshold as $\cos\frac{\pi}{12}$ and the projection distance threshold as 5 units in RGB space.

Remark 4.5. The second condition suffices for exact collinearity, however, if we only use the second condition for inexact collinearity, there can be problems. For example, consider the case in figure 4-4. The two lines are not collinear, but the two points on the dotted line are close to each other and straddle the solid line. The second collinearity test will pass, but the first will fail. This is why we need both conditions to pass.

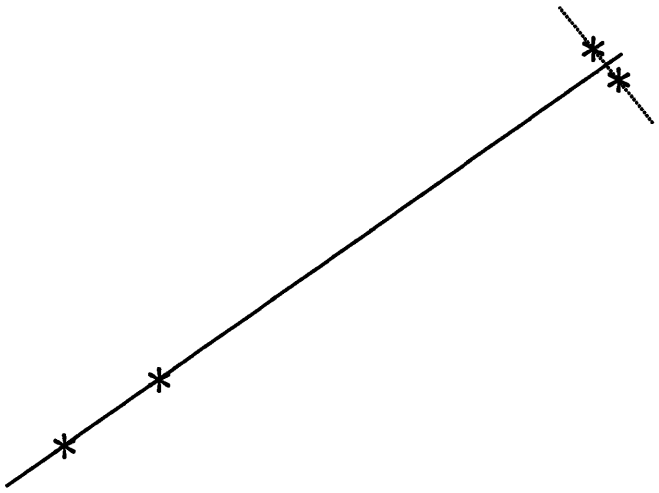


Figure 4-4: Non-collinear lines: The lines are not collinear, but if we use only the second condition for inexact collinearity, we would detect them as collinear.

Coplanarity Detection

In this section, we will assume that the lines are not collinear. With two points for each line, there are a total of four points. We first remove the point that is closest to all the other points. The remaining three points define a plane; if the points selected are collinear, we try a different set of three points, and since the lines are not collinear, there exist a set of three points that define a plane. Associated with this plane is a

vector $\mathbf{v}$ that is perpendicular to this plane.

Now, the fourth point lies in this plane – and therefore the lines are exactly coplanar – if the following two conditions are met:

1. The projection distance of this point on the plane is zero.
2. Take the three lines formed by taking this point with the three points that formed the plane. These lines are all perpendicular to the vector $\mathbf{v}$; equivalently the absolute value of the cosines of their angles with $\mathbf{v}$ are all zero.

As with the collinear case, we relax the above conditions for inexact coplanarity:

1. The projection distance of this point on the plane is sufficiently small.
2. Take the three lines formed by taking this point with the three points that formed the plane. The absolute values of the cosines of the angles of these lines with $\mathbf{v}$ are all sufficiently small.

We will consider two lines as being coplanar if they are not collinear and both tests above pass. We use the same thresholds as in the collinear case: The cosine angle threshold is $\cos\frac{\pi}{12}$ and the projection distance threshold is 5 units in RGB space.

Remark 4.6. The first condition suffices for exact coplanarity; however if we only use the first condition for inexact coplanarity, there can be problems. For example, consider the case in figure 4-5. The lines are not coplanar, but since the points on the dotted line are close to each other, the first test would pass but the second would fail. Thus we need both conditions to pass.

Post-Detection Transformations

Once we have detected if the pair of lines are collinear, coplanar, or otherwise, we transform the points on the line as follows: First, we perform principal component analysis on the set of four points comprising the two lines. Next, we do the following depending on the case we detected:

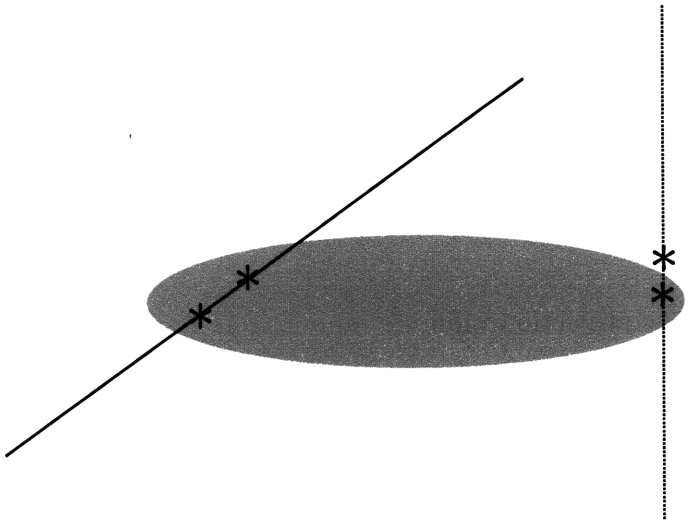


Figure 4-5: Non-coplanar lines: The lines are not coplanar, but if we use only the first condition for inexact coplanarity, we would detect them as coplanar.

- If collinearity was detected, project the four points on the first principal component. Also project the image pixel for which we wish to estimate α onto the first principal component.
- If coplanarity was detected, project the four points on the plane spanned by the first two principal components. Also project the image pixel for which we wish to estimate α onto this plane.
- We have noted previously that if the lines are neither collinear nor coplanar, we will still use the coplanar solution. In this case, we will transform the points as in the coplanar case.

Once the points are transformed, we solve for α depending on the case we have detected.

4.3.5 Solving for α: The Collinear case

In the collinear case, it is not possible to use the full line model. Hence we will use the line segment model. Given the foreground line segment and the background line segment, as well as the image pixel I for which we wish to estimate α – all collinear after the transformations we performed in section 4.3.4 – we do the following:

1. Take F' as the point on the foreground line segment closest to the image pixel I.

2. Take B' as the point on the background line segment closest to the image pixel I.

3. Take α as the projection value:

$$\alpha = \frac{(I - B')^T(F' - B')}{(F' - B')^T(F' - B')}$$

A simple depiction of this process can be shown in figure 4-6.

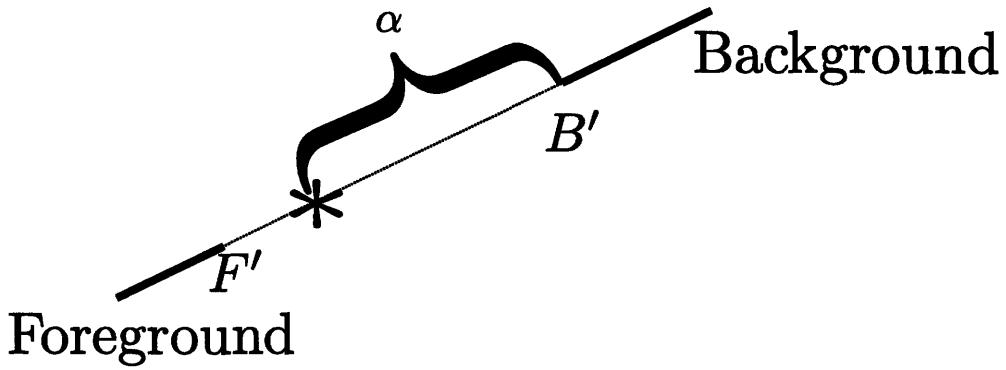


Figure 4-6: Solving the collinear case: A visualization. The asterisk denotes the image pixel, while the solid lines denote the foreground and background line segments.

If F' and B' are identical, the above process fails; in fact, F' and B' can only be identical when the foreground and background line segments overlap. In this case, we take F' and B' to be the means of the foreground and background lines respectively (the projection of the means of the sample foreground and background pixels onto their respective lines). If F' and B' are still identical after this, we set $\alpha = 0.5$.

4.3.6 Solving for α: The Coplanar case

In the coplanar case, both the full line model and the line segment model can be used. We will discuss the motivation and solution for both in the following sections.

As before, we shall assume that after the transformations discussed in section 4.3.4, the lines and the image pixel lie on the same plane.

The Full Line Model

In the full line model, we can use any point on the entire foreground (background) line as the F' (B') pixel values. Following the standard compositing equation (1.1), the image pixel I lies on the line connecting the F' and B' pixels. To incorporate shading, we want the foreground (background) shade that is closest to the pixel. Hence, our problem statement is:

> Find the point F' on the foreground line and the point B' on the background line that minimizes $||F' - I|| + ||B' - I||$ such that the $F' - B'$ line passes through I.

A visual interpretation can be found in figure 4-7.

Another justification for this optimization condition is as follows: Another possible way to incorporate shading is to find the foreground and background shades that are closest to each other, that is, to find the points F' and B' such that $||F' - B'||$ is minimal and the $F' - B'$ line passes through I. In the case when I lies between the foreground and background distributions, as is usually the case, this is equivalent to the above formulation.

Since all the points and lines are coplanar, we will first convert them to two-dimensional cartesian coordinates in a manner such that the image pixel I is the origin in the new two-dimensional coordinate system. This conversion transformation is elementary and is described in Appendix A.4.3; since this transformation preserves distances and angles, we can solve the problem in the transformed two-dimensional space, undo the transformation, and preserve the correct solution in the original three dimensional space. From this point on, we will describe the solution in two-dimensional space.

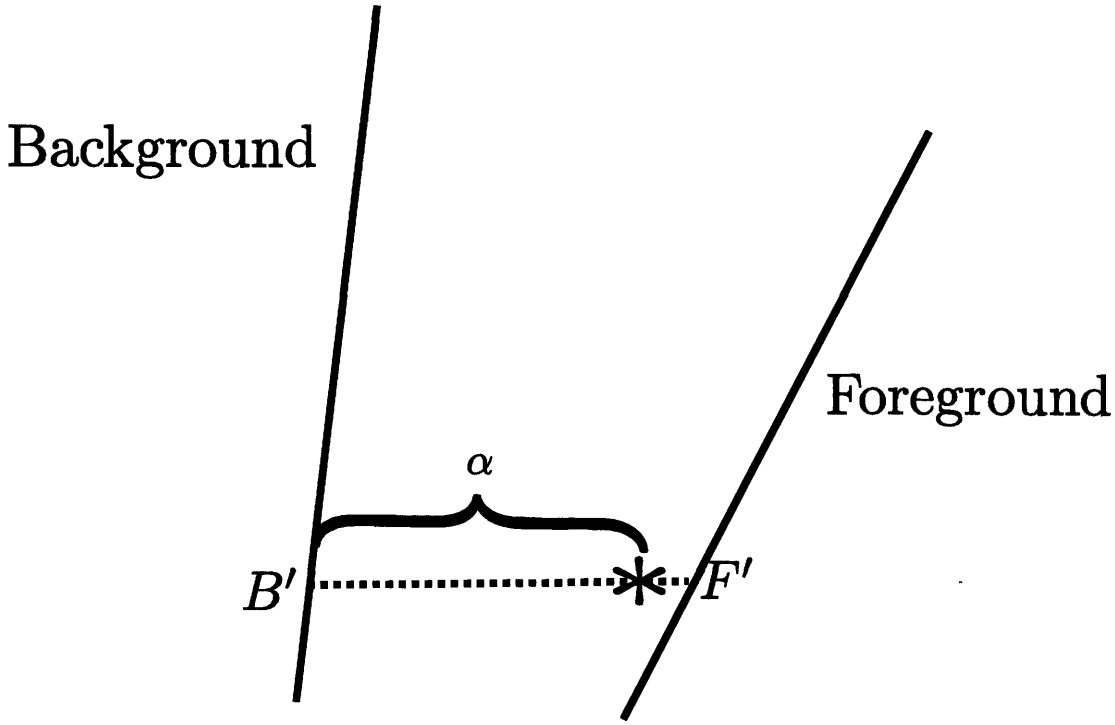


Figure 4-7: Solving the coplanar case with the full line model: A visualization. The asterisk denotes the image pixel, while the solid lines denote the foreground and background line segments.

Next, we will assume that one of the foreground or background lines is vertical. If not, we may rotate the space such that one of the lines becomes vertical; such a transformation is described in Appendix A.4.2. This transformation preserves distances and angles, so solving the problem in the transformed space is equivalent to solving it in the original space. This assumption of a vertical line is very important; we will discuss its importance later.

Now the image pixel I is the origin, so we require the $F' - B'$ line to pass through the origin. If one of the lines – suppose it is the foreground line – passes through the origin, then F' is the origin, and B' is the projection of the origin onto the background line. A similar solution can be found if the background line passes through the origin.

Next, if the two lines are parallel, the solution can be found by projecting the origin onto both lines. Now we will assume that the lines are not parallel and neither line passes through the origin. Since the pixel I is the origin, and we wish to minimize the sum of its distance from F' and B', which are points on the foreground and background lines respectively, we can restate the problem statement as follows:

> Given two lines in Cartesian space, one of which is vertical, find a third line passing through the origin that intersect with the two lines at points $\mathbf{p_1}$ and $\mathbf{p_2}$ such that $||\mathbf{p_1}|| + ||\mathbf{p_2}||$ is minimized.

Since one line is vertical and does not pass through the origin, the solution cannot be a vertical line. Hence, we may write the solution line as $y = mx$, where the slope m is the desired quantity to be found. Furthermore, since the lines are not parallel and one line is vertical, we may write the equation of the non-vertical line as $y = m_1x + c_1$ and the equation of the vertical line as $x = c_2$. We will assume that c_2 is non-negative; if it is negative, we can always rotate the plane by π.

The intersection of the solution line $y = mx$ with the line $y = m_1x + c_1$ is

$$\mathbf{p_1} = \left(\frac{c_1}{m - m_1}, \frac{mc_1}{m - m_1}\right) \tag{4.1}$$

and the intersection with the line $x = c_2$ is

$$\mathbf{p_2} = (c_2, mc_2) \tag{4.2}$$

Since c_2 is non-negative, we can write the objective function as

$$||\mathbf{p_1}|| + ||\mathbf{p_2}|| = \left|\frac{c_1}{m - m_1}\right| \sqrt{1 + m^2} + c_2\sqrt{1 + m^2} = \sqrt{1 + m^2}\left(c_2 + \left|\frac{c_1}{m - m_1}\right|\right) \tag{4.3}$$

There are two cases: First consider the case when $\frac{c_1}{m-m_1} \geq 0$, so that we may rewrite (4.3) as

$$\sqrt{1+m^2}\left(c_2 + \frac{c_1}{m-m_1}\right)$$

Taking the first order condition with respect to m yields

$$\left(c_2 + \frac{c_1}{m-m_1}\right)\frac{m}{\sqrt{1+m^2}} - \frac{c_1\sqrt{1+m^2}}{(m-m_1)^2} = 0$$

Since neither line passes through the origin, the answer cannot be $m = m_1$ and therefore we multiply throughout by $\sqrt{1+m^2}(m-m_1)^2$:

$$m\left(c_2(m-m_1)^2 + c_1(m-m_1)\right) - c_1(1+m^2) = 0$$

Since neither line passes through the origin, $c_2 \neq 0$. Thus we divide throughout by c_2 and simplify:

$$m^3 + m^2(-2m_1) + m\left(m_1^2 - m_1\frac{c_1}{c_2}\right) - \frac{c_1}{c_2} = 0 \tag{4.4}$$

This cubic equation in m has at least one and at most three real roots, and there exists a closed form solution to find all its roots (for example, see [20]).

Now, we consider the second case, when $\frac{c_1}{m-m_1} \leq 0$. In this case, we rewrite (4.3) as

$$\sqrt{1+m^2}\left(c_2 - \frac{c_1}{m-m_1}\right)$$

and the corresponding cubic equation that is the counterpart of (4.4) is

$$m^3 + m^2(-2m_1) + m\left(m_1^2 + m_1\frac{c_1}{c_2}\right) + \frac{c_1}{c_2} = 0 \tag{4.5}$$

Together, solving (4.4) and (4.5) gives us between two and six real roots. We can then try with each real root to find the choice of m that minimizes our objective

function (4.3). In the following paragraph, we demonstrate that there will be a real root of (4.4) satisfying $\frac{c_1}{m-m_1} \geq 0$ and a real root of (4.5) satisfying $\frac{c_1}{m-m_1} \leq 0$, which is sufficient to justify the assumptions of these two equations.

We quickly discuss the existence of an interior solution to the minimization problem and why this procedure works. Consider the geometric interpretation of the objective function. Since one of the lines is vertical and does not pass through the origin, as m tends towards both ∞ and $-\infty$, the solution line becomes vertical and the objective function tends towards ∞; for the same reasoning with the non-vertical line, the same is true as m tends towards m_1 from either direction. Since the objective function (4.3) is differentiable in m in the regions $(-\infty, m_1)$ and (m_1, ∞) and tends towards ∞ at both boundaries in both regions, it follows that the solution to the minimization problem can be found in the interior of these regions and there exists a local minimum, solvable by taking a first order condition, in each of the regions $(-\infty, m_1)$ and (m_1, ∞). Since $\frac{c_1}{m-m_1}$ has different signs in these two regions, it immediately follows that we can find the local minimum in the region where $\frac{c_1}{m-m_1} \geq 0$ by solving (4.4) and the region where $\frac{c_1}{m-m_1} \leq 0$ by solving (4.5), as both these equations are equivalent to solving the first order conditions in these regions.

Finally, we make a few notes on the importance of assuming that one line is vertical:

1. It allows us to guarantee that the solution line is not vertical, and therefore we can write it in the form $y = mx$.

2. It allows us to guarantee that the objective function (4.3) tends towards ∞ as m tends towards ∞ and $-\infty$, and therefore argue the existence of an interior solution to the minimization problem.

3. Suppose both lines are not vertical; the equation of the second line can then be written as $y = m_2x + c_2$. If we proceed in the same way as before, the equivalent first order condition that is the counterpart of (4.4) is quartic instead of cubic. While quartic equations also have closed form solutions, there are two problems

with a quartic equation: Firstly, the algorithm is far more complicated, and secondly and more importantly, unlike a cubic equation a quartic equation is not guaranteed to have any real roots and thus there may be no solutions to the problem!

Once we have found the solution slope m and hence the solution line $y = mx$, the value of F' is simply the intersection of this line with the foreground line and B' is the intersection of this line with the background line. We can calculate the value of α without undoing any of the transformations performed, because all our transformations preserve lines and angles:

$$\alpha = \frac{(I - B')^T(F' - B')}{(F' - B')^T(F' - B')}$$

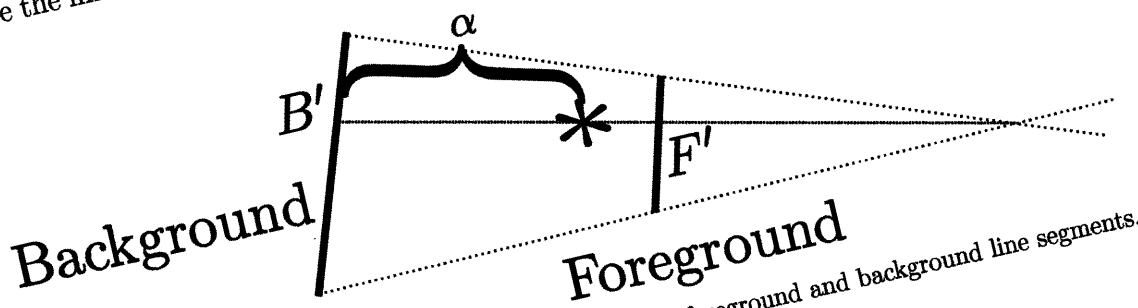
The Line Segment Model

In the line segment model, we want to find F' and B' that lie on the foreground and background line segments respectively such that the $F' - B'$ line passes through the image pixel. If this is not possible, we may extend both the foreground and background line segments by as little as possible, but proportionately with their length – so that we do not extend one line greatly and the other line by very little – so that this condition is satisfied.

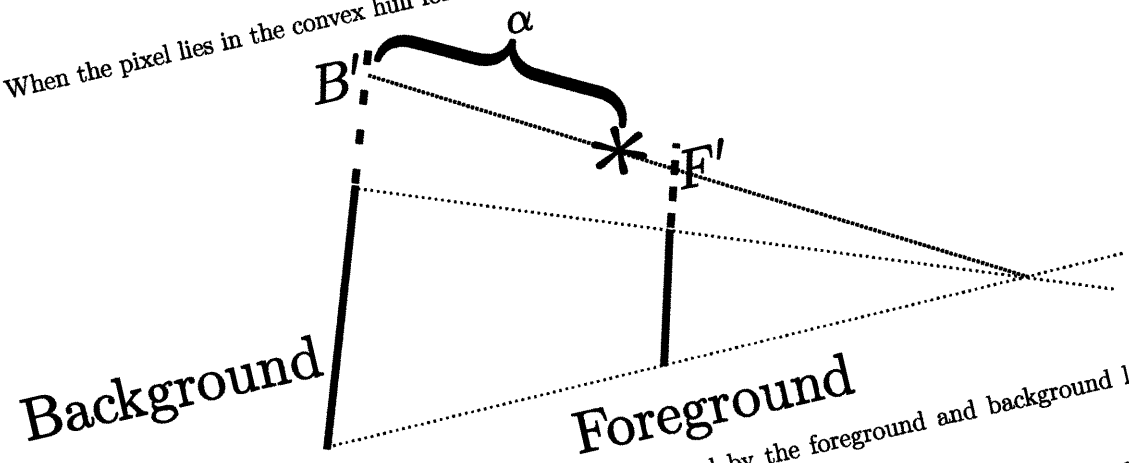
We present a geometric construction that achieves this goal. First, we join one endpoint of the foreground line segment with one endpoint of the background line segment to create a line, and join the other endpoints of the foreground and background line segments to create a second line. We will assume that these lines do not intersect in the interior of the convex hull formed by the foreground and background line segments; if not, we can always switch the endpoints around.

Take the intersection of the two lines, and join this intersection with the image pixel I to form another line. The intersection of this line with the foreground and background lines are the values of F' and B' respectively. Figure 4-8 shows this construction.

We can perform this geometric construction in closed form by using homogenous coordinates, which are described in Appendix A.5. One advantage of using homogenous coordinates is that it takes care of the special case when the lines formed by joining the endpoints of the foreground and background line segments are parallel; in this case the line from its "intersection" to the image pixel I will be parallel to these lines.



(a) When the pixel lies in the convex hull formed by the foreground and background line segments.



(b) When the pixel does not lie in the convex hull formed by the foreground and background line segments.

Figure 4-8: Solving the coplanar case with the full line model: A visualization. The asterisk denotes the image pixel, while the solid lines denote the foreground and background line segments. In figure 4-8(b), the pixel does not lie in the convex hull formed by the foreground and background line segments; however, our construction allows the extension of the line segments (dashed) in a reasonable way.

Figure 4-8 shows two desirable properties of this construction: Figure 4-8(a) shows that in the case when there is no need to extend the lines – or equivalently, when

the image pixel lies in the convex hull formed by the foreground and background line segments – the construction gives a good solution. Figure 4-8(b) shows that when there is a need to extend the lines – or equivalently, when the image pixel does not lie in the convex hull – the construction extends the lines in a manner such that the longer line (the background line in figure 4-8(b)) is extended longer than the shorter line; in other words, the lines are extended proportionately to their lengths. For example, if the line segments were formed by taking standard deviations, then the lines are extended proportional to their standard deviation, which is a desirable property.

As with all previous cases, once F' and B' are found, we can calculate α:

$$\alpha = \frac{(I - B')^T(F' - B')}{(F' - B')^T(F' - B')}$$

4.3.7 Solving for α: Neither Collinear nor Coplanar

Although this case will never be used – because we force coplanarity and utilize the solution in section 4.3.6 – we will briefly describe the solution for the sake of completeness.

In this scenario, we will use the full line model. The line segment model is not applicable here: As long as the lines are not coplanar, there exists at most one solution – and there may be none, as we will see – for F' on the foreground line and B' on the background line such that the $F' - B'$ line passes through the image pixel. Thus, restricting possible solutions to line segments does nothing.

We first translate the coordinates such that the image pixel lies on the origin. Thus, as in the full line model for the coplanar case described in section 4.3.6, we wish to find pixels F' on the foreground line and B' on the background line such that the $F' - B'$ line passes through the origin. If one line – say the foreground line – passes through the origin, then F' is the origin and B' is the projection of the origin on the background line. A similar solution can be obtained if the background line passes

through the origin. Hence we shall assume the lines do not pass through the origin.

We will briefly describe a simple geometric solution to this problem[2]. Consider the plane formed by the origin and the foreground line. Any line passing through the origin in this plane will intersect the foreground line, except the line parallel to the foreground line, and no other lines passing through the origin – that is, the lines passing through the origin but not in this plane – will intersect the foreground line. Therefore, the solution for B' is given by the intersection of the background line with this plane. If the background line is parallel with this plane, there is no solution.

We can then obtain F' by taking the intersection of the line passing from B' through the origin with the foreground line. If no intersection exists, then no solution exists, otherwise this procedure gives the unique solution for F' and B'. As before, we can calculate α from F' and B':

$$\alpha = \frac{(I - B')^T(F' - B')}{(F' - B')^T(F' - B')}$$

4.3.8 Multiple Foreground/Background Cluster Pairs

Up to this point, we have described how we can obtain α from a single pair of foreground/background lines. Since we have partitioned the foreground and background pixels into multiple lines, we need to choose between different solutions of F', B' and α. For this, we utilize a scoring system, where the higher the score, the more desirable the solution. We then pick the pair of lines, and its associated solution for F', B' and α, that has the highest score.

Our scoring system is heuristic, but is based on the following guidelines:

1. To solve for α, we had to project the pixel onto a plane or line, without account-

[2]An algebraic solution that uses line equations is easier to code and derive, but lacks the geometric intuition to understand its problem cases when no solution can be found. Since we will not be utilizing the non-collinear, non-coplanar case, we will not describe this solution.

ing for the projection distance. We want to put a penalty for a high projection distance.

2. To allow for shading, we should allow the calculated F' and B' to deviate substantially from the mean of the foreground and background line distributions without penalty. The amount of deviation allowed should depend on the standard deviation of the distributions.

3. There should be a penalty if F' and B' deviate sufficiently far from the mean of the foreground and background line distributions. However, if the calculated F' and B' are extremely far from the foreground and background means respectively, increasing their distance from their respective means should not increase the penalty substantially.

   This is to prevent this penalty from essentially dominating the choice of foreground and background cluster pair when all foreground (background) clusters have F' (B') far from the foreground (background) mean – as may be the case for a point that is spatially far from the foreground and/or background edges of the trimap. For example, if we use a squared distance penalty, increasing the distance of F' (B') from the foreground (background) mean by 1 – a relatively small amount – can increase the penalty by a great amount if the distance is already far.

4. We would like the solved α to be sparse i.e. near 0 or 1. If α is too small or too large i.e. much smaller than 0 or much larger than 1, there should be a penalty as well.

Following these guidelines, we devise the following scoring system:

$$Score = -\Bigg(Penalty_{proj}(d) + Penalty_{FB}\left(\frac{||F' - \mu_F||}{\sigma_f}\right) + Penalty_{FB}\left(\frac{||B' - \mu_B||}{\sigma_b}\right) + Penalty_{\alpha}(\alpha)\Bigg) \quad (4.6)$$

In the above, d represents the projection distance in RGB space, F' and B' represent the calculated F' and B' values, μ_F and μ_B are the projections of the means of the foreground and background distributions onto their respective lines, and σ_f and σ_b are the standard deviations of the projection values of the foreground and background distributions. Hence, in the foreground and background penalties, we take the deviations of the estimated values from their means, normalized by their standard deviations; the foreground and background penalty functions take the same form.

We describe each of the penalty functions as follows: The projection penalty is given by

$$Penalty_{proj}(d) = \frac{d^2}{\sigma_d^2} \tag{4.7}$$

where σ_d represents the standard deviation of the projection distance. We use $\sigma_d = 2$ units in RGB space. The foreground and background penalties are given by

$$Penalty_{FB}(x) = \begin{cases} 0 & x \leq thresh \\ k(x - thresh) & x > thresh \end{cases} \tag{4.8}$$

where $thresh$ represents a threshold within which we allow shading without any penalty, and k is a linear penalty factor once the amount of shading exceeds $thresh$ standard deviations away from the mean. We use $thresh = 2$ and $k = 2$. Finally, the penalty for α, which drives the solution towards sparseness, is given by

$$Penalty_{\alpha}(\alpha) = cx^2, \; x = \min(|\alpha - 0|, |\alpha - 1|) \tag{4.9}$$

where x represents the deviation of α from 0 or 1 and c is a scaling factor. We use $c = 20$.

Combining equations (4.6), (4.7), (4.8) and (4.9) gives us our scoring system for a given foreground/background pair of clusters after F', B' and α have been estimated,

and we choose the foreground/background pair of clusters with the highest score.

While our scoring system may seem fairly ad-hoc and heuristic – especially with the choice of similarly arbitrary parameters – it works well in practice. Nevertheless, there is a lot of scope for future work in deriving a better scoring system; section 7.2 discusses this issue.

4.3.9 Further Optimizations and Special Cases

In this section, we discuss some optimizations that we perform to improve both accuracy and efficiency.

Gamma Correction

Although the Red, Green and Blue channels in a color image represent the red, green and blue intensities respectively, a computer display generally converts the input signal (the red/green/blue channels) into output light intensities in a *non-linear* manner, so that the image appears different. A standard model that models this non-linear relationship is

$$O \propto I^{\gamma}$$

where O represents the output light intensity and I represents the input value. γ is a display-dependent parameter that is usually around 2.2. To correct for this non-linear relationship, the RGB values stored in image files are usually pre-corrected:

$$I \propto A^{1/\gamma}$$

where A is the actual RGB value that is desired. This is known as *gamma correction* [26]. Since our model of shading assumes that the foreground and background distributions form a straight line in RGB space, we undo this gamma correction before performing Segment-Based Matting, using $\gamma = 2.2$.

Foreground and Background Detection: Overlapping Foreground and Background Clusters

In areas where the foreground and background have very similar distributions, the foreground and background clusters may overlap partially. This can be a problem when the pixel we wish to estimate for is near both these clusters in RGB space: It is possible for it to be detected as foreground but not background, or vice versa.

To mitigate this problem, we pre-detect if any foreground and background clusters overlap: For any foreground/background pair, we check if any of the pixels in the background cluster would be considered a foreground pixel, or any of the pixels in the foreground cluster would be considered a background pixel, following the tests described in section 4.3.2. If so, we assume the clusters overlap.

When a pixel is classified as foreground (background) following a given foreground (background) cluster, we check if that foreground (background) cluster overlaps with any clusters of the opposite type. If so, we also classify that pixel as the opposite type. This prevents us from assigning $\alpha = 0$ or $\alpha = 1$ immediately, and allows us to mark the pixel as being ambiguous. This will be useful, not directly in the Segment-Based Matting algorithm in this chapter, but in our approach described in chapter 5 that uses inpainting to resolve ambiguities in mattes.

Foreground and Background Detection: Pixels lying beyond the foreground/background clusters

It is usually assumed in Segment-Based Matting that the image pixels being estimated are either between the foreground and background clusters, or are sufficiently close to one of them. It may be possible that an image pixel lies "beyond" either one of these clusters, in which case we will not classify it as a foreground or background pixel, even though it should be one of them. Figure 4-9 shows a visual idealization of this scenario, where the pixel should belong to the foreground even though it is not

in the foreground distribution.

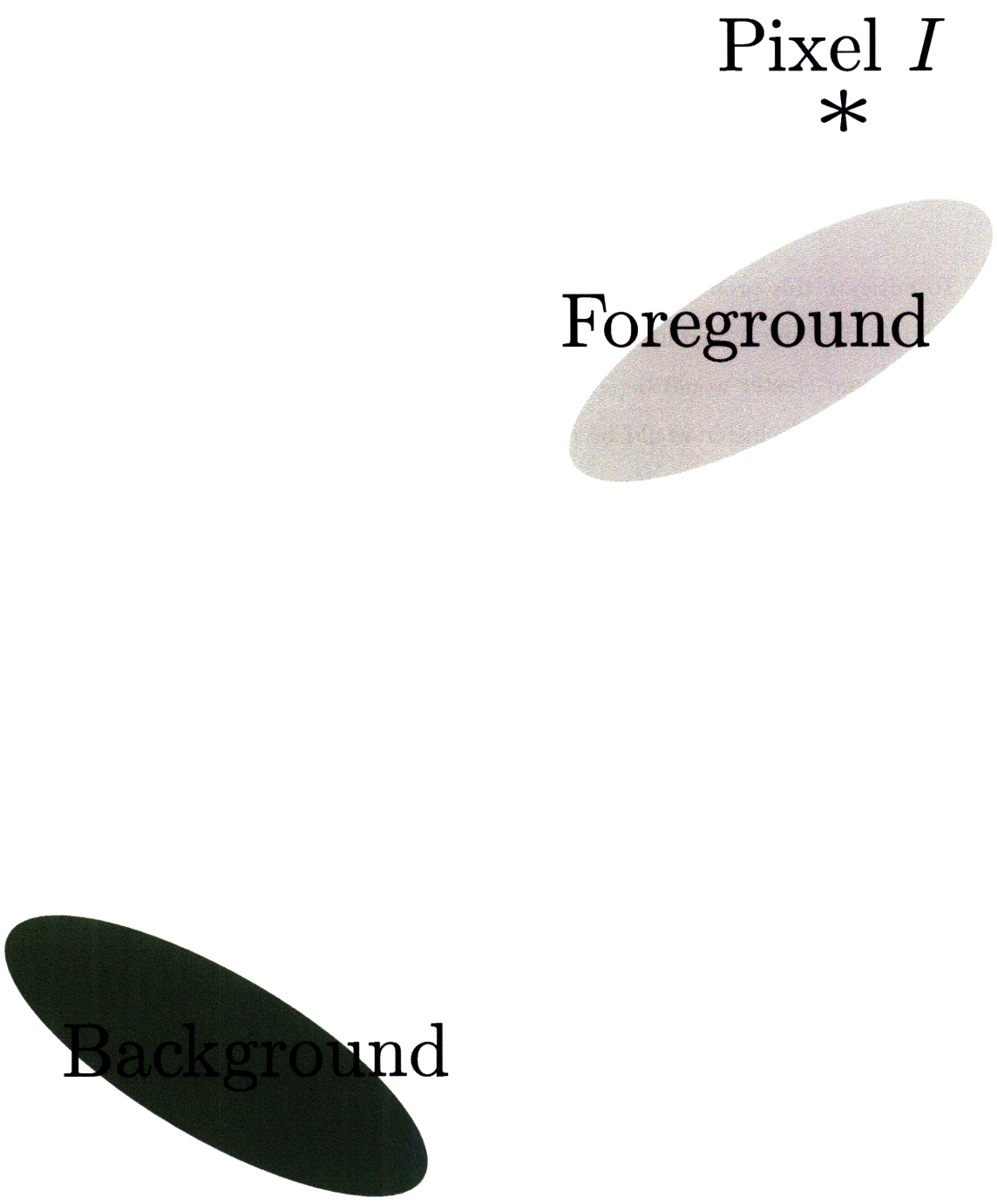


Figure 4-9: Pixels lying beyond the foreground or background clusters: The pixel should be classified as foreground.

Our approach to solve this is as follows:

1. After partitioning foreground and background pixels into clusters, we take the distance in RGB space of the mean of every cluster to the image pixel that we wish to estimate α for. We denote the smallest distance among all the foreground clusters as d_f and the smallest distance among all the background clusters as d_b.

2. We classify the pixel as foreground if:

   (a) $k_1 d_f \leq d_b$ AND

   (b) $d_b \geq k_2$

   where k_1 and k_2 are parameters we set. We classify the pixel as background if the above conditions hold after swapping d_b and d_f. The first condition ensures that the pixel is much closer to the foreground than the background, and the second ensures that the pixel is sufficiently far from the background – this prevents situations where the foreground and background are very close to each other and d_f and d_b are both very small, in which case – following the previous optimization described – we classify them as both foreground and background. We set $k_1 = 2.5$ and $k_2 = 10$ units in RGB space.

Single points in RGB color space.

It is not unusual for some clusters to be single points in RGB space. As an optimization, we ignore any cluster that contains a single point, as these are usually outliers in the distribution. However, it can also be the case that these clusters contain *multiple points, all with the same RGB value.* We do not ignore these cases; however, these clusters *do not generate a line distribution*!

Nevertheless, we can deal with these clusters fairly easily:

1. When we have two clusters that are both single points in RGB space, we apply the collinear solution described in section 4.3.5; in this case the F' and B' values are the single points. We also apply the collinear solution when one cluster is

a single point in RGB space, the other cluster is a line, and the line and the point are collinear.

2. When we have one cluster that is a single point in RGB space and the other cluster is a line, and the line and the point are not collinear, we apply the coplanar solution described in section 4.3.6. There are no modifications to the line segment model; in the full line model, we use the line passing through the single point in the direction from the foreground to the background cluster as the line distribution of the single point cluster; this models shading between the foreground and background.

4.4 Improving the Orchard-Bouman Clustering Algorithm

In section 3.2.4, we mentioned that a possible source of inaccuracy in Bayesian Matting was that it fixed the number of foreground and background clusters. In this section, we describe a heuristic modification to the Orchard-Bouman algorithm that dynamically decides when to stop splitting clusters into new clusters, and thus intelligently deduces the correct number of clusters.

Recall that the Orchard-Bouman algorithm greedily chooses a cluster to split based on the variance in its direction of largest variance – equivalently, the largest eigenvalue of the cluster's covariance matrix (see Appendix C) – and splits the cluster along the direction of largest variance, which is the largest eigenvector of the cluster's covariance matrix. Intuitively, we can use these variance values to deduce when to stop splitting: If the variance of the split clusters are not substantially smaller than the variance of the original cluster, we should stop splitting.

Now we make this intuition concrete. To understand our modifications, consider the following cases we wish to enforce:

1. If the pixels are distributed relatively evenly throughout the cluster, do not split.

2. If the variance of the cluster is not high, do not split. This rule is especially important if there are a small number of pixels: If there a very small number of pixels, splitting it will reduce the variance drastically, especially if after the split, there is a cluster with only one pixel – which has zero variance – and thus any rule that simply looks at the ratios of variances before and after will always split, even when there should not be splitting.

3. If the variance is much higher in one direction than in other directions, split. This suggests that there are multiple clusters in that direction.

4. We need to be able to split if there could be further splits in nearly-orthogonal directions on the split clusters; this ensures if there are multiple clusters that would require splits in different directions, we will still split – otherwise we might not split based on rule 1 as the variances in different directions may be fairly even (see figure 4-10(d) for an illustration)

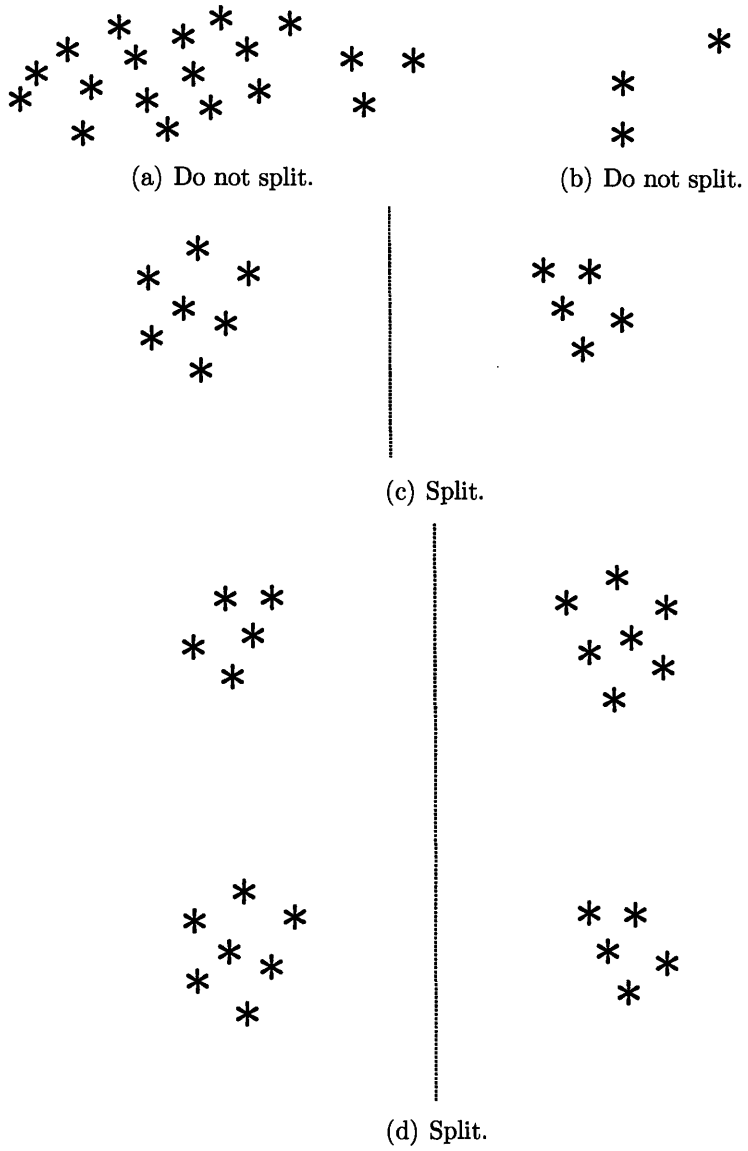
These cases are demonstrated in figures 4-10(a), 4-10(b), 4-10(c) and 4-10(d) respectively.

We first begin with some basic facts to help us calculate the variance in a given direction and the total variance of a cluster:

1. Given the covariance matrix Σ of pixels in a cluster, we can calculate the variance in a given direction $\mathbf{v}$ – where $\mathbf{v}$ is a unit vector – as $\mathbf{v}^T\Sigma\mathbf{v}$.

2. The largest eigenvector of Σ is the direction of largest variance, and the variance in that direction is its corresponding eigenvalue.

3. The total variance of the cluster is given by the sum of the eigenvalues of its covariance matrix.



(a) Do not split.

(b) Do not split.

(c) Split.

(d) Split.

Figure 4-10: The above plots show some cases and describe whether we wish to split them or not. If a split is preferred, we denote the axis of splitting with a dotted line. In figure 4-10(b), we want to avoid splitting clusters with a small number of pixels if the variance in the cluster is not high. In figure 4-10(d), we want to split the clusters so that we can perform further splitting steps on the split clusters created.

For derivations of these facts, see definition B.7 in Appendix B and Appendix C.

In the original Orchard-Bouman algorithm, the algorithm repeatedly chooses the cluster with the largest eigenvalue of its covariance matrix and splits it until the desired number of clusters is obtained. Our modified algorithm performs three steps,

where we have emphasized (italicized) the areas that differ from the original algorithm:

1. Choose the cluster with the largest eigenvalue.
2. *Decide if the cluster should be split. If so, split it as usual; if not, discard it and mark it so that it is never chosen again in the future.*
3. Repeat until the maximum number of clusters n is reached or *there are no clusters available to be chosen.*

Thus, the main aspect of our modification is a procedure for deciding if a cluster should be split. The user specifies a set of parameters: τ, n, k_1, k_2, k_3, where τ is a maximum variance parameter, n is the maximum number of clusters allowed (this replaces the fixed number of clusters in the original Orchard-Bouman algorithm), and k_1, k_2 and k_3 are parameters that will be defined later in this description. The values of these parameters will depend on the particular application – which in our case is partitioning pixels into clusters in RGB space – although in general reasonable values of these parameters will result in good dynamic behavior. For our application, we use $\tau = 25, n = 6, k_1 = 0.8, k_2 = 0.25, k_3 = 0.1$. We now have:

Algorithm 4.7. Deciding if a cluster should be split. We follow the following set of rules to decide if a cluster should be split:

1. Do not split if the total cluster variance is less than τ.
2. Split if the largest eigenvalue of its covariance matrix is more than k_1 times the total cluster variance.
3. Perform a trial split of the cluster – which will be denoted as the *original cluster* – to obtain two *split clusters*. We will perform the split if at least one of the split clusters satisfies the following property:

   Let $\mathbf{v}$ be the normalized eigenvector associated with the largest eigenvalue of the covariance matrix of the original cluster – equivalently, $\mathbf{v}$ is the direction of splitting. Let λ be the variance of the *split cluster*

in the direction of $\mathbf{v}$, and η be the variance of the *original cluster* in the direction of $\mathbf{v}$. Let ν be the largest eigenvalue of the covariance matrix of the *split cluster*. The property is satisfied if $\lambda < k_2\nu$ and $\lambda < k_3\eta$.

What do these rules do? The first rule prevents splitting when the total cluster variance is small, as illustrated in figure 4-10(b). The second rule splits when a large majority of the variance (recall that in a perfect spherical cluster, the largest eigenvalue is one-third of the total variance and in a perfect two-dimensional circular cluster, the largest eigenvalue is half the total variance) can be explained by one direction, which suggests – not always, but it is a good indicator – that the clusters are arranged in a manner similar to figure 4-10(c).

If the first two rules do not apply, it then becomes a matter of distinguishing between the cases in figure 4-10(a), where we do not split, and figure 4-10(d), where we split. There are two features that distinguish the cases: In 4-10(a), if we split, the variance along the direction of splitting does not decrease substantially, and the largest eigenvalues of the covariance matrices of the split clusters are small. In 4-10(b), after splitting, the variances of the split clusters along the direction of splitting decreases substantially, and the largest eigenvalue of the covariance matrices of the split cluster is large because *there is a different direction of high variance.*

The third rule in the algorithm tries to decide if at least one of the split clusters benefitted substantially from the split of the original cluster, in which case the split should proceed. To decide this, it uses the two properties above: The cluster should have a much smaller variance in the direction of the split, and it should still have a different direction where it has a relatively high variance.

Remark 4.8. One valid criticism of remaining within the Orchard-Bouman algorithm is that in Segment-Based Matting we model distributions as lines and therefore should

use a clustering mechanism that tries to cluster points into line segments (The work by Omer and Werman [21] on color lines cannot be directly used because it is relatively inefficient and assumes that the lines pass through the origin in RGB space, which is not necessarily the case in our application). Empirically, our modified Orchard-Bouman algorithm does perform reasonably well in this aspect, although this is certainly an area for future improvement.

Chapter 5

Using Inpainting/Texture Synthesis to Resolve Ambiguities in Matting

In this chapter, we describe a new approach that uses other techniques in image processing – inpainting and texture synthesis – to resolve ambiguities in image matting. By combining inpainting with matting, we can increase the accuracy of the generated alpha matte.

Inpainting [3] is the process of filling in holes in images – using ideas similar to contour continuations – such that the resulting image looks untampered; it is thus an image restoration technique. Texture synthesis [12] is a technique that allows a small texture element to be replicated smoothly over an large area. We use an application of texture synthesis introduced by Criminisi et al [11] to fill in holes in images with textures.

We will first provide a brief introduction to inpainting and texture synthesis in section 5.1. Section 5.2 provides a brief motivation behind our approach. Section 5.3 describes the approach. We present two applications of our approach: In section 5.4, we present its application to Bayesian Matting, and in section 5.5 we present its application to our Segment-Based Matting algorithm described in chapter 4. Although we have

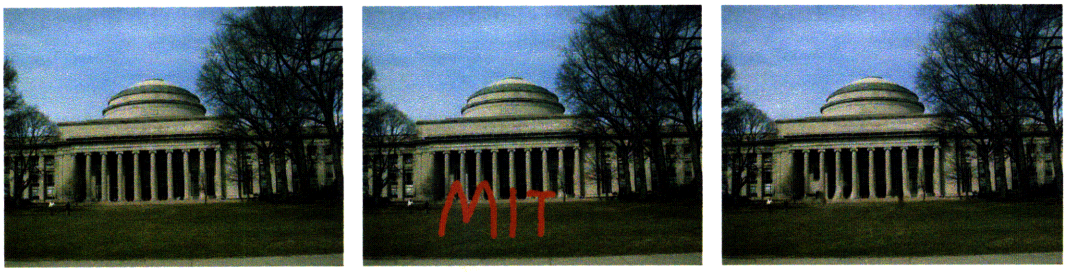
presented our approach for just two matting algorithms, the approach is fairly general and can be easily adapted to many matting algorithms.

5.1 Background: Inpainting and Texture Synthesis

To help explain our new approach that uses inpainting or texture synthesis to resolve ambiguities in image matting, in this section we will provide a brief introduction to both inpainting and texture synthesis.

5.1.1 Inpainting

As we have mentioned previously, inpainting is the restoration of damaged areas in an image. An example can be found in figure 5-1, where a damaged image is restored using Bertalmio's inpainting algorithm [3].



(a) Original Image (b) Marked Image (c) Restoration over Marked Area using Inpainting

Figure 5-1: An example of inpainting: An image is scribbled over and then restored using inpainting.

Intuitively, the algorithm attempts to maintain the continuity of contours from the areas surrounding the damaged region, so that the restored image looks natural. The algorithm tries to extend isophotes – lines of constant intensity – from the surrounding areas into the damaged region in a natural and continuous way, while progressively curving the lines of propagation as they enter the damaged area, to prevent them

from crossing.

To this end, the following two steps are repeated until convergence is reached.

1. Propagate isophotes into the damaged area.

2. Curve the lines of propagation to prevent them from crossing.

The propagation mechanism is an iterative process that progressively propagates intensity information into the damaged region. For a pixel p in the damaged region, we denote $I^t(p)$ as its value at iteration t. The propagation mechanism performs the iterative update step

$$(I^{t+1}(p) - I^t(p)) \propto \delta \mathbf{L}^t(p) \cdot \mathbf{N}^t(p)$$

where $\mathbf{L}^t(p)$ and $\mathbf{N}^t(p)$ are vectors that represent the intensity information and the direction of propagation for pixel p at iteration t. Intuitively, this is the desired formulation – to propagate the change of the intensity information in the direction of propagation, such that at steady state, when there are no changes to be made, the intensity information has been entirely propagated in the direction of propagation. [3] formulates the intensity information $\mathbf{L}^t(p)$ as the Laplacian at pixel p, and the direction of propagation $\mathbf{N}^t(p)$ as the direction perpendicular to the image gradient at pixel p.

To curve the lines of propagation, the algorithm applies a discretized version of anisotropic diffusion [24], which is a partial differential equation. This allows the lines to be curved without losing sharpness in the image.

5.1.2 Texture Synthesis

Texture synthesis is concerned with the creation of larger textures from smaller texture components. An example can be found in figure 5-2.

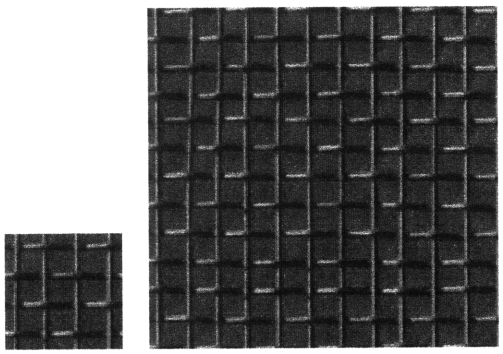


Figure 5-2: An example of texture synthesis: The example texture on the left is tiled in a smooth way to form the image on the right. Image from [12].

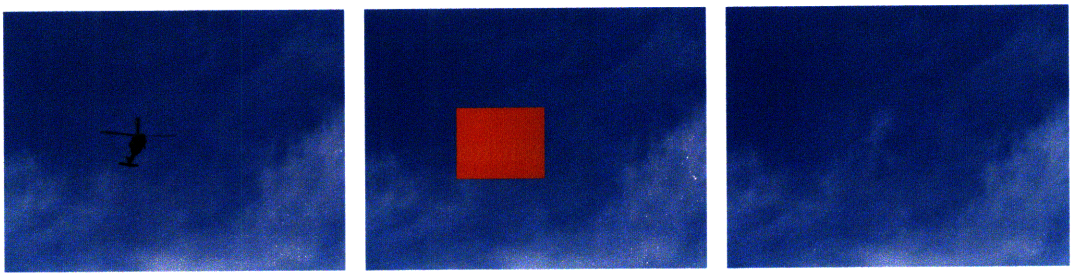
A simple texture synthesis algorithm proposed by Efros and Leung [12] proceeds as follows: From a small seed patch taken from the original texture element, grow the larger texture pixel by pixel outward. To synthesize a pixel, we consider its immediate spatial neighborhood; there would have been some pixels in the neighborhood that were previously synthesized and some that were not. Let X be the set of pixels in the neighborhood that were previously synthesized. The algorithm looks at every possible patch that has the same shape as X in the original texture and considers all the patches that are very similar to X, using a measure of similarity. Next to each of these patches in the original texture, there exists a pixel that would correspond to the pixel we are synthesizing. The algorithm then forms a weighted histogram of these pixels – where the weight is its corresponding patch's similarity to X – and samples from this histogram to synthesize the pixel.

It is not difficult to presume that a similar technique can be used to fill in holes in regions, by synthesizing textures that fit the desired area. In this case, the source texture element could be obtained from the area around the hole or even the entire image, minus the holes, since the above algorithm only deals with small neighborhood patches around each pixel.

This is indeed the method proposed by Criminisi et al [11]. Their novel contribution is an *ordering* of filling in the hole. The texture synthesis method previously described

would synthesize pixels in the hole inwards, in onion-peel order. [11] suggests that at every iteration, the pixel in the hole that is chosen to be synthesized next maximizes a combination of confidence – how confident we are that the pixel can be synthesized accurately – and the strength of the isophotes flowing into the pixel. This ensures that pixels that can be synthesized accurately and propagate a large amount of information are synthesized first to increase the accuracy of synthesizing other pixels in the hole.

Once the pixel is chosen, it is synthesized in a similar manner – not exactly identical, but the differences do not matter in this discussion – to the texture synthesis method in [12] that we described previously. An example of running Criminisi et al's algorithm for filling in holes is shown in figure 5-3.



(a) Source Image

(b) Marked Area

(c) Filling in the marked area with Criminisi et al's texture synthesis-based inpainting algorithm

Figure 5-3: Removing a helicopter from the sky: An example of filling in holes with Criminisi et al's texture synthesis-based inpainting algorithm.

5.2 Motivation

Most matting techniques – including our Segment-Based matting algorithm described in chapter 4 – rely on a clear distinction between the foreground and background color distributions. When the foreground and background color distributions overlap, and the image pixel lies within the overlap, the resulting calculated value of α may be inaccurate. For example, as we have seen in section 3.2.3, this can result in discon-

tinuities in the matte generated by Bayesian Matting, as pixels in a small spatial area with minor differences in RGB coordinates can have large differences in their estimated values of α when the foreground and background color distributions are very close.

In this case, our intuition proceeds as follows: Color contrast is probably the most important factor to a human viewer in determining the foreground elements in an image, and this is what color-based matting algorithms try to use. However, when there is ambiguity in the foreground and background color distributions, humans use different visual techniques, such as contour continuation or texture similarities. Similarly, our approach uses image processing analogues such as inpainting and texture synthesis to fill in regions in the alpha matte where there is color ambiguity.

5.3 Approach

The approach we propose is simple and elegant and can be adapted to many matting algorithms:

1. For each pixel, determine if it is a foreground pixel.
2. Similarly, also determine if it is a background pixel.
3. If a pixel is foreground and not background, assign it $\alpha = 1.0$. Similarly, if it is background and not foreground, assign it $\alpha = 0.0$.
4. If a pixel is neither foreground nor background, perform the matting algorithm.
5. If a pixel is *both* foreground and background, mark it as ambiguous.
6. After all pixels are estimated, first smooth the areas around the regions marked as ambiguous in the alpha matte (see section 5.3.1), and then perform either inpainting or texture synthesis to fill in these ambiguous regions. Thus, the marked regions serve as "holes" in the matte which region-filling/image restoration algorithms such as inpainting or texture synthesis will fill in.

We note that the first few steps of this approach are very similar to parts of the Segment-Based Matting algorithm described in chapter 4. As we will note in section 5.5, extending Segment-Based Matting to use this approach is extremely straightforward.

Although our approach, in the context of using inpainting, does try to enforce continuity of the alpha matte, we note that in contrast to matting algorithms that directly enforce continuity and/or try to balance continuity and color contrast, we only enforce continuity in regions where color-based solutions are ambiguous and fail. This is an important difference: In our approach, inferring the matte from color data is more important, and we only resort to other methods where we cannot infer the matte from color data. We feel that this is a more principled approach than trying to incorporate both color data and enforce continuity at the same time.

5.3.1 Smoothing around Ambiguous Regions

From the above discussion, an ambiguous pixel is a sign of similar foreground and background color distributions, which also indicates that its nearby spatial regions also have similar local foreground and background color distributions. Since our foreground and background detection mechanisms are not perfect, we may miss detecting some nearby pixels that have similar foreground and background distributions and are incorrectly estimated.

These wrongly estimated pixels which are not marked as ambiguous will cause a problem during inpainting or texture synthesis, because both inpainting and texture synthesis rely on information from pixels surrounding the region to be filled, and if these pixels are wrongly estimated, the resulting solution can be very inaccurate.

Therefore, as a preprocessing step before inpainting or texture synthesis, we take a small region around the pixels marked as ambiguous and smooth this region to reduce discontinuities. We propose two possible smoothing approaches which we will briefly

present: A simple Markov Random Field approach or smoothing using Levin et al's closed form laplacian matting algorithm.

Remark 5.1. Smoothing can be done even in regions that are not marked as ambiguous. This can be useful when the user wishes a smoother matte or there are discontinuities in the generated matte caused by the independent estimation of pixels.

Therefore, smoothing can be done either on just the ambiguous regions of the alpha matte – as described here – or if we allow further user input, in other areas also specified by the user. The smoothing approaches that we propose are sufficiently fast that the additional user input can be done interactively: The user specifies the area where smoothing is required and any parameters (for example, in the MRF approach in section 5.3.1, the parameters σ_c and σ_d) for the smoothing algorithm in this area, and he can see the results almost immediately. This can be repeated, with different parameters, on different areas of the alpha matte, and is thus extremely useful when different parts of the matte requires different degrees of smoothing. Thus, with further user input, a much more accurate matte can be obtained.

Smoothing using a Markov Random Field

Smoothing on the generated alpha matte can be done using a Markov Random Field (MRF) (see section B.6 for an introduction to MRFs).

The goal of the MRF is to balance smoothness between adjacent values of α with maintaining a value of α as close as possible to its originally estimated value. We can therefore formulate an objective function of the combination of all α values in the unknown region – to be maximized – as a product of *connectivity* and *data* terms. The connectivity terms – between neighboring pixels – try to maintain continuity and smoothness, while the data terms – for individual pixels – try to keep the values of α close to the original value. With this objective function, the MRF tries to find

$$\alpha = \arg\max_{\alpha} \prod_{i,j \text{ are neighboring unknown pixels}} \Psi_{i,j}(\alpha_i, \alpha_j) \prod_{\text{unknown pixel } i} \Psi_i(\alpha_i) \qquad (5.1)$$

where $\alpha = (\alpha_1, \alpha_2, \ldots)$ is the combined α of all unknown pixels, $\Psi_{i,j}(\alpha_i, \alpha_j)$ is the connectivity term between two neighboring unknown pixels i and j, and $\Psi_i(\alpha_i)$ is the data term associated with pixel i. We formulate both the connectivity and data terms as univariate Gaussians: The connectivity term is

$$\Psi_{i,j}(\alpha_i, \alpha_j) = \exp\left(-\frac{(\alpha_i - \alpha_j)^2}{2\sigma_c^2}\right)$$

and the data term is

$$\Psi_i(\alpha_i) = \exp\left(-\frac{(\alpha_i - \alpha_{i,original})^2}{2\sigma_d^2}\right)$$

where σ_c is the connectivity standard deviation, σ_d is the data standard deviation, and $\alpha_{i,original}$ is the α that was estimated originally by the matting algorithm. The less σ_c is, the smoother the resulting solution, and the less σ_d is, the more the solution tries to adhere to the originally estimated values of α.

To find α over the entire image that maximizes the objective function (5.1), for each pixel i we discretize α_i to 17 uniformly-spaced levels between 0 and 1 inclusive, formulate the objective function (5.1) as a MRF problem and optimize using loopy belief propagation – see Appendix B.6 for details.

Smoothing using Levin et al's Closed Form Laplacian Matting Algorithm

From section 2.3, Levin et al's closed form laplacian matting algorithm is an approach that tries to obtain a relatively smooth alpha matte subject to boundary conditions given by the marked foreground and background regions. To utilize this approach in our smoothing step, we simply use the closed form laplacian matting algorithm to

solve for the alpha matte in the region to be smoothed, with boundary conditions given by the estimated values of α around the region.

The degree of smoothing can be adjusted by the parameter ϵ given in equation (2.3); the higher the value of ϵ, the smoother the resulting solution.

5.4 Application to Bayesian Matting

To apply the technique to Bayesian Matting, we need to detect if a pixel belongs to the foreground and/or the background. We can then inpaint or apply texture synthesis in the ambiguous regions which we believe are both foreground and background.

We follow a similar testing framework as described in section 4.3.2 for Segment-Based Matting: A pixel belongs to the foreground if it belongs to the distribution of one of the foreground clusters, and similarly it belongs to the background if it belongs to one of the background clusters. All that remains is to describe how to test if a pixel belongs to a cluster.

In Bayesian Matting, a cluster is a three-dimensional Gaussian described by a mean μ and a covariance matrix Σ. Since the distribution is three-dimensional in three-dimensional RGB space, there is no need to test projection distance as in section 4.3.2, where a one-dimensional distribution was fitted in three-dimensional space. Thus, we can directly test if the pixel belongs to the given distribution.

The principled approach to test if a pixel belongs to a cluster – which utilizes property B.19 and the discussion from section B.3 in Appendix B – is as follows: Since the covariance matrix is symmetric positive definite[1], we can factorize it as $\Sigma = QSQ^T$, where Q is an orthogonal matrix and S is a diagonal matrix with positive diag-

[1]If Σ is only positive semidefinite, we add the term $\sigma_c^2 I$ to make it positive definite, where σ_c is the measurement error standard deviation and I is the 3×3 identity matrix; this addition can be justified as incorporating measurement error into the samples.

onal entries. Suppose X is a multivariate Gaussian random vector with mean μ and covariance matrix Σ; the statistic $S^{-1/2}Q^T(X-\mu)$ is distributed according to the standard multivariate normal with mean zero and covariance matrix equal to the identity (see property B.19). Thus, for image pixel I, we calculate the statistic $||S^{-1/2}Q^T(I-\mu)||$, which is its distance from the origin after normalizing it to the standard 3-dimensional multivariate normal. We then threshold on this statistic: If it exceeds a certain threshold, we believe it does not belong to the cluster, otherwise we claim it does belong to the cluster. This threshold depends on the stringency of the test; possible values include a threshold of 2.8 (a sphere of radius 2.8 contains about 95% of the probability of the standard three-dimensional multivariate normal) or 2.0 (a sphere of radius 2.0 contains slightly under 75% of the probability of the standard three-dimensional multivariate normal).

In practice, however, an equally effective – and slightly simpler and more efficient – technique thresholds on a different statistic: $(I-\mu)^T\Sigma^{-1}(I-\mu)$, where I is the observed pixel and μ and Σ are the mean and covariance matrix of the cluster. An intuitive justification of this statistic is that it represents the squared distance of the pixel from the mean of the cluster, normalized by the cluster covariance, and is thus the multi-dimensional analogue of $\left(\frac{X-\mu'}{\sigma'}\right)^2$, where X is a one-dimensional value which we test for belonging to a univariate normal distribution with mean μ' and standard deviation σ'. Since it is well known that one common test (see section B.3) for the univariate normal distribution will not reject that X belongs to the normal distribution with mean μ' and standard deviation σ' if the statistic $\left|\frac{X-\mu'}{\sigma'}\right|$ does not exceed 1.96, a similar test here would not reject that I belongs to the cluster with mean μ and covariance matrix Σ if the statistic $(I-\mu)^T\Sigma^{-1}(I-\mu)$ does not exceed $1.96^2 \approx 3.8$. Our implementation of augmenting Bayesian Matting to use the approach described in this chapter – for purposes of comparison in chapter 6 – will use this technique.

It is possible to use one of the optimizations described in section 4.3.9 and classify

a pixel as both foreground and background when a foreground (background) cluster that the pixel belongs to overlaps with a cluster of a different type.

There are two advantages when this approach is applied to Bayesian Matting: Efficiency and Accuracy with respect to Sparseness.

In section 3.2.1, we saw that the main speed bottleneck in Bayesian Matting was the numerical optimization needed to solve for F, B and α. However, in this approach, if we detect that an image pixel belongs to exactly one of the foreground or background, we solve for the unknown parameters immediately (for example, if we detect foreground, we set $\alpha = 1$, F to the image pixel and B is inconsequential). Thus, we can skip the numerical optimization procedure for all pixels detected as exactly one of foreground or background; these pixels usually comprise a large percentage of the unknown region and hence efficiency is greatly improved.

In section 3.2.2, we saw that one important accuracy problem with Bayesian Matting was that it did not generate sparse α – where the majority of α is 0 or 1 – even when the foreground/background boundary was sharp. By forcing $\alpha = 0$ when the pixel is definitely background and $\alpha = 1$ when the pixel is definitely foreground, we enforce the desirable sparseness property and improve the sharpness and accuracy of the matte.

5.5 Application to Segment-Based Matting

The approach introduced here is easily applied to the Segment-Based Matting approach described in chapter 4: Since we already detect if a pixel belongs to the foreground and/or the background (see section 4.3.2), we can simply inpaint or apply texture synthesis in the regions which we believe are both the foreground and the background.

The main advantage of applying this approach to Segment-Based Matting is that accuracy is improved when the foreground and background distributions overlap and the calculated alpha value is ambiguous.

Chapter 6

Results and Discussion

In this chapter, we will present the results from running Segment-Based Matting from chapter 4 and our extension presented in chapter 5 – using inpainting to resolve ambiguities in generated alpha mattes – applied to both Bayesian Matting and Segment-Based Matting. We will briefly discuss our results as we present them.

Our techniques are run on a variety of images, for which scaled-down thumbnails are presented in figure 6-1 together with their associated trimaps. We compare our results with the results obtained – by running with the same images and trimaps – on both our modified Bayesian Matting algorithm described in section 3.3 and Levin et al's closed form laplacian matting algorithm. Although Levin et al's algorithm is scribble-based, we can simply consider the trimap as being two very dense scribbles, one for the foreground and one for the background. Thus, we standardize the inputs across all the algorithms.

For each image, we measure the performance of the algorithms in two ways: Efficiency and Accuracy. To measure efficiency, we use the run-times of each algorithm. To study accuracy, we examine the matte in closer detail, as well as use it to composite the extracted foreground element onto a blue background. As noted in [2], compositing on a blue background[1] is preferable for studying accuracy because arti-

[1]Depending on the foreground color, compositing on a checkerboard of colors can also be very good.

facts in the matte solution can be more closely examined, as compared to compositing it on a complex background.

For the basis of comparison, our Segment-Based Matting implementation will utilize the line segment model in the coplanar case (see section 4.3.6). The differences between this and the full line model is generally negligible, and the comparisons presented in this chapter will still hold valid. For the extension described in chapter 5, we use inpainting to fill in ambiguous regions and an MRF to smooth the areas around the ambiguous regions; the solutions from using texture synthesis are generally quite similar.

6.1 Efficiency and Runtimes

Table 6.1 shows the running times of each algorithm on each of our test images, using as a reference point our Bayesian Matting implementation. The columns Bayesian++ and Segment-Based++ indicate the running times for the Bayesian Matting and Segment-Based Matting algorithms extended by using inpainting to resolve ambiguous regions as described in chapter 5.

We see that Levin et al's algorithm is much faster than all the other approaches – although this is an unfair comparison, as all programs were written in MATLAB where Levin et al's algorithm is extremely fast and the others are relatively slow due to a large number of array accesses. More importantly, we observe a few things:

1. The Bayesian++ algorithm is significantly faster than the Bayesian Matting algorithm. This implies that a large percentage of pixels can easily be classified directly as foreground or background – using the tests described in section 5.4 – and there is no need to run the slow Bayesian Matting iterative optimization procedure on these pixels.

2. The Segment-Based and Segment-Based++ algorithms are faster than even the

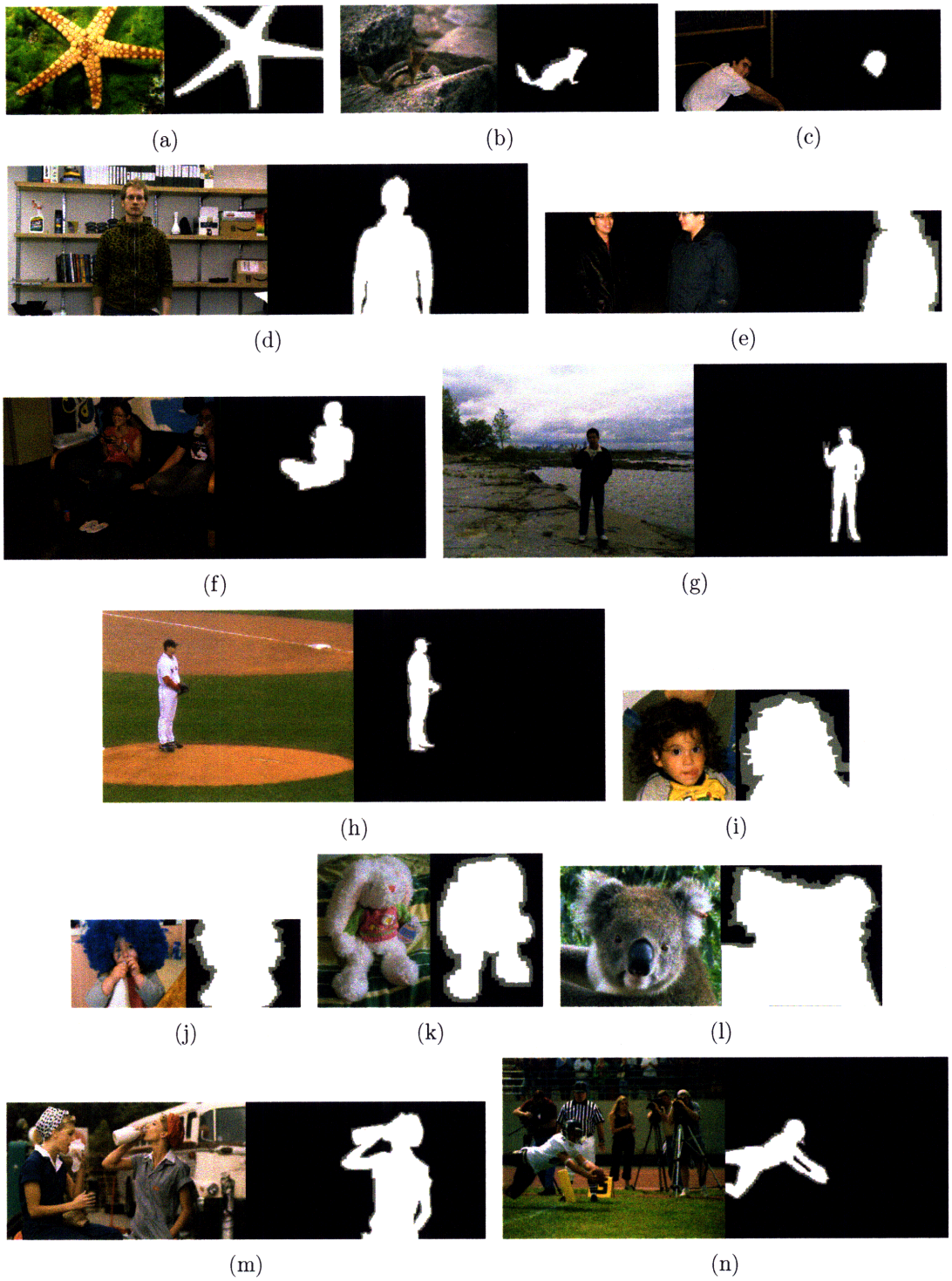


Figure 6-1: Test Images for Matting and Associated Trimaps. Images 6-1(a) and 6-1(b) are from the Berkeley Segmentation Dataset [19], while images 6-1(i), 6-1(j), 6-1(k) and 6-1(l) are from [16].

Table 6.1: Relative Running Times of Different Algorithms

Image	Bayesian	Levin et al	Bayesian++	Segment-Based	Segment-Based++
(a)	100%	1.5%	41.7%	25.6%	26.0%
(b)	100%	4.2%	37.4%	32.0%	26.0%
(c)	100%	7.9%	41.7%	17.9%	21.2%
(d)	100%	5.3%	26.3%	23.2%	23.8%
(e)	100%	2.4%	16.8%	12.9%	13.5%
(f)	100%	5.2%	21.0%	14.6%	15.8%
(g)	100%	6.1%	30.8%	19.5%	18.9%
(h)	100%	7.6%	43.0%	24.6%	23.2%
(i)	100%	1.3%	40.5%	22.9%	19.6%
(j)	100%	1.3%	36.4%	18.0%	18.4%
(k)	100%	1.3%	30.2%	22.5%	22.8%
(l)	100%	1.3%	45.4%	41.3%	41.3%
(m)	100%	4.1%	34.0%	25.2%	20.7%
(n)	100%	7.8%	45.5%	29.2%	26.0%

Bayesian++ algorithm. This shows that – as expected – a closed form solution is much faster than numerical optimization.

We wish to elaborate slightly on the second point. Although from table 6.1 the time differences between Bayesian++ and the Segment-Based and Segment-Based++ approaches may not appear large, this is because all these approaches spend a relatively large amount of time in sampling local pixels and clustering them. Although we had noted in table 3.1 that in Bayesian Matting this time was relatively insignificant, in more efficient approaches this time can become significant. Table 6.2 shows the estimation time of each algorithm, after sampling and clustering, using the Bayesian Matting time as the reference point. We see that in fact the processing time spent in estimation in the Segment-Based and Segment-Based++ Algorithms is much less than in Bayesian++, which itself is a significant improvement over Bayesian Matting.

We have thus demonstrated that classifying a large number of pixels directly as foreground or background can greatly improve the efficiency of Bayesian Matting, and further improvements in efficiency can be obtained by using Segment-Based Matting, which has a closed form solution and thus improves the estimation time.

Table 6.2: Relative *Estimation* Running Times of Different Algorithms, *after sampling and clustering.*

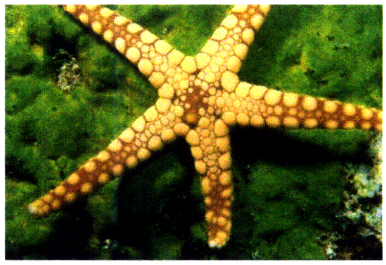
Image	Bayesian	Bayesian++	Segment-Based	Segment-Based++
(a)	100%	34.4%	16.4%	16.9%
(b)	100%	29.7%	23.6%	16.8%
(c)	100%	34.5%	7.8%	11.5%
(d)	100%	17.2%	13.7%	14.4%
(e)	100%	6.5%	2.1%	2.8%
(f)	100%	11.2%	4.0%	5.3%
(g)	100%	22.2%	9.6%	8.9%
(h)	100%	35.9%	15.3%	13.7%
(i)	100%	33.2%	13.3%	9.7%
(j)	100%	28.5%	7.9%	8.3%
(k)	100%	21.6%	13.0%	13.3%
(l)	100%	38.7%	34.1%	34.0%
(m)	100%	27.9%	17.6%	12.1%
(n)	100%	38.7%	20.4%	16.8%

Remark 6.1. Since the sampling time is significant in the Bayesian++, Segment-based and Segment-based++ algorithms, it would be worthwhile to use more efficient heuristics and data structures to accelerate sampling and clustering.

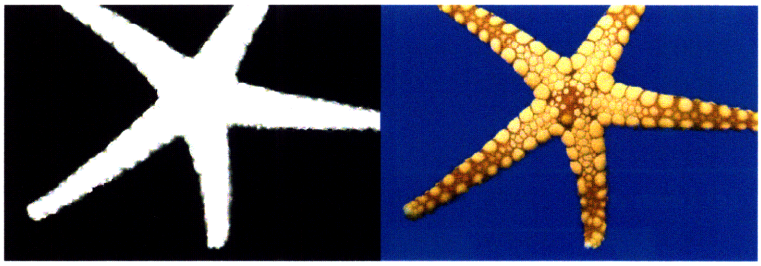
6.2 Accuracy I: Sharpness of Matte

We now briefly examine the accuracy of the generated matte in the context of its sharpness. We look at image 6-1(a), which has a clearly defined foreground/background boundary and thus should have a sharp alpha matte.

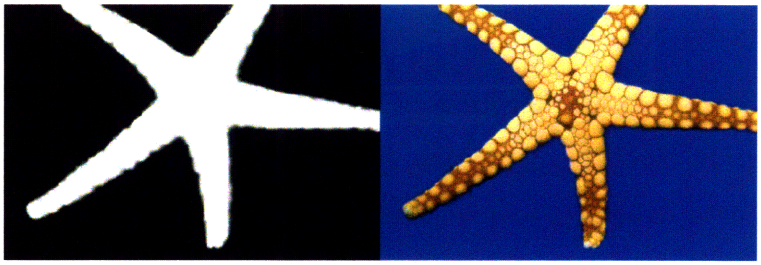
Figures 6-2 shows the results of running the different algorithms on this image, and figures 6-3 and 6-4 show some close ups of the alpha matte and the result of compositing on a blue background. Since the foreground/background boundary is sharp, the ambiguous areas where the foreground and background color distributions overlap are negligible; thus using inpainting to resolve ambiguous regions in Segment-Based matting is essentially identical to simply using Segment-Based Matting.



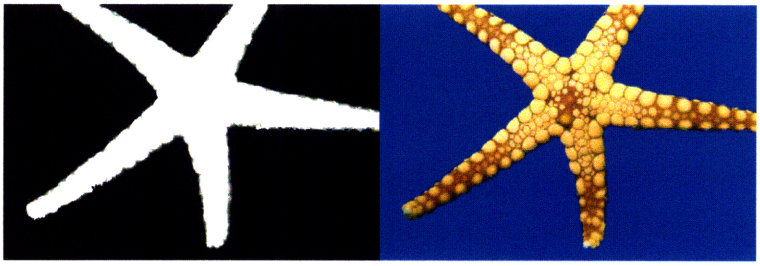
(a) Original Image



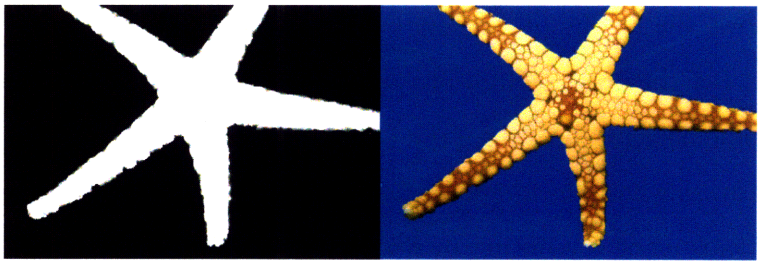
(b) Bayesian



(c) Levin et al

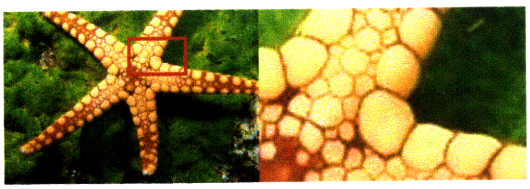


(d) Bayesian with inpainting

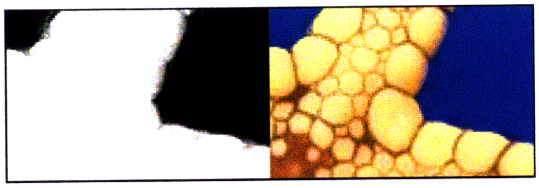


(e) Segment-Based Matting

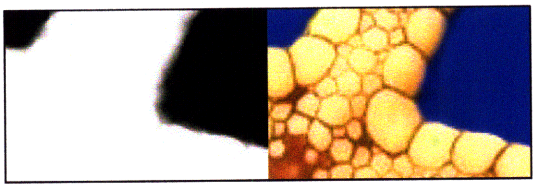
Figure 6-2: Results of running the different algorithms on image 6-1(a), as well as compositing the results on a blue background.



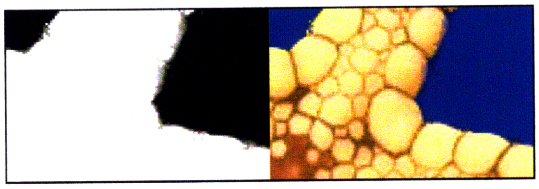
(a) Original Image



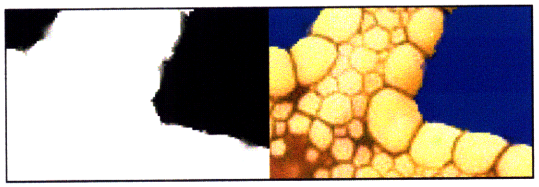
(b) Bayesian



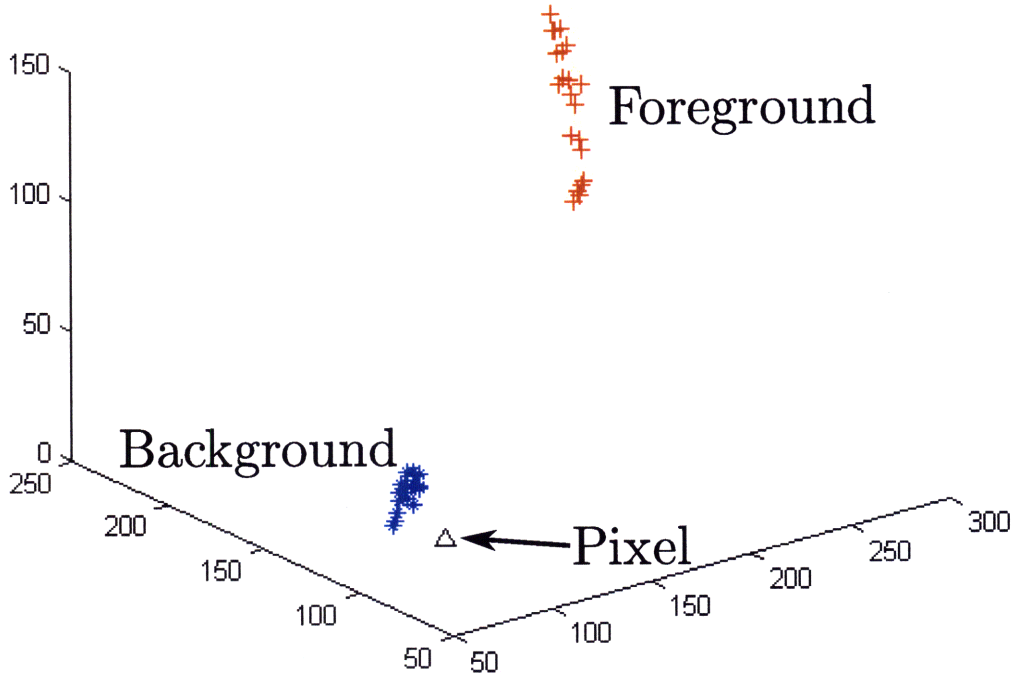
(c) Levin et al



(d) Bayesian with inpainting

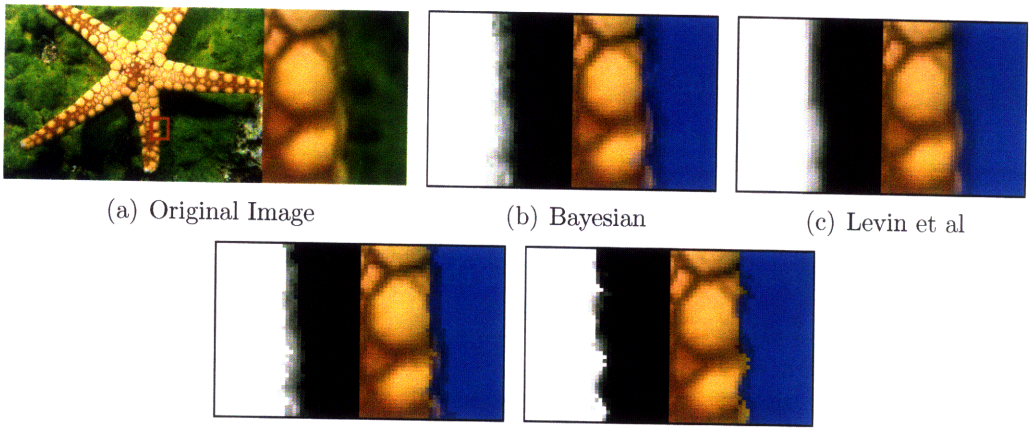


(e) Segment-Based Matting



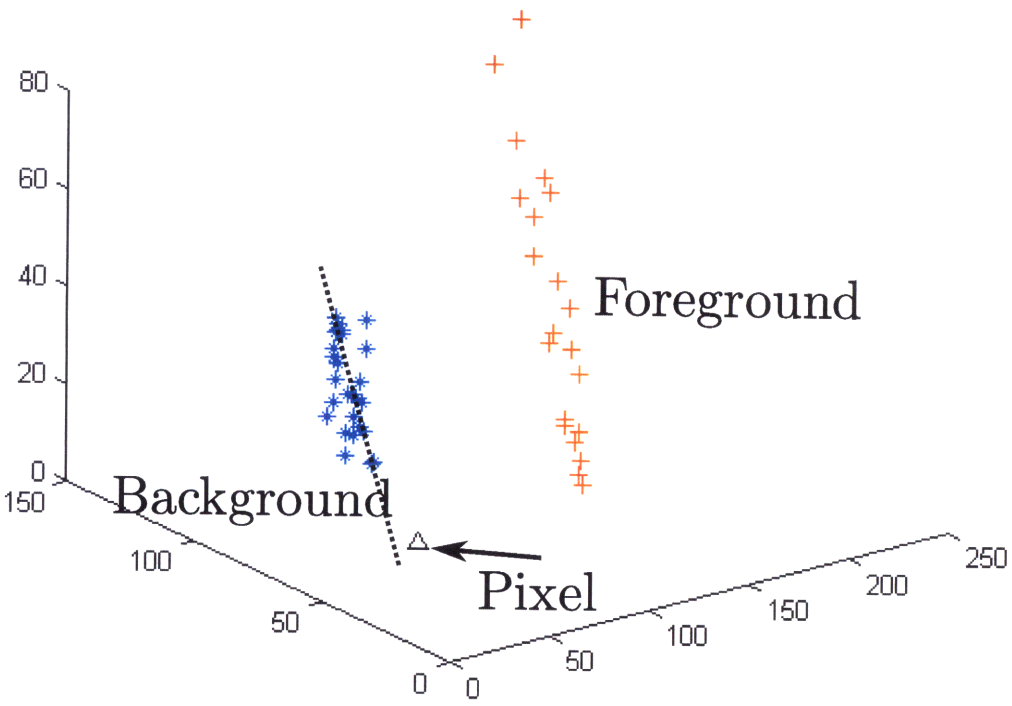
(f) Plot of Neighborhood around one problem pixel

Figure 6-3: A close up view of some of the results in figure 6-2.



(a) Original Image (b) Bayesian (c) Levin et al

(d) Bayesian with inpainting (e) Segment-Based Matting



(f) Plot of Neighborhood around one problem pixel

Figure 6-4: More close up views of some of the results in figure 6-2. The dotted line indicates an approximate shading line fitted to the background distribution.

From the figures, we observe the following:

1. The result by Levin et al is too smooth; even when the boundary is sharp, it tries to enforce smoothness and thus the matte is inaccurate.

2. The result from Bayesian Matting is also not sharp; this is probably due to its bias towards means as described in section 3.2.2.

3. Using inpainting to resolve ambiguous regions in Bayesian Matting improves the sharpness greatly; this is not due to the inpainting itself, but rather to the fact that we test for foreground and background pixels and assign $\alpha = 1$ or $\alpha = 0$ directly to them. However, the matte is still far from perfect, because there remain areas for which α is not directly assigned but are estimated by Bayesian Matting, and the alpha matte in these areas is not sharp.

4. Segment-Based Matting gives the best result.

The result of Levin et al demonstrates why enforcing smoothness in the alpha matte is not always the best choice; in fact we would argue – as we did in chapter 5 – that smoothness should only be enforced in areas where there may be problems in using color to estimate the alpha matte, in order to ensure that the matte is as accurate as possible. Another example of this can be found in figure 6-9, which is a close up view of figure 6-5: The foreground/background boundary is sharp but the alpha matte generated by Levin et al is not (there are no regions of ambiguity in that area and thus the results of Segment-Based Matting with or without inpainting are identical); in addition, adding inpainting to Bayesian Matting greatly improves the sharpness of the matte for the same reasons as in figures 6-3 and 6-4.

To understand why Segment-Based matting performs well, we look at the plots in figures 6-3(f) and 6-4(f), which plot the foreground and background color distributions in RGB space around a particular pixel. In figure 6-3(f), the image pixel is detected as background, not directly because of the foreground and background tests described in 4.3.2, but because of one of the optimizations described in section 4.3.9, where

we classify pixels lying beyond the foreground (background) clusters as foreground (background). See also figure 4-9.

In figure 6-4(f), the image pixel is given a very low value of α. This is because Segment-Based matting models shading; as can be seen from the dotted line in the figure, the image pixel lies on a short extension to the background shade and our shading model captures this effect.

In summary, we have shown that the Segment-Based Matting approach can generate sharper and more accurate mattes, due to its modelling of shading and its checking if an image pixel belongs to the foreground or the background before estimation.

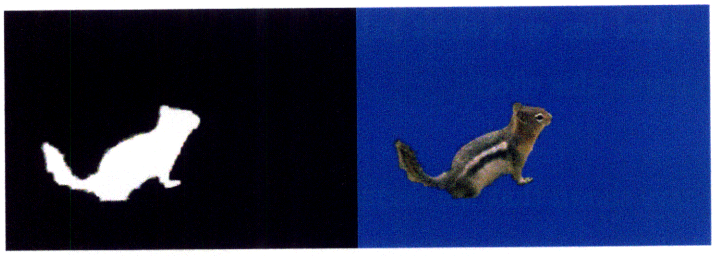
6.3 Accuracy II: Continuity and Regions of Ambiguity

In section 3.2.3, we briefly discussed the continuity of the alpha matte generated by Bayesian Matting, and noted that this was discontinuous in areas where the foreground and background distributions were similar. In chapter 5, we proposed an approach where we would detect these regions of ambiguity and use inpainting – enforcing some form of contour continuity – to fill in these regions and resolve the ambiguity. In this section, we study the effect of this approach and discuss the issue of accuracy of the alpha matte with regards to its continuity.

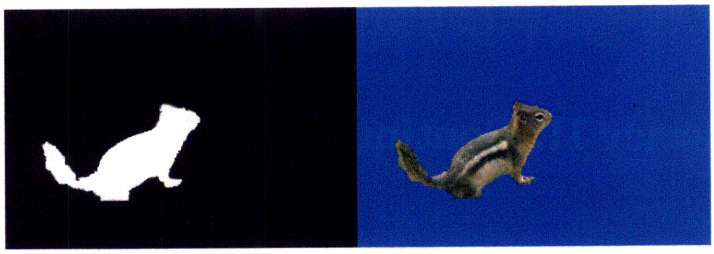
To this goal, figures 6-5 and 6-7 show the results of running the matting algorithms on the images in figures 6-1(b) and 6-1(c) respectively, and figures 6-6 and 6-8 show close-up views on some ambiguous areas – where the foreground and background color distributions are similar – from figures 6-5 and 6-7 respectively.

(a) Original Image

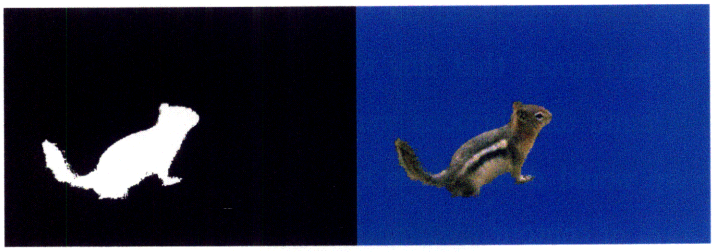
(b) Bayesian



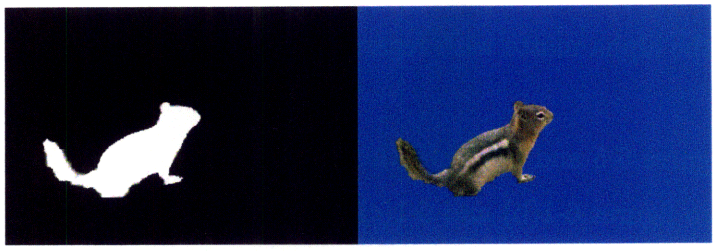
(c) Levin et al



(d) Bayesian with inpainting



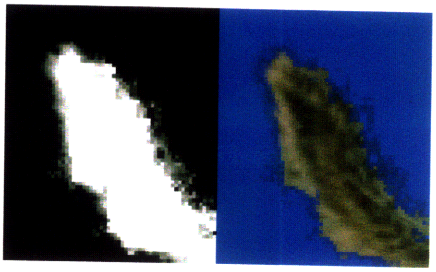
(e) Segment-Based Matting, without inpainting



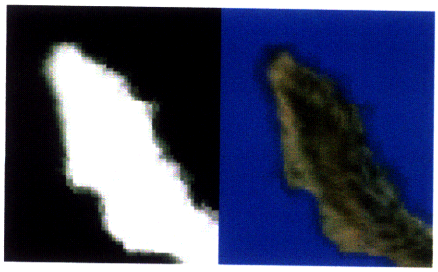
(f) Segment-Based Matting, with inpainting

Figure 6-5: Results of running the different algorithms on image 6-1(b), as well as compositing the results on a blue background.

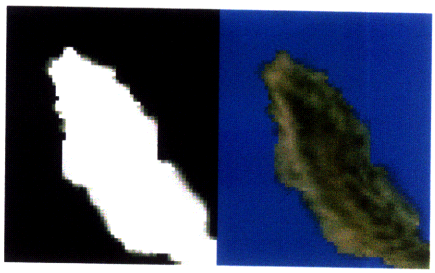
(a) Original Image



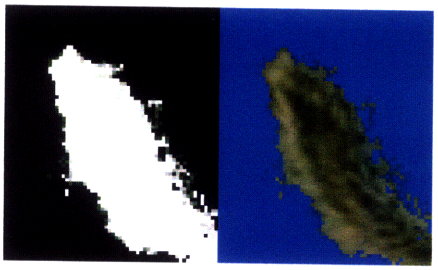
(b) Bayesian



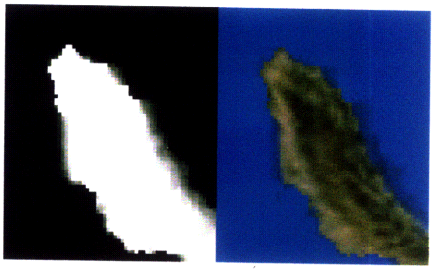
(c) Levin et al



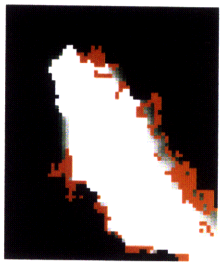
(d) Bayesian with inpainting



(e) Segment-Based Matting, without inpainting

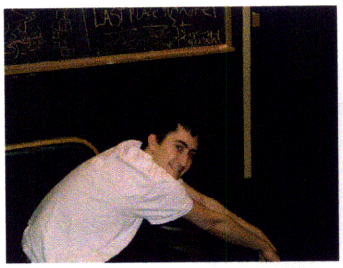


(f) Segment-Based Matting, with inpainting

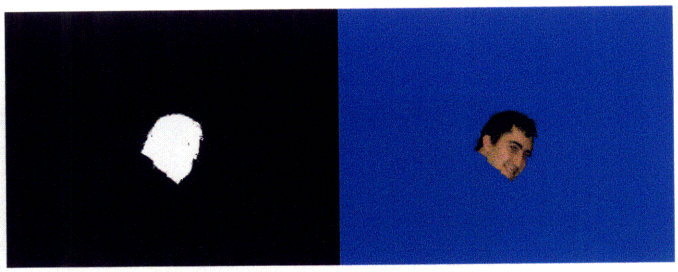


(g) Ambiguous regions marked in red

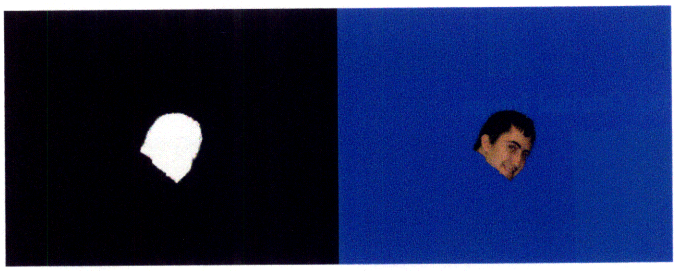
Figure 6-6: A close up view of some of the results in figure 6-5, together with the regions of ambiguity.



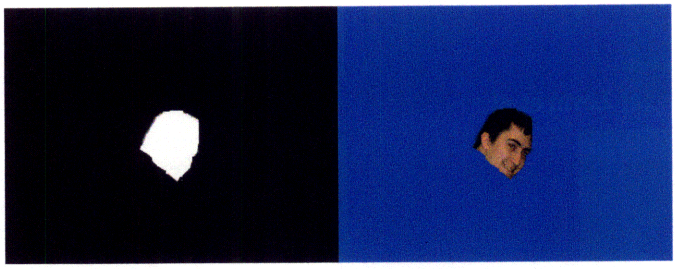
(a) Original Image



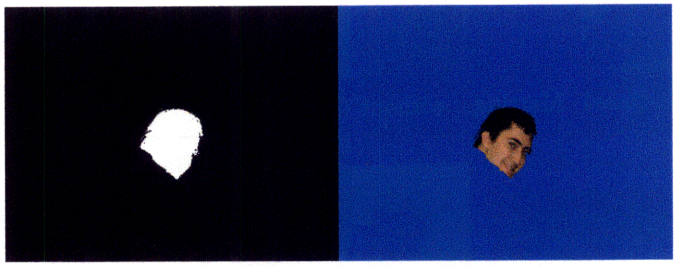
(b) Bayesian



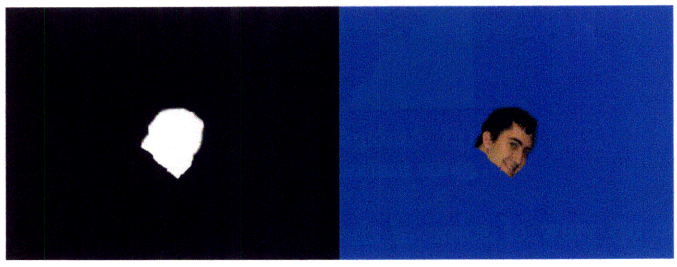
(c) Levin et al



(d) Bayesian with inpainting



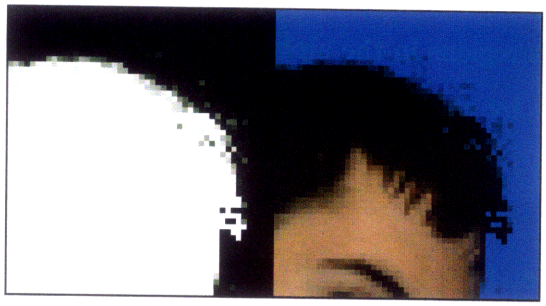
(e) Segment-Based Matting, without inpainting



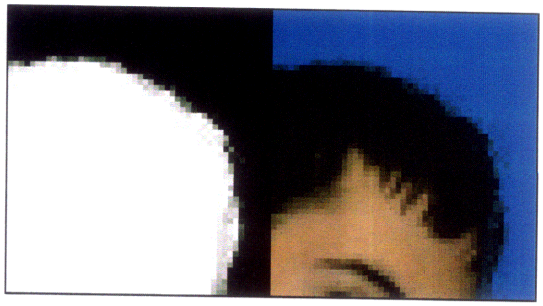
(f) Segment-Based Matting, with inpainting

Figure 6-7: Results of running the different algorithms on image 6-1(c), as well as compositing the results on a blue background.

(a) Original Image

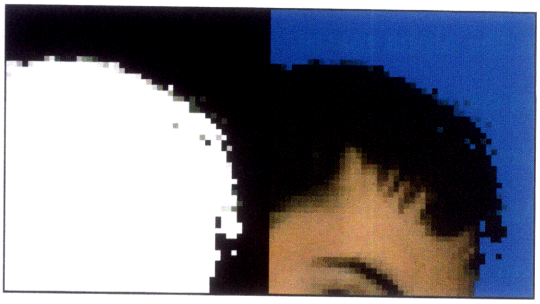


(b) Bayesian

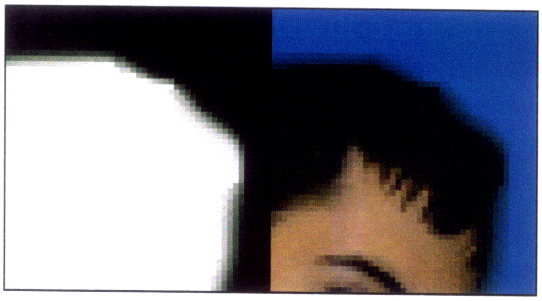


(c) Levin et al

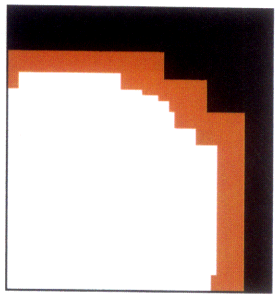
(d) Bayesian with inpainting



(e) Segment-Based Matting, without inpainting



(f) Segment-Based Matting, with inpainting



(g) Ambiguous regions marked in red

Figure 6-8: A close up view of some of the results in figure 6-7, together with the regions of ambiguity.

From the figures, we observe the following:

1. Both the Bayesian and Segment-Based approaches without inpainting have discontinuous regions.

2. Applying the inpainting extension improves both the Bayesian and Segment-Based approaches greatly.

3. Levin et al performs adequately – but probably not as well as Segment-based Matting with Inpainting.

Thus, using inpainting to resolve ambiguous regions in the alpha matte can improve the accuracy and appearance of the matte by enforcing continuity in areas where color estimation can fail.

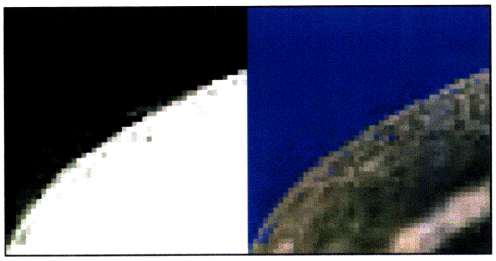
The reason why Levin et al performs adequately, even in areas where the foreground and background color distributions are similar, is that it already enforces continuity in the alpha matte. However, enforcing continuity all the time, even in areas where we can clearly estimate α from the color distribution, can lead to problems. We have already seen in the previous section that it can lead to mattes that are not sharp. Another problem is when the correct alpha matte has holes in it; enforcing continuity can lead to these holes being covered up in the estimated matte.

Two examples of this can be found in figures 6-10 and 6-12 and their associated close-up views in figures 6-11 and 6-13; these are the results of running the algorithms on images 6-1(d) and 6-1(n) respectively. In figure 6-11, the solution by Levin et al captures the books and the shelf behind the person as part of the person, even when they are not; in figure 6-13, the solution by Levin et al captures the white background – part of the referee's outfit – as part of the footballer's helmet, even though this is incorrect.

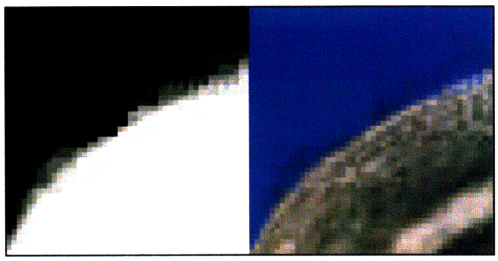
These examples show the danger of trying to enforce continuity in all cases regardless of the local foreground and background color distributions. Therefore, we believe

that our approach – use color to estimate, and only try to enforce continuity if the color distributions are ambiguous – is more precise.

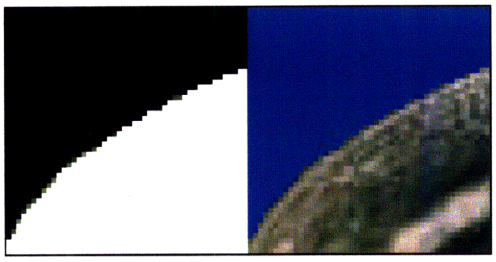
(a) Original Image



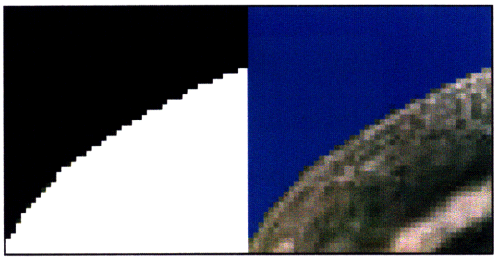
(b) Bayesian



(c) Levin et al

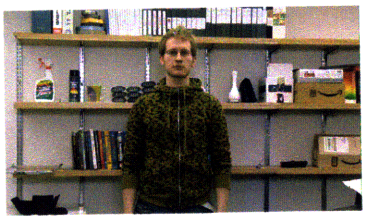


(d) Bayesian with inpainting

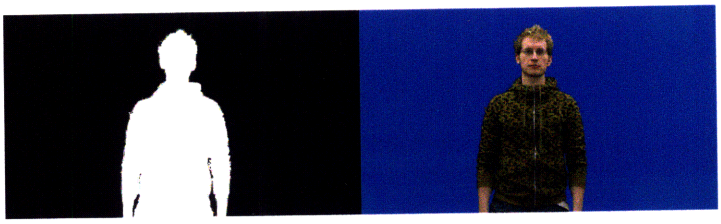


(e) Segment-Based Matting

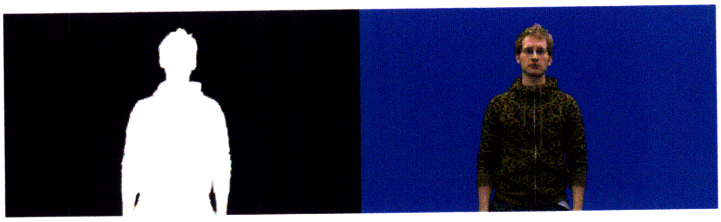
Figure 6-9: More close up views of some of the results in figure 6-5.



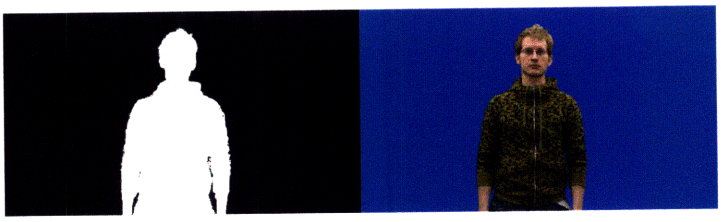
(a) Original Image



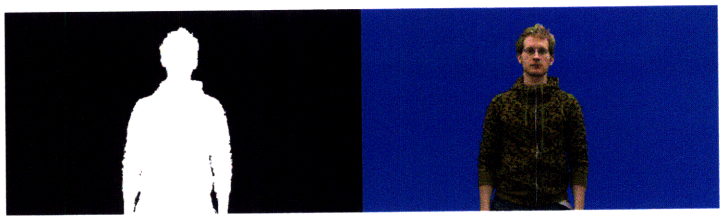
(b) Bayesian



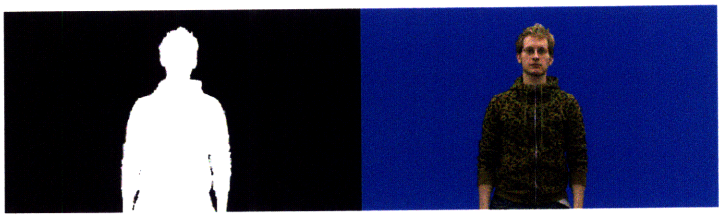
(c) Levin et al



(d) Bayesian with inpainting

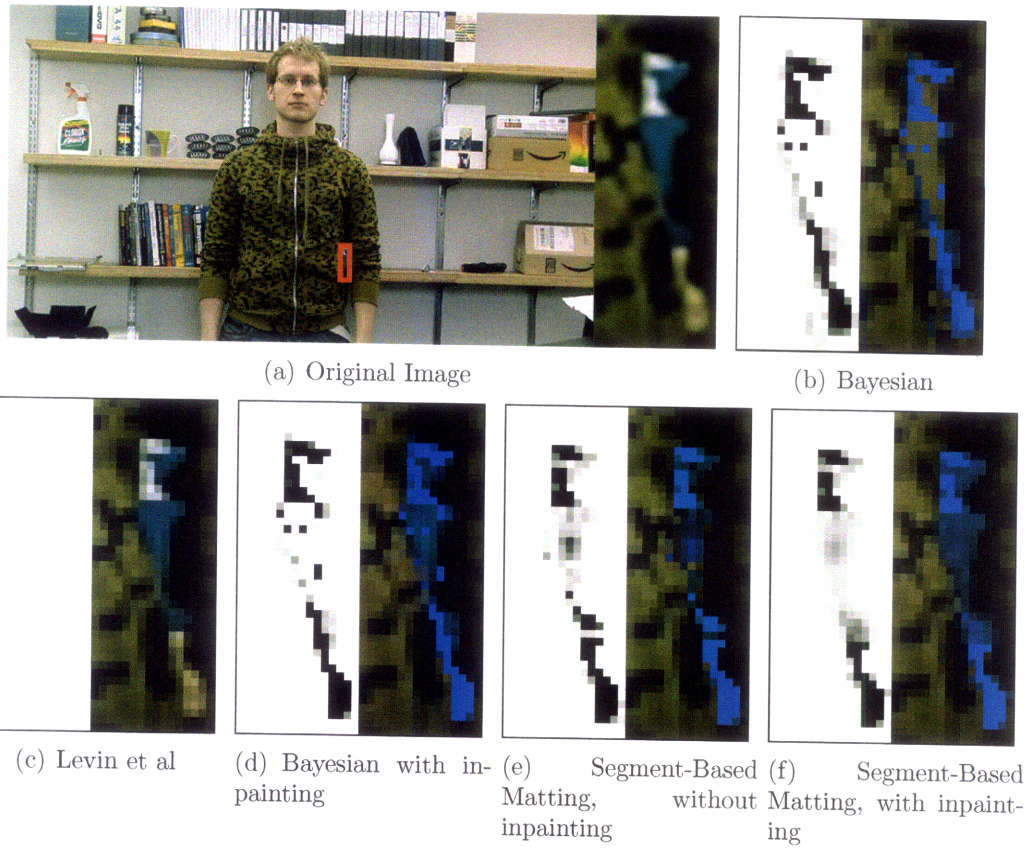


(e) Segment-Based Matting, without inpainting



(f) Segment-Based Matting, with inpainting

Figure 6-10: Results of running the different algorithms on image 6-1(d), as well as compositing the results on a blue background.



(a) Original Image

(b) Bayesian

(c) Levin et al

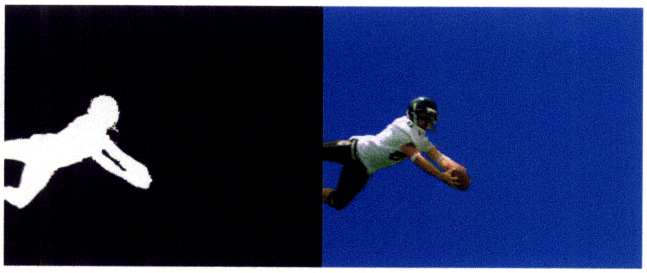
(d) Bayesian with inpainting

(e) Segment-Based Matting, without inpainting

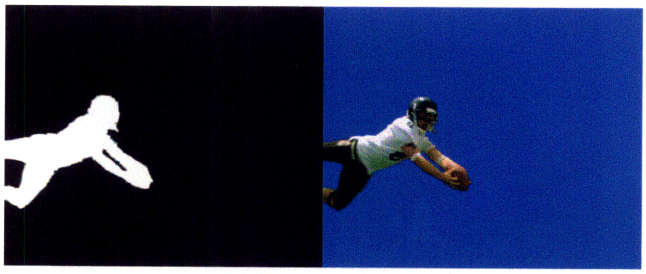
(f) Segment-Based Matting, with inpainting

Figure 6-11: A close up view of some of the results in figure 6-10.

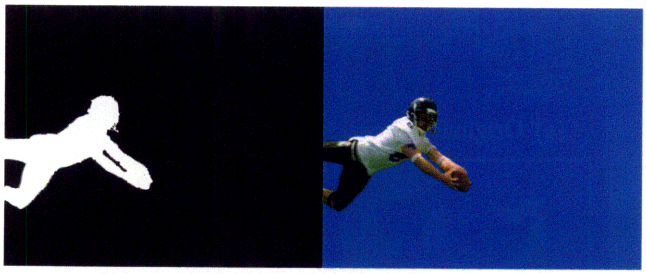
(a) Original Image



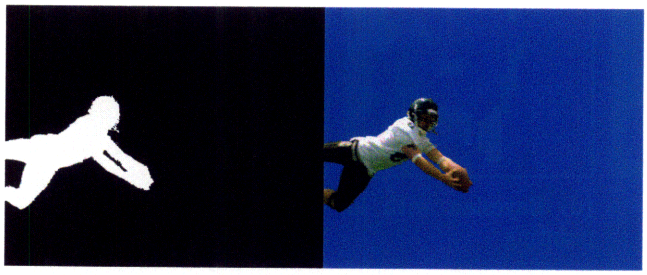
(b) Bayesian



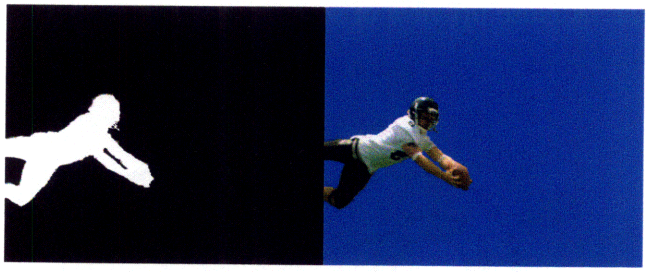
(c) Levin et al



(d) Bayesian with inpainting



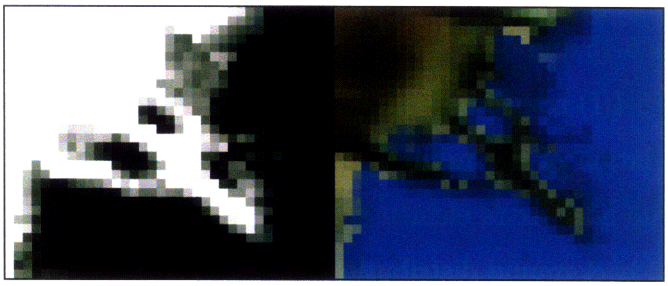
(e) Segment-Based Matting, without inpainting



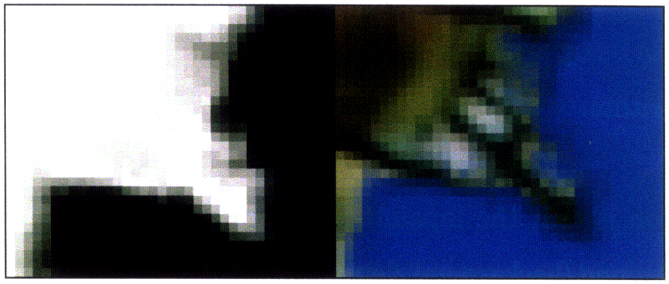
(f) Segment-Based Matting, with inpainting

Figure 6-12: Results of running the different algorithms on image 6-1(n), as well as compositing the results on a blue background.

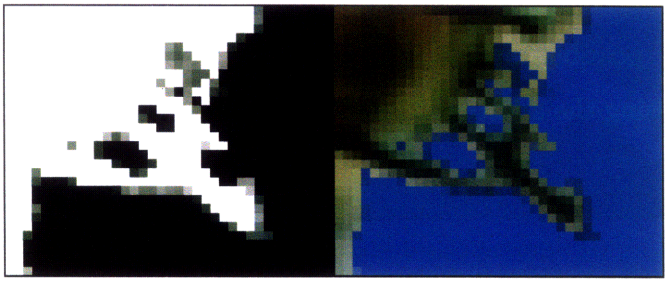
(a) Original Image



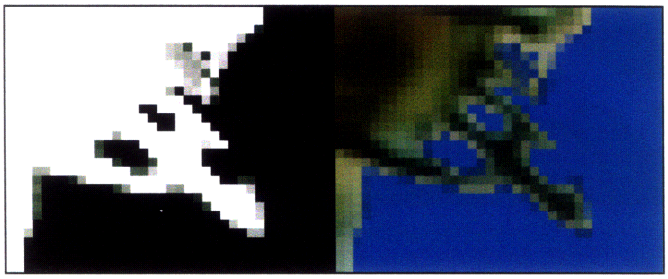
(b) Bayesian



(c) Levin et al



(d) Bayesian with inpainting



(e) Segment-Based Matting

Figure 6-13: A close up view of some of the results in figure 6-12.

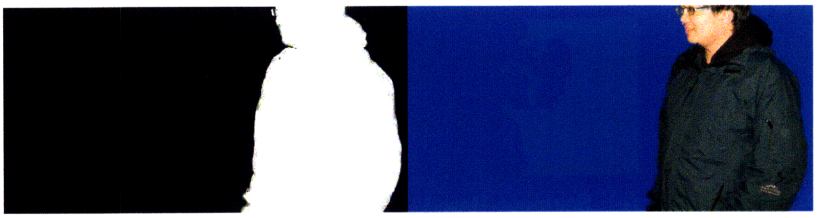
6.4 Some Problem Cases

Unfortunately, there are also some cases where our inpainting extension produces sub-optimal results. These sub-optimal cases happen when the inpainting algorithm fails to obtain an accurate matte in the ambiguous regions.

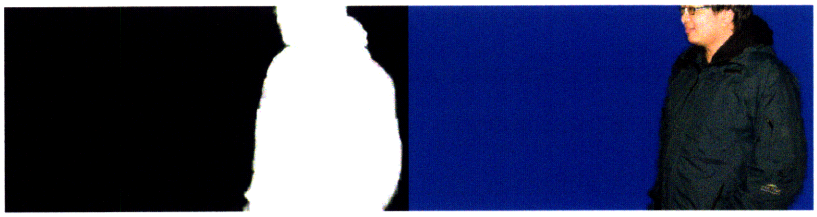
Two examples of this can be found in figures 6-14 and 6-16 and their associated close-up views in figures 6-15 and 6-17; these are the results of running the algorithms on images 6-1(e) and 6-1(f) respectively. In both cases, adding the inpainting extension results in a degradation of the result of the Segment-Based approach, because the inpainting algorithm adds extra elements to the foreground. We have plotted the local foreground and background distributions around a typical pixel from the regions marked as ambiguous in figures 6-15(h) and 6-17(h); we see that these pixels are marked correctly as being ambiguous, and hence the problem lies with the inpainting aspect of the algorithm.

One possible reason is that the inpainting algorithm is entirely independent of the matting and foreground/background detection algorithms, and only uses the generated alpha matte, not the original image and trimap; it may be useful for future research to study possible inpainting algorithms that can be adapted to the matting framework, using the additional information provided by the original image and trimap.

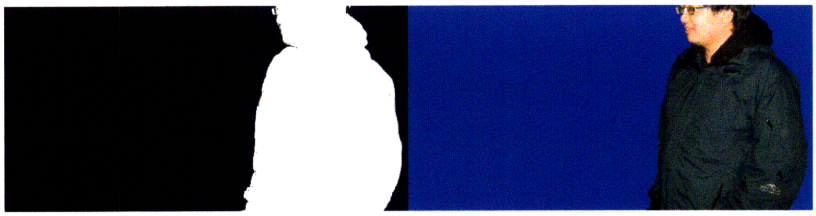
(a) Original Image



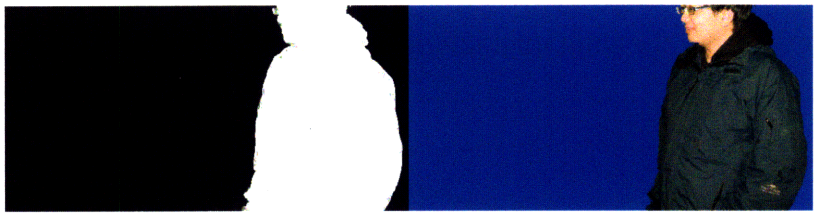
(b) Bayesian



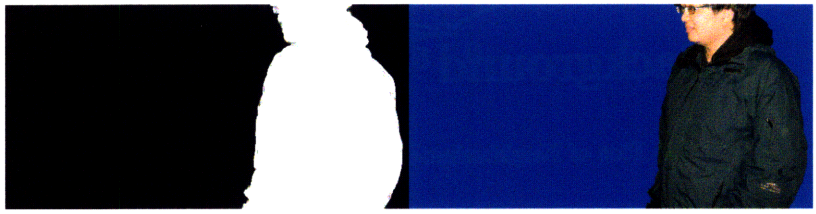
(c) Levin et al



(d) Bayesian with inpainting

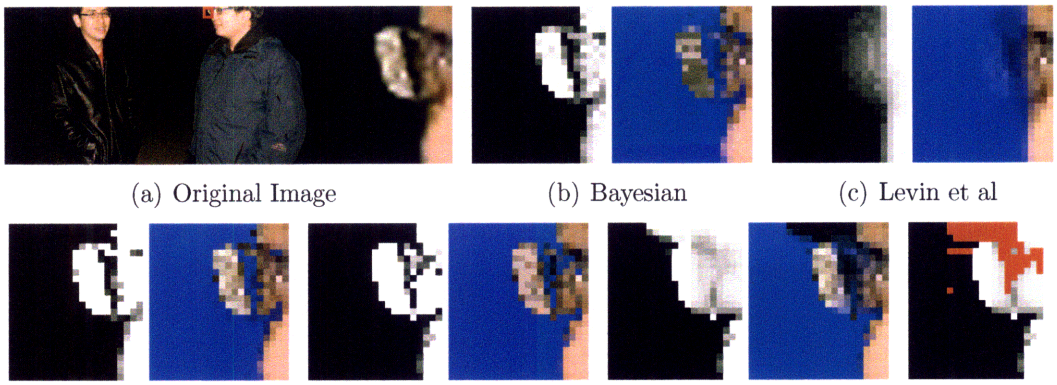


(e) Segment-Based Matting, without inpainting



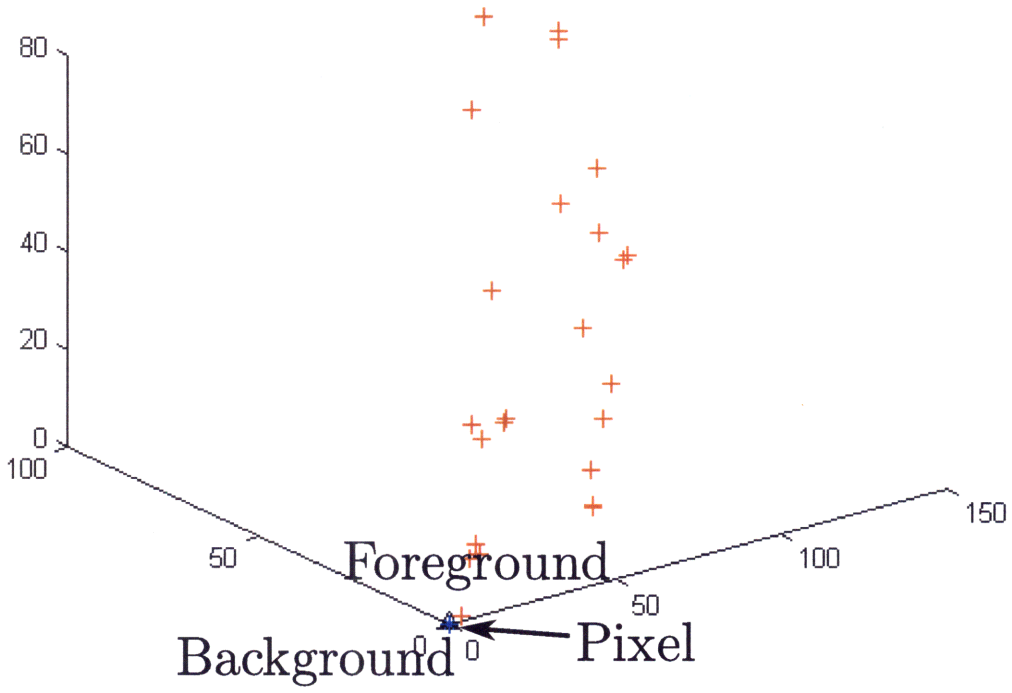
(f) Segment-Based Matting, with inpainting

Figure 6-14: Results of running the different algorithms on image 6-1(e), as well as compositing the results on a blue background.



(a) Original Image (b) Bayesian (c) Levin et al

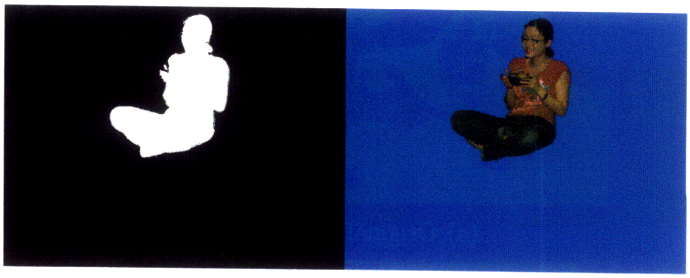
(d) Bayesian with inpainting (e) Segment-Based Matting, without inpainting (f) Segment-Based Matting, with inpainting (g) Ambiguous regions marked in red



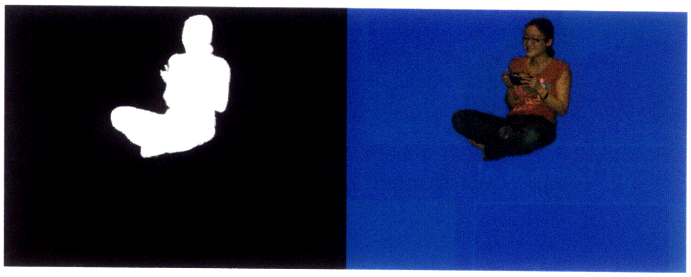
(h) Plot of Neighborhood around one problem pixel

Figure 6-15: A close up view of some of the results in figure 6-14, together with the regions of ambiguity.

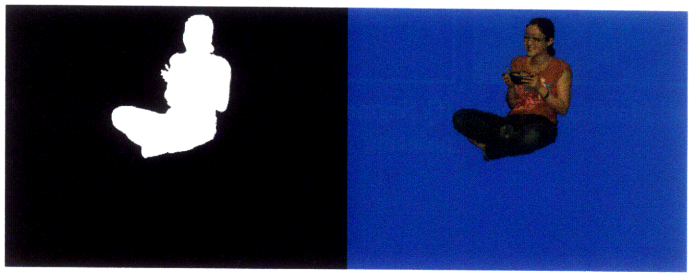
(a) Original Image



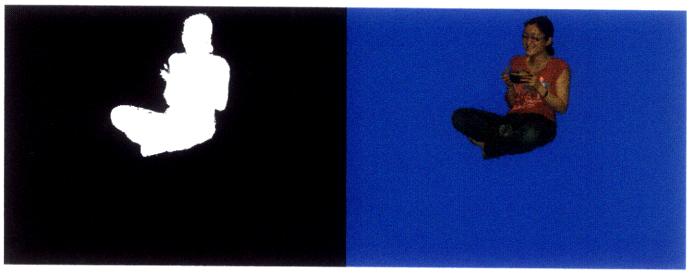
(b) Bayesian



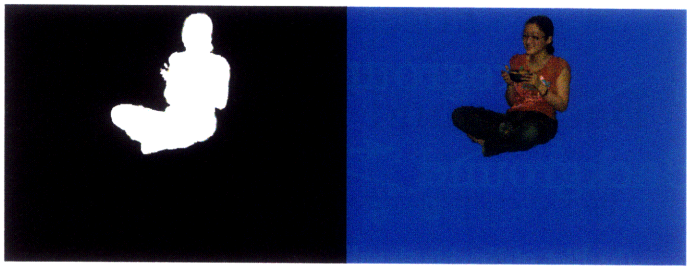
(c) Levin et al



(d) Bayesian with inpainting



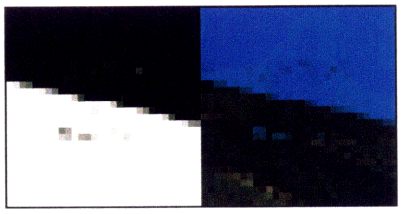
(e) Segment-Based Matting, without inpainting



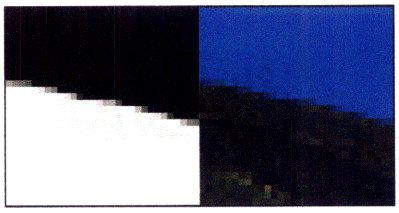
(f) Segment-Based Matting, with inpainting

Figure 6-16: Results of running the different algorithms on image 6-1(f), as well as compositing the results on a blue background.

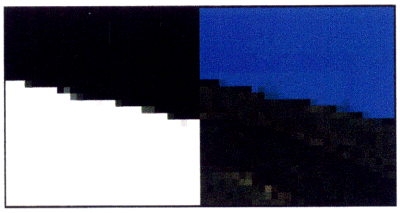
(a) Original Image



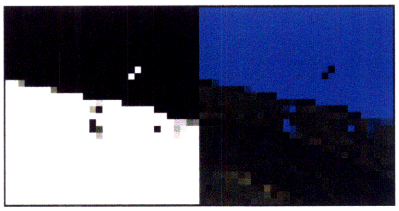
(b) Bayesian



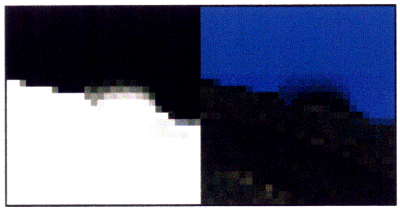
(c) Levin et al



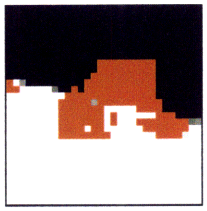
(d) Bayesian with inpainting



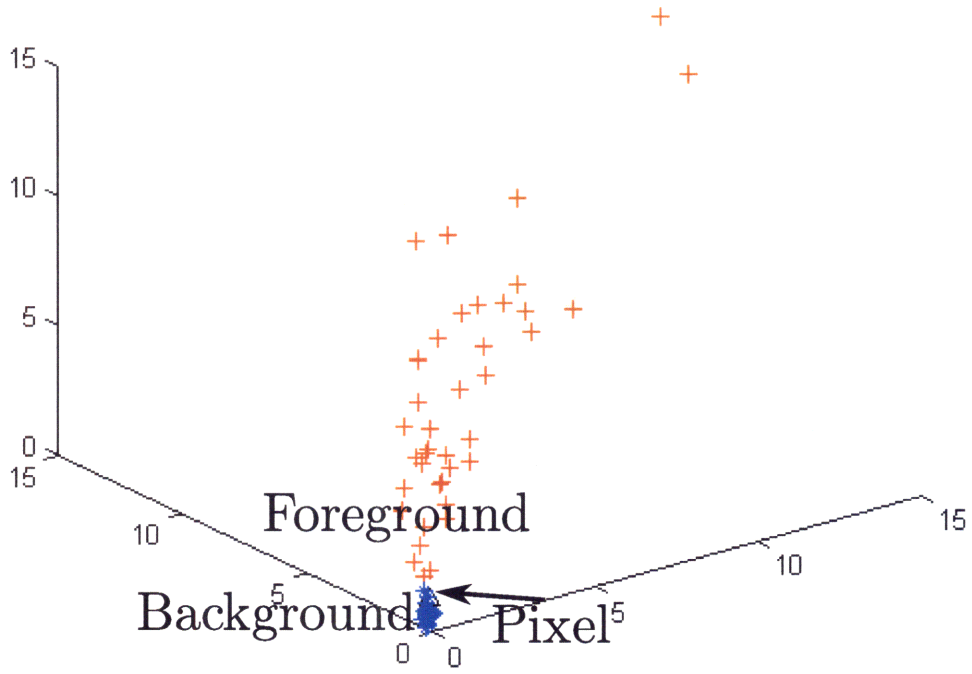
(e) Segment-Based Matting, without inpainting



(f) Segment-Based Matting, with inpainting



(g) Ambiguous regions marked in red



(h) Plot of Neighborhood around one problem pixel

Figure 6-17: A close up view of some of the results in figure 6-16, together with the regions of ambiguity.

6.5 Other Results

Figures 6-18 to 6-25 show the results of running the different algorithms on the remaining images in figure 6-1. In these results, the key observations we have made in the previous sections are generally true:

1. The Segment-Based Matting solution is generally sharp and accurate.

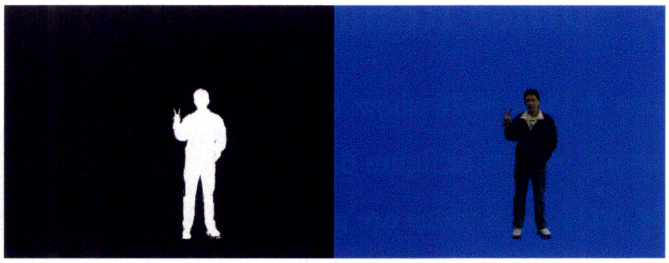
2. The solutions from Levin et al and from Bayesian Matting are usually not sharp; in the case of Levin et al, the matte is usually too smooth and there are no clear edges separating the foreground/background boundary.

3. Applying the inpainting extension makes the Bayesian Matting solution much sharper and more accurate, primarily because of the foreground and background detection mechanism that enforces sparseness of the alpha matte in areas which are "obviously" foreground or background.

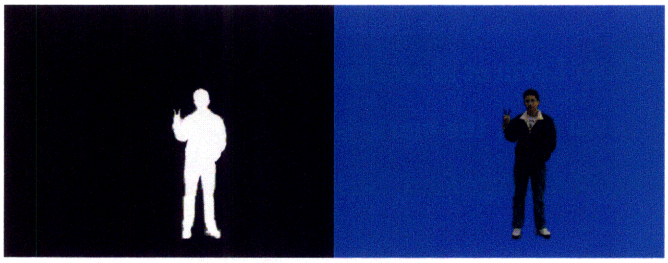
4. Applying the inpainting extension improves the accuracy of both the Bayesian and Segment-Based Matting algorithms in ambiguous regions where the foreground and background color distributions are fairly similar.

These observations can be seen in figure 6-19, which is a close up view of a difficult matting area from figure 6-18, which itself shows the results of running the matting algorithms on image 6-1(g); although none of the results are optimal, the best result is obtained from using Segment-Based Matting with inpainting. In particular, Levin et al's solution is too smooth and creates an artificial "box" around the shoe, while the Bayesian and Segment-Based Matting approaches without inpainting generate discontinuous mattes in the regions of color ambiguity.

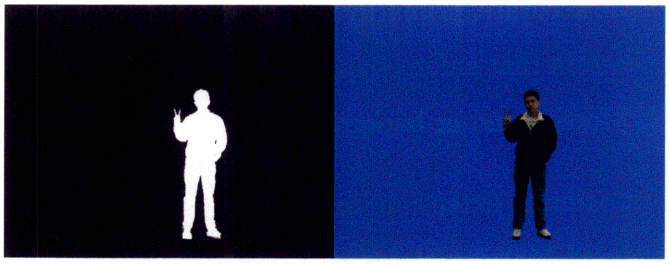
(a) Original Image



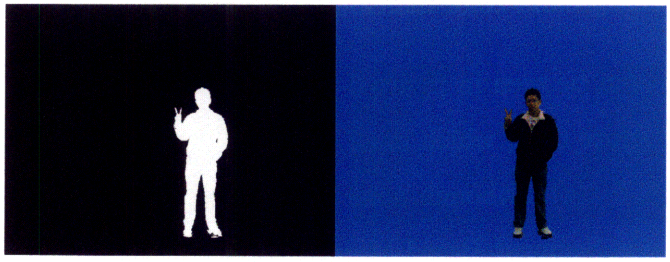
(b) Bayesian



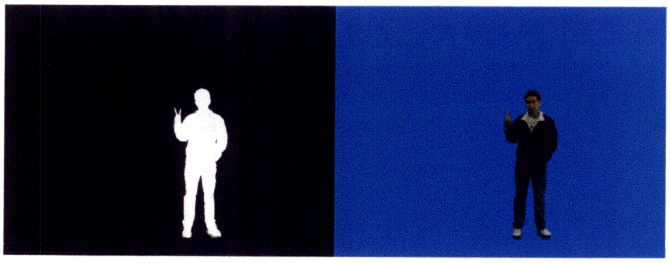
(c) Levin et al



(d) Bayesian with inpainting

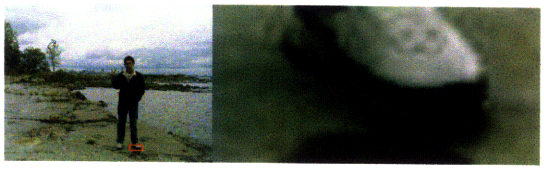


(e) Segment-Based Matting, without inpainting

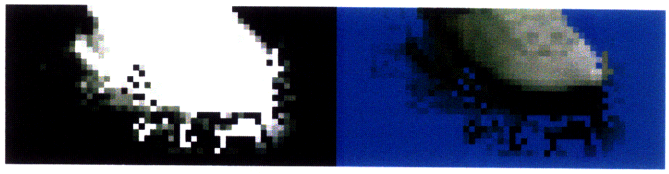


(f) Segment-Based Matting, with inpainting

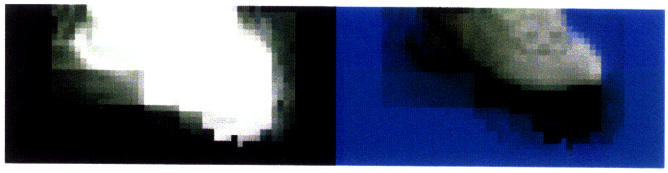
Figure 6-18: Results of running the different algorithms on image 6-1(g), as well as compositing the results on a blue background.



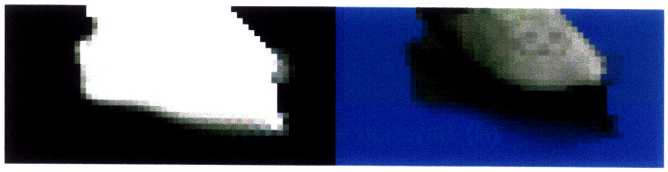
(a) Original Image



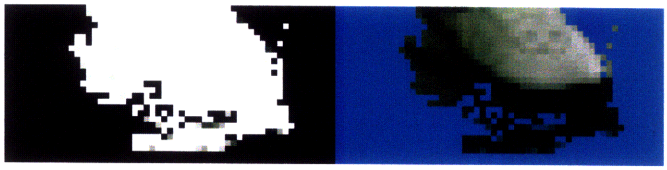
(b) Bayesian



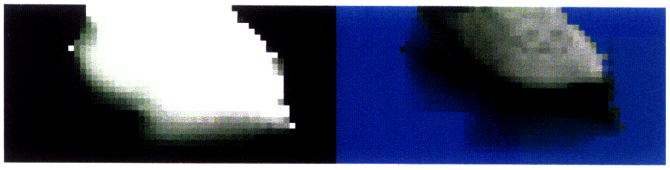
(c) Levin et al



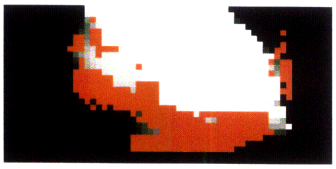
(d) Bayesian with inpainting



(e) Segment-Based Matting, without inpainting



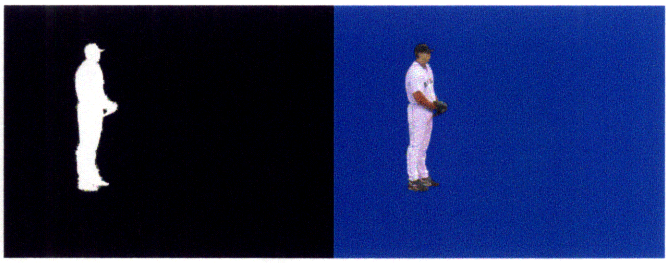
(f) Segment-Based Matting, with inpainting



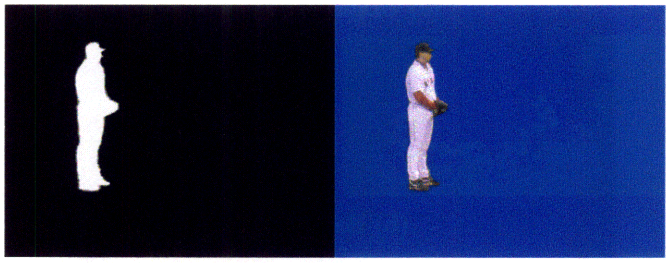
(g) Ambiguous regions marked in red

Figure 6-19: A close up view of some of the results in figure 6-18, together with the regions of ambiguity.

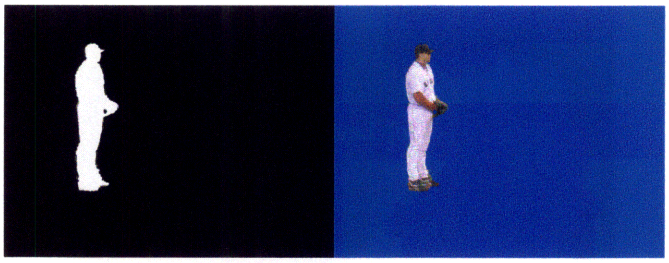
(a) Original Image



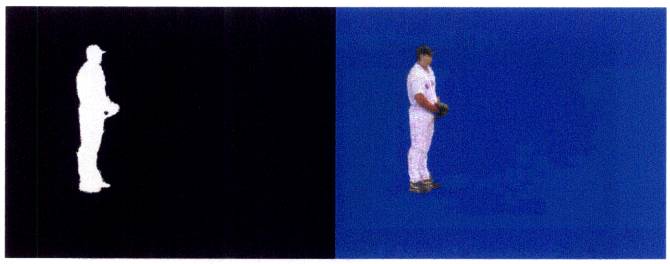
(b) Bayesian



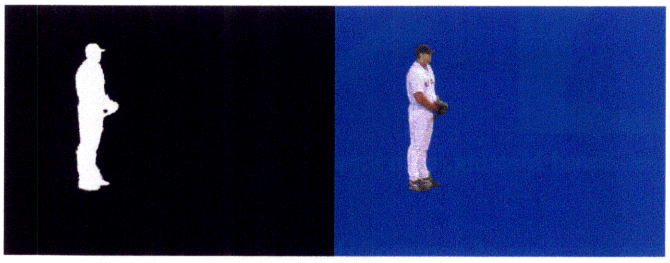
(c) Levin et al



(d) Bayesian with inpainting



(e) Segment-Based Matting, without inpainting



(f) Segment-Based Matting, with inpainting

Figure 6-20: Results of running the different algorithms on image 6-1(h), as well as compositing the results on a blue background.

(a) Original Image

(b) Bayesian

(c) Levin et al

(d) Bayesian with inpainting

(e) Segment-Based Matting, without inpainting

(f) Segment-Based Matting, with inpainting

Figure 6-21: Results of running the different algorithms on image 6-1(i), as well as compositing the results on a blue background.

(a) Original Image

(b) Bayesian

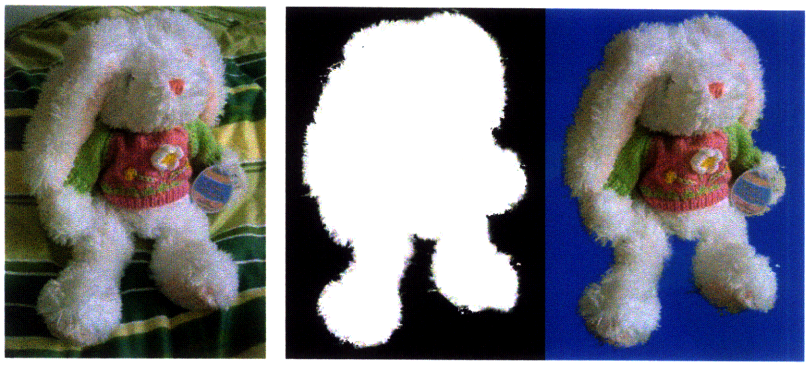
(c) Levin et al

(d) Bayesian with inpainting

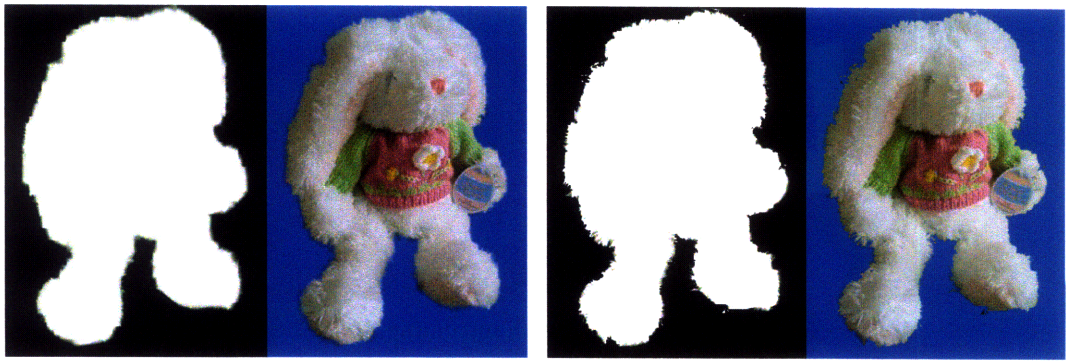
(e) Segment-Based Matting, without inpainting

(f) Segment-Based Matting, with inpainting

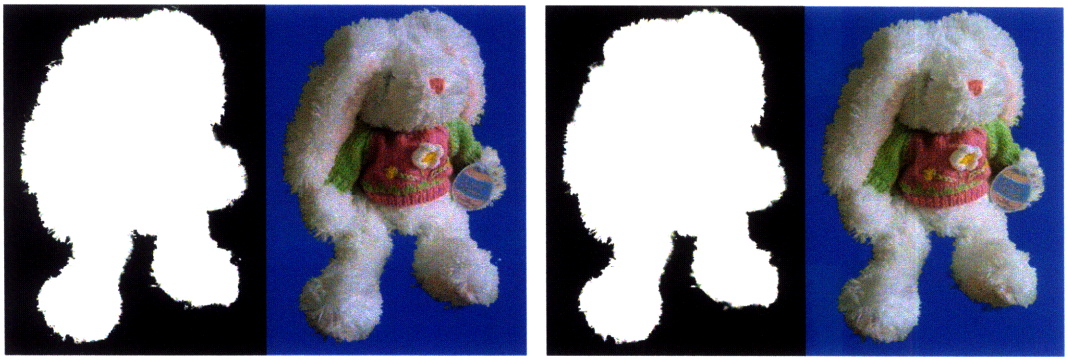
Figure 6-22: Results of running the different algorithms on image 6-1(j), as well as compositing the results on a blue background.



(a) Original Image (b) Bayesian

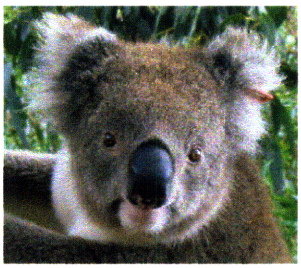


(c) Levin et al (d) Bayesian with inpainting



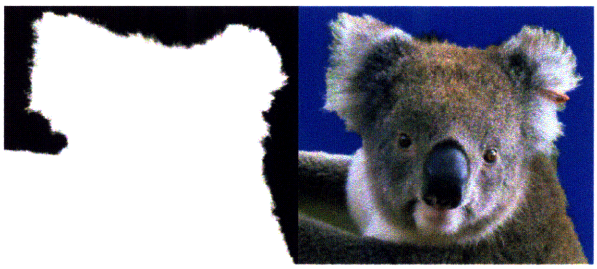
(e) Segment-Based Matting, without inpainting (f) Segment-Based Matting, with inpainting

Figure 6-23: Results of running the different algorithms on image 6-1(k), as well as compositing the results on a blue background.

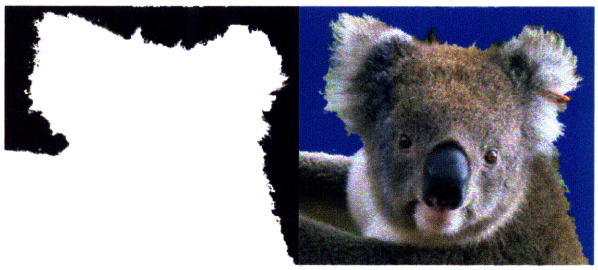


(a) Original Image

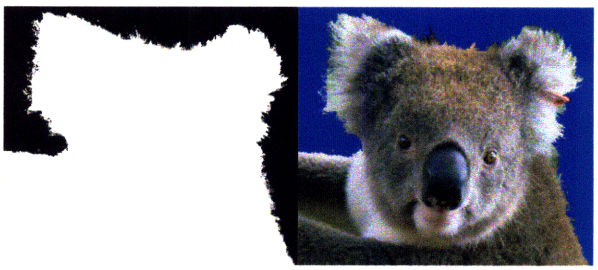
(b) Bayesian



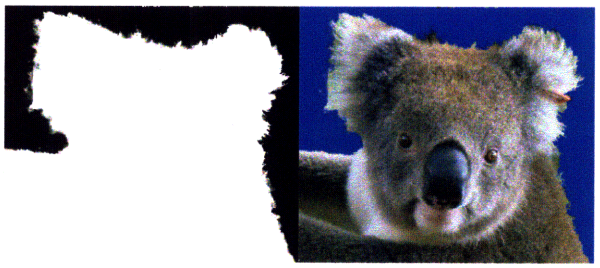
(c) Levin et al



(d) Bayesian with inpainting



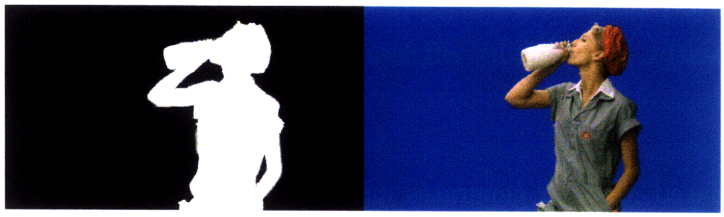
(e) Segment-Based Matting, without inpainting



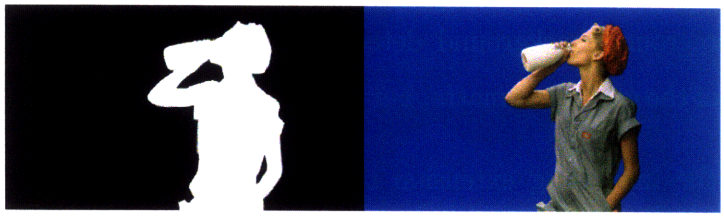
(f) Segment-Based Matting, with inpainting

Figure 6-24: Results of running the different algorithms on image 6-1(l), as well as compositing the results on a blue background.

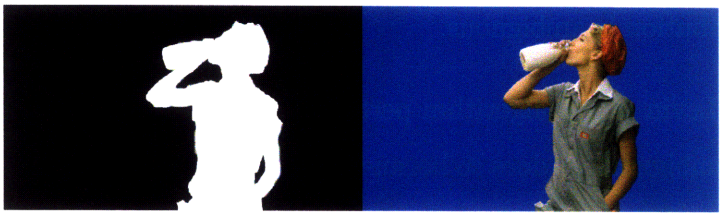
(a) Original Image



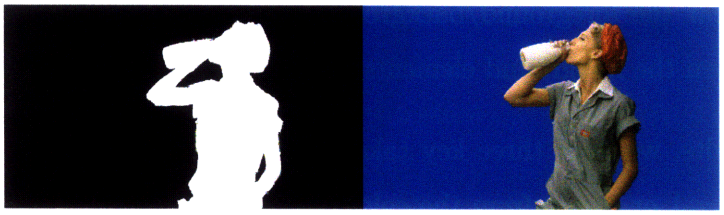
(b) Bayesian



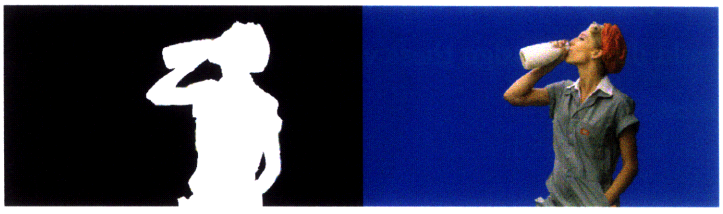
(c) Levin et al



(d) Bayesian with inpainting



(e) Segment-Based Matting, without inpainting



(f) Segment-Based Matting, with inpainting

Figure 6-25: Results of running the different algorithms on image 6-1(m), as well as compositing the results on a blue background.

6.6 Summary of Results and Further Discussion

In this chapter, we have demonstrated the following:

1. Segment-Based Matting is far more efficient than Bayesian Matting.

2. Adding foreground/background detection to Bayesian Matting improves its efficiency greatly, although its efficiency remains far inferior to Segment-Based Matting.

3. Segment-Based Matting generates mattes that are sharper and more accurate than both Bayesian Matting and Levin et al's closed form laplacian matting. Adding foreground/background detection to Bayesian Matting improves the sharpness of the generated matte, but is still inferior to Segment-Based Matting.

4. Applying the inpainting extension to both Bayesian Matting and Segment-Based Matting improves the accuracy of the matte in regions where the local color distribution is ambiguous.

5. Although Levin et al's algorithm performs well in areas where the color distribution is ambiguous, it does so because it enforces continuity of the alpha matte everywhere. This can lead to several problems, including mattes that are too smooth near the foreground/background boundary and incorrectly estimated mattes when the foreground elements have holes.

From these results, we infer three key take-aways: Firstly, for most pixels, we can directly infer $\alpha = 1$ or $\alpha = 0$ from the color distribution; this can be seen from the efficiency gain in Bayesian Matting after adding foreground/background detection. Therefore, we should directly assign these values; this can lead to increased efficiency and matte sharpness.

Secondly, modelling shading can improve the sharpness of the matte. The Bayesian Matting solution is not sharp – even with foreground/background detection – because of its bias towards means, which leads to fractional values of α in areas that are of the

same color as the foreground or background but of a different shade. Segment-Based Matting models shading and therefore results in a sharper and more accurate matte.

Thirdly, and most importantly, we propose an important idea in matting: Enforcing continuity directly can lead to undesirable results. It is more desirable – as in our approach of using inpainting to resolve ambiguous regions in Bayesian Matting or Shading-Based Matting – to try using color-based estimation first, and only enforce continuity when the uncertainty of color-based estimation is high. Alternatively, another suitable approach – and one that can be adapted to other matting algorithms that already enforce continuity – is to make the degree of local continuity enforced dependent on the local color information, such that a high amount of continuity is enforced when the color uncertainty is high, and a low amount of continuity is enforced when the color uncertainty is low. This allows good estimation when there is high color uncertainty, and a sharp and accurate matte when the foreground and background boundary is clear and sharp and the color uncertainty is low.

Chapter 7

Conclusions and Further Work

This thesis has three main contributions:

1. We have enumerated problems with Bayesian Matting.

2. Inspired by these problems, we have proposed Segment-Based Matting: A color-based statistical matting algorithm that models shading using color lines to provide sharper mattes. In addition, it has a closed form solution which improves efficiency.

3. We have proposed a new approach that uses inpainting or texture synthesis to resolve regions of color ambiguity in mattes; this allows a better matte to be obtained in areas of color ambiguity.

We have demonstrated that our combined approach – of using inpainting to resolve regions of color ambiguity in mattes generated by Segment-Based Matting – is efficient and generates sharper and more accurate mattes. From our results, we have obtained a few important ideas about matting. These ideas are summarized in section 6.6, but we will quickly repeat them here:

1. For most pixels, $\alpha = 0$ or $\alpha = 1$, and we can improve efficiency by detecting these pixels before performing any estimation.

2. Modelling Shading in Matting can improve the sharpness and accuracy of the matte.

3. Most importantly, the degree of local continuity enforced in the alpha matte by the matting algorithm should depend on the local color distribution; the more similar the local foreground and background color distributions are, the higher the amount of continuity enforced in the matte. This allows good estimation in areas where the color distribution is ambiguous – by enforcing continuity – but still ensures that the matte is sharp and accurate in areas of very distinct foreground and background color distributions, where continuity is not enforced.

We conclude this thesis by providing three directions of future work. Section 7.1 briefly discusses some issues regarding sampling. Section 7.2 discusses techniques to obtain better parameter choices in our algorithms. Finally, section 7.3 discusses applying our ideas regarding matting and inpainting to other matting algorithms. The discussion in section 7.1 is relatively minor in importance compared to the other two.

7.1 A Better Method of Sampling

One problem with any kind of sampling is a "horizon" effect: It is possible that the area from which you sample from is "just a little too small", and if we expanded the sampling radius by 1 pixel, we would capture important pixels. [35] has proposed a different technique for sampling from a trimap, where instead of sampling from a circular region around the pixel, we sample along the trimap boundaries near the pixel. This is better for regions with complex boundaries; such a sampling algorithm would greater capture the color variation along the boundary.

Another problem with sampling is the fact that the radius of sampling can be different for different pixels, since the radius expands until a minimum number of samples are collected. Ideally, we would want the local foreground and background color distributions to change "smoothly" as we move along the foreground/background boundary. However, if the radius of sampling changes along this boundary, this may not be the case, and can be a separate source of discontinuity in the estimated matte. Hence,

work on a better sampling technique may prove to be useful.

Finally, we noted in section 6.1 that sampling takes a significant proportion of time in Segment-Based Matting. Designing more efficient sampling algorithms and associated data structures can significantly improve the runtime efficiency of Segment-Based Matting.

7.2 Data-Dependent and User-Defined Parameters

The algorithms described in this thesis use many different parameters. These parameters are generally constant throughout the image. However, it may be useful for these parameters to be set dynamically throughout the image based on the data and inferred values; in some cases it may even be possible for the user to define different parameters for different areas in the image.

We list some possible examples below:

- In section 5.3.1, we listed techniques for smoothing an area before inpainting it, and suggested that it was also possible for the user to choose different smoothing parameters for different areas. It may also be possible to use other data available, such as the trimap and original image, to dynamically choose different smoothing parameters for different ambiguous regions.

- In section 4.3.8, we described a scoring and penalty system for evaluating a given foreground and background pair of clusters, in order to choose the pair that gave the best solution for α. This system utilized many different parameters. These parameters could be adjusted by the user in different areas of the image or dynamically modified based on the provided data. For example, in areas where the trimap is looser – and thus require more shading extrapolation – the penalty for deviating too far away from the mean could be reduced. Similarly, if the user believes that the degree of shading in a particular region should be very

large, he can reduce the penalty for substantially deviating from the mean in that region. As another example, if the calculated value of α was high, the foreground becomes far more important than the background, in which case the scoring system might emphasize more on the estimated foreground color rather than the estimated background color.

- In the line segment model, the line segment represents a permissable range of shading. We proposed a fairly arbitrary method of generating this line segment. It is possible that in certain regions – for example, in areas where the trimap is looser – the permissable range of shading should be larger. In this case, the line segment generated should be longer. Similarly, it is possible for the user to mark areas in the image where he believes the range of shading allowed should be different and specify the amount of shading in these regions.

7.3 Application of our ideas to other Matting Algorithms; creating an "matting-biased" inpainting algorithm

In chapter 5, we proposed a new approach to use inpainting to resolve ambiguous regions in mattes. This approach can be applied to other matting algorithms. Many algorithms – such as robust matting [35] and the iterative optimization approach [34] – have a measure of confidence for the estimated values at each pixel. It may be possible to modify these algorithms to inpaint over regions of low confidence.

Our proposed approach of using inpainting to resolve ambiguous regions – as described in chapter 5 – is an extreme example of an idea we described in section 6.6: The amount of local smoothing enforced on the alpha matte should depend on the color ambiguity: If the foreground and background color distributions are very distinct, the amount of smoothing enforced should be very low, while if the foreground and background color distributions are very similar, the amount of smoothing en-

forced should be much higher. Our approach is extreme because we do not perform any smoothing or inpainting unless the distributions are extremely similar. It is certainly possible to perform a "softer" version of this where the level of smoothing is adjusted based on the ambiguity of the local foreground and background color distributions.

The above idea can be applied to many matting algorithms that enforce smoothness; in these algorithms, each pixel usually has a parameter that enforces smoothness in its local area. In the original incarnation of these matting algorithms, this parameter is usually globally static and identical for all pixels; however, it may be worthwhile to dynamically change this parameter based on data such as the similarity of the local foreground and background color distributions.

Finally, we noted in section 6.4 that the inpainting algorithm could fill in ambiguous regions incorrectly because it did not utilize other information such as the trimap and the original image. It may be possible to adjust the parameters of the inpainting algorithm dynamically in a local region based on these information, creating a "matting-biased" inpainting algorithm. This is speculation and we have not put much thought into the process for obtaining such an algorithm; however, it may be worthwhile to devote research effort to this problem.

Appendix A

Linear Algebra and Important Operations

This chapter covers some important mathematical definitions and operations in elementary linear algebra that are relevant to this thesis. Many of the concepts presented here should be familiar to readers with a good grasp of elementary linear algebra; a good review of linear algebra can be found in [33].

For readers familiar with linear algebra, sections A.1 through A.3 may be skipped, but section A.4 should be quickly scanned through. Section A.5 briefly describes homogenous coordinates; for readers unfamiliar with this concept, a simple introduction can be found in [7].

A.1 Basic Definitions

Definition A.1. A **vector** is a one-dimensional array of values. In this thesis, all vectors are column vectors: A three-dimensional vector has size 3×1.

Definition A.2. A **matrix** is a two-dimensional array of values. For example,

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & -1 \\ 3 & 5 & 1 \end{bmatrix}$$

is a 2×3 matrix and its $\mathbf{A}_{1,2}$ entry is $\mathbf{A}_{1,2} = 2$. Note that a vector may also be viewed as a matrix with only one column. A **square** matrix is a matrix with the same number of rows and columns.

Definition A.3. The **transpose** operator, indicated by a superscript T, flips the rows and columns of a matrix. For example,

$$\begin{bmatrix} 1 & 2 & -1 \\ 3 & 5 & 1 \end{bmatrix}^T = \begin{bmatrix} 1 & 3 \\ 2 & 5 \\ -1 & 1 \end{bmatrix}$$

Definition A.4. A **symmetric** matrix is a square matrix that is equal to its transpose: $\mathbf{A}^T = \mathbf{A}$.

Definition A.5. Matrix **addition and subtraction** between two matrices is simply the addition/subtraction of the pairwise elements between the two matrices. The matrices have to be the same size. For example,

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + \begin{bmatrix} 3 & -1 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 4 & -1 \\ 2 & 2 \end{bmatrix}$$

Matrix addition is commutative and associative.

Suppose $\mathbf{A}$ is a matrix of size $a \times b$ and $\mathbf{B}$ is a matrix of size $b \times c$. Then matrix **multiplication** between $\mathbf{A}$ and $\mathbf{B}$ generates $\mathbf{C}$, a matrix of size $a \times c$ such that

$$C_{i,j} = \sum_k A_{i,k} B_{k,j}$$

To be able to multiply two matrices, the number of columns in the first matrix must equal the number of rows in the second matrix. Matrix multiplication is associative but **not** commutative.

Matrix multiplication and addition follow the distributive law. For example: $(\mathbf{A} + \mathbf{B})\mathbf{C}$

$= \mathbf{AC} + \mathbf{BC}$.

Definition A.6. The **identity** matrix I of a given size is the square matrix with the diagonal entries equal to 1 and all other entries zero. For example, the identity matrix of size 3 is

$$I_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Definition A.7. The **inverse** of a square matrix is the matrix that multiplies the original matrix (following the definition of matrix multiplication) to give the identity matrix. If $\mathbf{A}$ is a square matrix, then $\mathbf{A}\mathbf{A}^{-1} = \mathbf{A}^{-1}\mathbf{A} = I$. Note that inverses only exist for square matrices and inverses are themselves square matrices. If a matrix has an inverse, it is **invertible**; if not, it is **singular**.

Remark A.8. The equation $\mathbf{Ax} = \mathbf{b}$, where $\mathbf{A}$ is a matrix, $\mathbf{x}$ and $\mathbf{b}$ are vectors of the appropriate sizes, and with $\mathbf{A}$ and $\mathbf{b}$ known and $\mathbf{x}$ unknown is often referred to as a **linear system of equations**. If $\mathbf{A}$ is a square invertible matrix, then a simple solution is $\mathbf{x} = \mathbf{A^{-1}b}$ which is easily verified by substitution: $\mathbf{Ax} = \mathbf{AA^{-1}b} = \mathbf{Ib} = \mathbf{b}$.

Definition A.9. The **vector scalar product**, also known as the **dot product** between two vectors of identical sizes is the sum of the pairwise product of each component. For example,

$$\begin{bmatrix} 1 \\ -1 \end{bmatrix} \cdot \begin{bmatrix} 2 \\ 3 \end{bmatrix} = (1 \times 2) + (-1 \times 3) = -1$$

A convenient way to obtain the product using matrix multiplication is to take the transpose of the first vector and multiply it by the second (see definitions A.3 and A.5): For example, we may write the dot product between vectors $\mathbf{a}$ and $\mathbf{b}$ as $\mathbf{a^T b}$.

Definition A.10. The **vector cross product**, or **cross product** for short between

two 3×1 vectors is the vector defined as:

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \times \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} a_2b_3 - a_3b_2 \\ a_3b_1 - a_1b_3 \\ a_1b_2 - a_2b_1 \end{bmatrix}$$

Definition A.11. Two vectors are **orthogonal** if their dot product is zero. Orthogonality is equivalent to two vectors being perpendicular.

Property A.12. If $\mathbf{c} = \mathbf{a} \times \mathbf{b}$, then $\mathbf{c}$ is orthogonal to both $\mathbf{a}$ and $\mathbf{b}$. This can be easily verified by taking the dot product of $\mathbf{c}$ (defined in definition A.10) with $\mathbf{a}$ or $\mathbf{b}$.

Definition A.13. The **magnitude**, **length** or **norm** of a vector is defined as the square root of its dot product with itself: $||\mathbf{a}|| = \sqrt{\mathbf{a}^T\mathbf{a}}$. A vector can be normalized to be a **unit vector** (i.e. with a norm of 1) by dividing itself by its norm.

Definition A.14. A collection of n different vectors in n-dimensional space is called an **orthonormal basis** if the following two conditions hold:

1. Every vector has unit magnitude (of 1).
2. The vectors are pairwise orthogonal.

These vectors are then denoted as the **basis vectors** of the orthonormal basis. A simple example of an orthonormal basis is the columns of an identity matrix (each column is a basis vector).

Property A.15. Fix an arbitrary orthonormal basis in n-dimensional space. Any point in n-dimensional space can be represented as a linear combination of the basis vectors. Furthermore, this combination is unique. Equivalently, if we place the basis vectors as the columns of a matrix $\mathbf{A}$, then the equation $\mathbf{Ax} = \mathbf{b}$ has exactly one solution $\mathbf{x}$ for any n-dimensional vector $\mathbf{b}$.

Definition A.16. An **orthogonal** matrix is a matrix whose inverse is its transpose: $\mathbf{A}$ is an orthogonal matrix iff $\mathbf{AA^T} = \mathbf{A^TA} = I$. The columns of an orthogonal matrix form an orthonormal basis; similarly, an orthogonal matrix can be formed by inserting the basis vectors of an orthonormal basis as its columns.

Property A.17. Multiplying by an orthogonal matrix preserves the magnitude of a vector: If $\mathbf{A}$ is an orthogonal matrix, then $||\mathbf{Ax}|| = ||\mathbf{x}||$ for all vectors $\mathbf{x}$. Furthermore, multiplying by an orthogonal matrix also preserves dot products: $(\mathbf{Ax})^T(\mathbf{Ay}) = \mathbf{x^T y}$.

This property is easily derived from definition A.16: $(\mathbf{Ax})^T(\mathbf{Ay}) = \mathbf{x^T A^T Ay} = \mathbf{x^T y}$. Substituting $\mathbf{x} = \mathbf{y}$ yields $||\mathbf{Ax}||^2 = ||\mathbf{x}||^2$, or $||\mathbf{Ax}|| = ||\mathbf{x}||$.

Remark A.18. The above property is equivalent to saying that multiplying by an orthogonal matrix *preserves distances and angles.*

Definition A.19. A symmetric $n \times n$ matrix $\mathbf{A}$ is **positive definite** if for any $n \times 1$ non-zero vector $\mathbf{v}$,

$$\mathbf{v^T Av} > 0$$

If we allow the possibility of equality in the above criteria, so that we replace $>$ with $\geq$, then $\mathbf{A}$ is termed **positive semidefinite**.

A.1.1 Lines

A line can be defined in many different ways:

1. A line can be defined with a point $\mathbf{p}$ and a slope $\mathbf{v}$, in which case the line is defined by all points $\mathbf{p} + t\mathbf{v}, t \in (-\infty, \infty)$.

2. A line can also be defined by two points $\mathbf{p_1}$ and $\mathbf{p_2}$ on it. In this case, it can be transformed into the previous definition of a line: Take the point $\mathbf{p} = \mathbf{p_1}$ and the slope $\mathbf{v} = \mathbf{p_2} - \mathbf{p_1}$.

3. Finally, a line can be defined by all points satisfying a set of algebraic equations. The equations needed will depend on the dimensionality of the space, but we will mention the two-dimensional case in the (x, y) Cartesian plane. There are at least two different ways of defining a line in the Cartesian plane:

(a) A line can be defined as $y = mx + c_1$ for non-vertical lines and $x = c_2$ for vertical lines, where m denotes the slope, c_1 the y-intercept and c_2 the x-intercept.

(b) A line can be defined as $ax + by + c = 0$. In this case, the slope is $-a/b$, the y-intercept is $-c/b$ and the x-intercept as $-c/a$.

A.1.2 Planes

In this section, we will assume *three-dimensional* space.

A plane can be defined in two different ways:

1. A plane can be defined by a point $\mathbf{p}$ on the plane, and a vector $\mathbf{v}$ that is perpendicular (or orthogonal) to the plane. In this case the plane can be defined by all points $\mathbf{x}$ such that $(\mathbf{x} - \mathbf{p})^{\mathbf{T}}\mathbf{v} = 0$, using the definition of orthogonality (definition A.11). See figure A-1 for a graphical interpretation.

2. A plane can be defined by three non-collinear points $\mathbf{p_1}, \mathbf{p_2}$ and $\mathbf{p_3}$ that all lie on the plane. In this case, we can convert it to the previous definition of the plane. Since $\mathbf{p_2} - \mathbf{p_1}$ and $\mathbf{p_3} - \mathbf{p_1}$ are non-collinear vectors that both lie on the plane, any vector perpendicular to both these vectors is perpendicular to the plane. Therefore, using property A.12 we can obtain a vector $\mathbf{v}$ perpendicular to the plane via $\mathbf{v} = (\mathbf{p_2} - \mathbf{p_1}) \times (\mathbf{p_3} - \mathbf{p_1})$. For the point $\mathbf{p}$ on the plane, we can arbitrarily choose one of $\mathbf{p_1}, \mathbf{p_2}$ or $\mathbf{p_3}$.

A.2 Projections

The projection of a point $\mathbf{x}$ onto a line/plane is the point $\hat{\mathbf{x}}$ on the line/plane that has the smallest distance to $\mathbf{x}$ i.e. the point that minimizes $||\mathbf{x} - \hat{\mathbf{x}}||$. In this section, we assume three-dimensional space. The following subsections will discuss the derivation of formulas to project a point onto a line or plane.

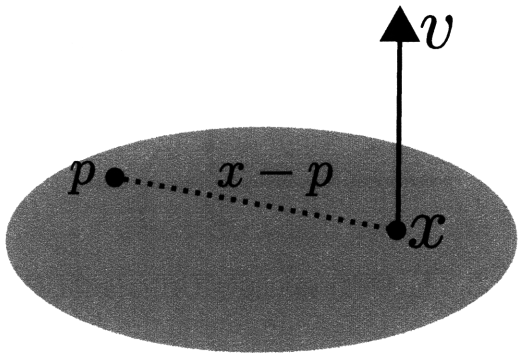


Figure A-1: A Graphical Depiction of Defining a Plane.

A.2.1 Orthogonality Criteria

A fairly intuitive criteria for deriving the projection formulas is that the line $\mathbf{x} - \hat{\mathbf{x}}$ is perpendicular to the line or plane on which $\mathbf{x}$ is to be projected on. This criteria is known as the **orthogonality criteria** for projection.

A proof of the orthogonality criteria proceeds as follows: There exists a point $\mathbf{y}$ on the line/plane such that the line $\mathbf{x} - \mathbf{y}$ is perpendicular to the line/plane. Suppose the line $\mathbf{x} - \hat{\mathbf{x}}$ was not perpendicular to the line/plane. Since $\hat{\mathbf{x}}$ lies on the line/plane, the line segment from $\hat{\mathbf{x}}$ to $\mathbf{y}$ also lies on the line/plane. Therefore, the line segment from $\mathbf{x}$ to $\mathbf{y}$ is perpendicular to the line segment from $\hat{\mathbf{x}}$ to $\mathbf{y}$, and by Pythagoras' formula

$$||\mathbf{x} - \hat{\mathbf{x}}||^2 = ||\mathbf{x} - \mathbf{y}||^2 + ||\hat{\mathbf{x}} - \mathbf{y}||^2$$

and since $\hat{\mathbf{x}} \neq \mathbf{y}$ by assumption, it follows that $||\mathbf{x} - \hat{\mathbf{x}}|| > ||\mathbf{x} - \mathbf{y}||$, which contradicts the assumption that $\hat{\mathbf{x}}$ minimizes $||\mathbf{x} - \hat{\mathbf{x}}||$. Hence we have proven the orthogonality criteria by contradiction. See figure A-2 for a visual depiction.

A.2.2 Projecting Onto a Line

We will assume that the line can be represented, following the first definition in section A.1.1, as a point $\mathbf{p}$ and a slope $\mathbf{v}$. Since $\hat{\mathbf{x}}$ must lie on the line, we may write

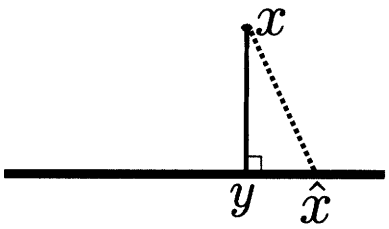


Figure A-2: The Orthogonality Criteria for Projection

$\hat{\mathbf{x}} = \mathbf{p} + t\mathbf{v}$, with t to be determined. Following the orthogonality criteria, the line segment $\mathbf{x} - \hat{\mathbf{x}}$ must be perpendicular to the slope $\mathbf{v}$, and hence

$$(\mathbf{x} - \hat{\mathbf{x}})^T\mathbf{v} = (\mathbf{x} - \mathbf{p} - t\mathbf{v})^T\mathbf{v} = 0$$

and after some rearrangement we obtain

$$t = \frac{(\mathbf{x} - \mathbf{p})^\mathbf{T}\mathbf{v}}{\mathbf{v}^\mathbf{T}\mathbf{v}} \tag{A.1}$$

and hence we obtain the projection

$$\hat{\mathbf{x}} = \mathbf{p} + t\mathbf{v} = \mathbf{p} + \frac{(\mathbf{x} - \mathbf{p})^\mathbf{T}\mathbf{v}}{\mathbf{v}^\mathbf{T}\mathbf{v}}\mathbf{v} \tag{A.2}$$

A.2.3 Projecting Onto a Plane

We will assume that the plane can be represented, following the first definition in section A.1.2, as a point on the plane $\mathbf{p}$ and a vector perpendicular to the plane $\mathbf{v}$. We will use an algebraic solution, although an alternate approach using the orthogonality property will also be sketched out.

We wish to find $\hat{\mathbf{x}}$ that minimizes $||\mathbf{x} - \hat{\mathbf{x}}||$, or equivalently minimizing half its square $\frac{1}{2}(\mathbf{x} - \hat{\mathbf{x}})^T(\mathbf{x} - \hat{\mathbf{x}})$, subject to the condition that $\hat{\mathbf{x}}$ lies on the plane, or equivalently $(\hat{\mathbf{x}} - \mathbf{p})^T\mathbf{v} = 0$. We add a Lagrange multiplier λ and obtain the objective function:

$$\frac{1}{2}(\mathbf{x} - \hat{\mathbf{x}})^T(\mathbf{x} - \hat{\mathbf{x}}) - \lambda(\hat{\mathbf{x}} - \mathbf{p})^T\mathbf{v}$$

This is convex in $\hat{\mathbf{x}}$, so the first order condition with respect to $\hat{\mathbf{x}}$ will yield a minima. Taking the first order condition and simplifying yields

$$\hat{\mathbf{x}} = \lambda \mathbf{v} + \mathbf{x} \tag{A.3}$$

Substituting (A.3) into $(\hat{\mathbf{x}} - \mathbf{p})^T \mathbf{v} = 0$ and simplifying will give us

$$\lambda = \frac{(\mathbf{p} - \mathbf{x})^\mathbf{T} \mathbf{v}}{\mathbf{v}^T \mathbf{v}}$$

and substituting back into (A.3) gives us the solution

$$\hat{\mathbf{x}} = \lambda \mathbf{v} + \mathbf{x} = \frac{(\mathbf{p} - \mathbf{x})^\mathbf{T} \mathbf{v}}{\mathbf{v}^T \mathbf{v}} \mathbf{v} + \mathbf{x} \tag{A.4}$$

We will also quickly sketch out a derivation using the orthogonality property. From the orthogonality property, we know that $\mathbf{x} - \hat{\mathbf{x}}$ is perpendicular to the plane and thus parallel to $\mathbf{v}$. Let us assume for a moment that $\mathbf{p}$ is the origin. Let the point $\mathbf{y}$ be the projection of $\mathbf{x}$ onto the line with slope $\mathbf{v}$ running through the origin. The line $\mathbf{x} - \mathbf{y}$ is perpendicular to $\mathbf{v}$ and satisfies the condition $(\mathbf{x} - \mathbf{y} - \mathbf{p})^T \mathbf{v} = 0$ (since $\mathbf{p}$ is the origin), and is thus on the plane. Since $\mathbf{x} - (\mathbf{x} - \mathbf{y}) = \mathbf{y}$ is parallel to $\mathbf{v}$ (as it is on the line with slope $\mathbf{v}$ passing through the origin), the solution is therefore given by $\mathbf{x} - \mathbf{y}$. If $\mathbf{p}$ is non-zero, we have to additionally add the projection of $\mathbf{p}$ onto the line with slope $\mathbf{v}$ running through the origin. Therefore the answer is

$$\hat{\mathbf{x}} = \mathbf{x} - \text{projection of } \mathbf{x} \text{ on } \mathbf{v} + \text{projection of } \mathbf{p} \text{ on } \mathbf{v}$$

Using equation (A.2) to calculate the projections of $\mathbf{x}$ and $\mathbf{p}$ on the line defined by the origin point and the slope $\mathbf{v}$, we obtain the answer

$$\hat{\mathbf{x}} = \mathbf{x} - \frac{\mathbf{x}^\mathbf{T} \mathbf{v}}{\mathbf{v}^\mathbf{T} \mathbf{v}} \mathbf{v} + \frac{\mathbf{p}^\mathbf{T} \mathbf{v}}{\mathbf{v}^\mathbf{T} \mathbf{v}} \mathbf{v}$$

which of course is equal to (A.4).

A.3 Eigenvectors and Eigenvalues

Given a square matrix $\mathbf{A}$ of size $n \times n$, we wish to find a non-zero vector $\mathbf{x}$ of size $n \times 1$ and a scalar λ satisfying

$$\mathbf{Ax} = \lambda\mathbf{x} \tag{A.5}$$

The scalar λ is called an **eigenvalue** of $\mathbf{A}$ and its corresponding vector $\mathbf{x}$ is called the corresponding **eigenvector**. There will be n eigenvalues, not always unique and possibly complex-valued. In the general case, some eigenvalues may have an infinite number of associated unit-length eigenvectors. However, there exists the following extremely useful and non-trivial property, which we will state without proof (see [33] for a proof):

Property A.20. If $\mathbf{A}$ is *symmetric*, all eigenvalues are real. Furthermore, there will exist exactly n unit-length eigenvectors, and each of these eigenvectors are orthogonal to each other.

From this point on, we will assume that the matrix $\mathbf{A}$ is symmetric and that property A.20 holds. Without loss of generality, we shall assume that each of the eigenvectors have unit length: The eigenvectors may be normalized by dividing by their length, and (A.5) will still be satisfied. Hence, following property A.20 and recalling definition A.14, we arrive at the following property:

Property A.21. The (normalized) eigenvectors of a symmetric matrix form an orthonormal basis.

A.3.1 Eigenvalue Decomposition of Symmetric Matrices

For a given symmetric matrix $\mathbf{A}$, let the orthogonal matrix $\mathbf{Q}$ be such that its columns contain the n different normalized eigenvectors of $\mathbf{A}$. Let $\mathbf{S}$ be a matrix that is zero everywhere except on the diagonals, and place the eigenvalues of $\mathbf{A}$ on its diagonals, in such a way that $S_{1,1}$ contains the eigenvalue corresponding to the eigenvector on

the first column of $\mathbf{Q}$, $S_{2,2}$ contains the eigenvalue corresponding to the eigenvector on the second column of $\mathbf{Q}$, and so on.

By repeating equation (A.5) for each eigenvalue/eigenvector pair, and placing the results in matrix form, it can be worked out that

$$\mathbf{AQ} = \mathbf{QS}$$

By multiplying $\mathbf{Q}^{-1}$ on both sides of the above equation, and recalling definition A.16, we obtain the **eigenvalue decomposition** of $\mathbf{A}$:

$$\mathbf{A} = \mathbf{QSQ}^{-1} = \mathbf{QSQ}^{\mathbf{T}} \tag{A.6}$$

where $\mathbf{S}$ is a diagonal matrix containing the eigenvalues of $\mathbf{A}$ on its diagonals and $\mathbf{Q}$ is an orthogonal matrix whose columns are the eigenvectors of $\mathbf{A}$, corresponding to the eigenvalues on the diagonal elements of $\mathbf{S}$.

A.3.2 Rayleigh Quotient for Symmetric Matrices

The Rayleigh quotient for a fixed symmetric matrix $\mathbf{A}$ and an arbitrary vector $\mathbf{v}$ is defined as

$$\frac{\mathbf{v}^{\mathbf{T}}\mathbf{Av}}{\mathbf{v}^{\mathbf{T}}\mathbf{v}} \tag{A.7}$$

For simplicity of exposition, we will restrict ourselves to vectors $\mathbf{v}$ of unit magnitude, so that the denominator of (A.7) is 1; this implies that our proofs will not directly translate over to the general case, however, the results obtained will still be true in the general case.

Consider the orthogonal matrix $\mathbf{Q}$ from section A.3.1, which contained the normalized eigenvalues of $\mathbf{A}$ on its columns. Without loss of generality, we will assume that its columns are sorted according to eigenvalue, so that the first column of $\mathbf{Q}$ contains the

eigenvector corresponding to the largest eigenvalue, the second column of $\mathbf{Q}$ contains the eigenvector corresponding to the second largest eigenvalue, etc. Similarly, the diagonal entries of the matrix $\mathbf{S}$ in the Eigenvalue Decomposition (A.6) of $\mathbf{A}$ decrease down the diagonal.

Since the columns of $\mathbf{Q}$ form an orthonormal basis, any vector $\mathbf{v}$ may be written as $\mathbf{v} = \mathbf{Qy}$ for some vector $\mathbf{y}$ (see property A.15). With $\mathbf{v}$ of unit magnitude, and using the Eigenvalue Decomposition (A.6), we can rewrite (A.7) as

$$\mathbf{v^T A v} = (\mathbf{Qy})^\mathbf{T}\mathbf{A}(\mathbf{Qy}) = \mathbf{y^T Q^T}(\mathbf{QSQ^T})\mathbf{Qy} = \mathbf{y^T S y} \tag{A.8}$$

since $\mathbf{Q^T Q} = I$ from definition A.16. Since $\mathbf{v}$ has unit magnitude, it follows from property A.17 that $\mathbf{y}$ also has unit magnitude. Since $\mathbf{S}$ is a diagonal matrix with entries decreasing down the diagonal, the vector $\mathbf{y}$ that maximizes (A.8) is simply $[1\,0\,0\ldots]^T$. Thus the vector $\mathbf{v} = \mathbf{Qy}$ that maximizes the Rayleigh Quotient is simply the eigenvector corresponding to the largest eigenvalue of $\mathbf{A}$, which we will term as the *largest eigenvector.*

Suppose we wanted the vector that maximized (A.8) but had to be orthogonal to the largest eigenvector. Since this vector has to be orthogonal to the largest eigenvector, the first element of $\mathbf{y}$ has to be zero. Following the same reasoning as previously, it follows that the solution is $\mathbf{y} = [0\,1\,0\ldots]^T$, and the vector $\mathbf{v} = \mathbf{Qy}$ is the eigenvector corresponding to the second largest eigenvalue of $\mathbf{A}$. We can continue this reasoning to obtain the following important property (which is also true in the general case when we do not restrict the magnitude of $\mathbf{v}$ to be 1):

Property A.22. Suppose we wanted a vector $\mathbf{v}$ that maximized (A.7) subject to the condition that $\mathbf{v}$ was orthogonal to the eigenvectors corresponding to the largest k eigenvalues of $\mathbf{A}$. The solution is that $\mathbf{v}$ is the eigenvector corresponding to the $(k+1)th$ largest eigenvalue of $\mathbf{A}$. Note that scaling this eigenvector (multiplying it by a non-zero constant) will yield the same value of (A.7).

A.4 Transformations

This section will describe some transformations of points and lines in two and three dimensional space using tools of linear algebra.

A.4.1 Rotations in two dimensions

The rotation matrix that rotates points in two-dimensional Cartesian space counterclockwise around the origin by angle θ is given by

$$T_\theta = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \tag{A.9}$$

Hence, the rotation of $\mathbf{x}$ counterclockwise around the origin by θ radians is given by $T_\theta \mathbf{x}$.

An informal derivation of (A.9), which assumes that rotation is a linear operation and hence follows the law of superposition, is as follows: The point on the x-axis $[1 \quad 0]^T$ becomes $[\cos\theta \quad \sin\theta]^T$ when rotated counterclockwise around the origin by angle θ and similarly the point on the y-axis $[0 \quad 1]^T$ becomes $[-\sin\theta \quad \cos\theta]^T$. Following the law of superposition, point $\mathbf{x} = [x_1 \quad x_2]^T$ when rotated counterclockwise by angle θ becomes

$$x_1 \begin{bmatrix} \cos\theta \\ \sin\theta \end{bmatrix} + x_2 \begin{bmatrix} -\sin\theta \\ \cos\theta \end{bmatrix}$$

or, when put in matrix form, $T_\theta \mathbf{x}$, as desired.

The usual interpretation of the matrix T_θ is that it rotates points counterclockwise around the origin by θ, as we have described. However, an alternate interpretation is extremely useful:

Observation A.23. Another interpretation of the matrix T_θ is that it rotates the

coordinate system **clockwise** by angle θ. To obtain some intuition for this, consider looking at a point on a sheet of paper. There are two ways to view this point rotated counterclockwise around some given origin: Either we rotate the sheet of paper counterclockwise, thus rotating the point on it (this is the original interpretation), or we may rotate our head clockwise, thus changing the way we view the sheet of paper (this is the alternate interpretation described here).

We conclude this subsection with a very important property:

Property A.24. Following definition A.16, the matrix T_θ is an orthogonal matrix, and thus its inverse is its transpose. Following remark A.18, we may also conclude that the rotation transformation preserves distances between points and angles between lines.

A.4.2 Rotating a Plane to make a Given Line Vertical

In this thesis, we will require rotating points in a plane around the origin to make a given line vertical. We can do the operations we desire in the transformed space after rotation, and then do the inverse rotation to obtain the results in the original space. For this transform $\rightarrow$ compute $\rightarrow$ inverse transform to work for our purposes, we require that the transformation preserve distances and angles. However, from property A.24, the rotation transformation does have this property.

Thus, to make a given line vertical, we will need the appropriate rotation matrix, and hence all we require is the appropriate rotation angle. We will first assume that the line is not already vertical (otherwise no transformation is required), and thus according to the third representation in section A.1.1, we may write the equation of the line (in 2-dimensional Cartesian space) as $y = mx + c$. The slope of the line is m and its angle with the x-axis is given by $|\arctan(m)|$ – see figure A-3. We take absolute values so as to avoid having negative angles.

We will use the interpretation in observation A.23 to now derive the correct angle.

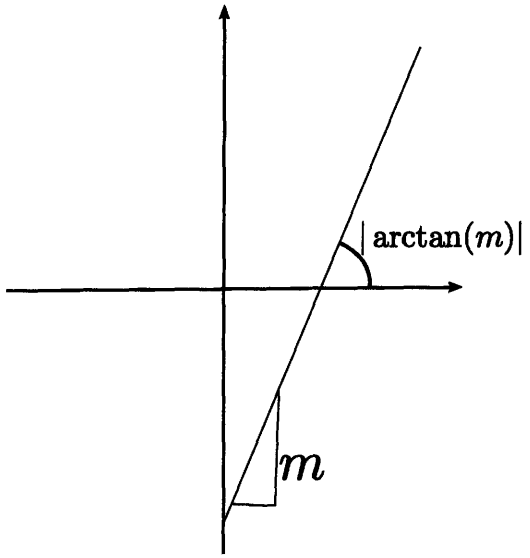


Figure A-3: Rotating a Line to be Vertical

First assume a positive slope m, as shown in figure A-3. We will need to rotate the coordinate axes clockwise by an angle of $\frac{\pi}{2} - |\arctan(m)|$. Now suppose the slope m is negative. In this case, we also rotate the coordinate axes by the same angle, but *counterclockwise*, or in the negative direction. To conclude, our rotation matrix is given by (A.9) with angle θ given by

$$\theta = sgn(m)\left(\frac{\pi}{2} - |\arctan(m)|\right)$$

where $sgn(m)$ takes the value 1 if $m \geq 0$ and -1 otherwise.

A.4.3 Getting 2-D coordinates of Points in a Plane

The previous sections have discussed operations in two dimensions in a Cartesian coordinate system. However, more often, we work with planes in three-dimensional space. We therefore require a transformation to map from a plane in three-dimensions to a two-dimensional Cartesian coordinate system and vice versa. As in section A.4.2, we will require that this transformation preserve distances and angles. In addition, we will also require that a given point $\mathbf{p}$ on the plane be mapped to the origin in this

transformation. As before, we will represent the plane by the point $\mathbf{p}$ on the plane and a vector $\mathbf{v}$ perpendicular to the plane.

This transformation is not unique; for example, we may multiply the resulting transformed points (in Cartesian coordinates) by any rotation matrix and we will obtain another transformation that also preserves distances and lines. Therefore, we will simply state one way of obtaining this transformation.

First, we need to obtain two vectors $\mathbf{a}$ and $\mathbf{b}$ both parallel to the plane and orthogonal to each other. One way of obtaining these vectors, if the points were generated by a statistical process, would be to use the first two principal components returned by Principal Component Analysis (see Appendix C). More generally, we may obtain vector $\mathbf{a}$ by projecting a random point to the plane and using the vector from the projected point to $\mathbf{p}$ (if its projection is $\mathbf{p}$, repeat the process!). We can then obtain $\mathbf{b} = \mathbf{a} \times \mathbf{v}$ using property A.12.

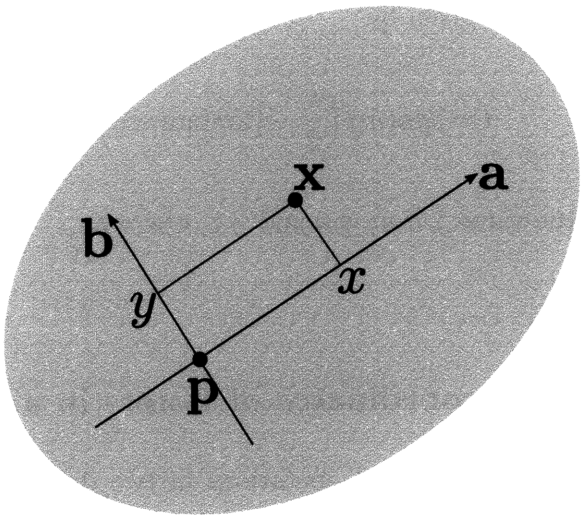


Figure A-4: Transforming from Three to Two dimensions

In order for the transformation to preserve distances and angles, we will normalize both $\mathbf{a}$ and $\mathbf{b}$ by dividing by their magnitudes, so they are of both unit magnitude. We will use the normalized $\mathbf{a}$ and $\mathbf{b}$ vectors as the x and y axes of our two dimensional

Cartesian plane. Now for any point $\mathbf{x}$ on the plane, we may obtain its x and y coordinates in Cartesian space by projecting the vector $(\mathbf{x} - \mathbf{p})$ to the $\mathbf{a}$ and $\mathbf{b}$ vectors respectively and obtaining the distances along the vectors. This may be done using equation (A.1) – setting the denominator to 1 since both $\mathbf{a}$ and $\mathbf{b}$ are unit vectors – to give us the following transformation to obtain two-dimensional Cartesian Coordinates:

$$\mathbf{x_{2D}} = \begin{bmatrix} \mathbf{a}^T \\ \mathbf{b}^T \end{bmatrix} (\mathbf{x} - \mathbf{p})$$

This transformation can be visualized in figure A-4. The inverse transformation can be easily derived using the interpretation provided by the figure:

$$\mathbf{x} = \mathbf{p} + [\mathbf{a} \quad \mathbf{b}]\mathbf{x_{2D}}$$

A.5 Homogenous Coordinates in Two Dimensions

This section briefly describes the use of Homogenous Coordinates in Two Dimensions to calculate line intersections and obtain lines from points.

A.5.1 Representing Lines and Points Using Homogenous Coordinates

A point in Cartesian space can be represented as (x, y). In homogenous coordinates, a third *scaling* parameter is added: The point (x, y, w) in homogenous coordinates is equivalent to $(x/w, y/w)$ in Cartesian space, with w being the scaling parameter. If $w = 0$, there is no equivalent representation in Cartesian space; rather, we may think of $(x, y, 0)$ as representing a *vector* in the (x, y) direction.

We recall from the third definition of a line in section A.1.1 that a line can be repre-

sented as $ax + by + c = 0$. Since this representation is invariant to scaling, we may rewrite this as $\mathbf{l} \cdot \mathbf{p} = 0$ where $\mathbf{l} = (a, b, c)$ is the representation of the line in homogenous coordinates and $\mathbf{p} = (x, y, w)$ is a point on the line in homogenous coordinates.

We summarize:

1. A point can be represented as (x, y, w); its equivalent representation in Cartesian coordinates is $(x/w, y/w)$.

2. A line can be represented as (a, b, c), representing the line $ax + by + c = 0$.

A.5.2 Obtaining a line passing through two points

Given two points $\mathbf{p_1} = (x_1, y_1, w_1)$ and $\mathbf{p_2} = (x_2, y_2, w_2)$ in homogenous coordinates, we wish to find a line passing $\mathbf{l} = (a, b, c)$ passing through both these points.

From the line equation, it suffices for $\mathbf{l} \cdot \mathbf{p_1} = 0$ and $\mathbf{l} \cdot \mathbf{p_2} = 0$. Equivalently, $\mathbf{l}$ is orthogonal to both $\mathbf{p_1}$ and $\mathbf{p_2}$ if they were represented as vectors in *3-dimensional* space! We now know how to obtain $\mathbf{l}$: From property A.12 of cross products, it follows that $\mathbf{l} = \mathbf{p_1} \times \mathbf{p_2}$.

A.5.3 Obtaining the intersection between two lines

Given two lines $\mathbf{l_1} = (a_1, b_1, c_1)$ and $\mathbf{l_2} = (a_2, b_2, c_2)$ in homogenous coordinates, we wish to find their intersection $\mathbf{p} = (x, y, w)$.

As in the previous section, it suffices for $\mathbf{l_1} \cdot \mathbf{p} = 0$ and $\mathbf{l_2} \cdot \mathbf{p} = 0$, and following the same logic, the solution is $\mathbf{p} = \mathbf{l_1} \times \mathbf{l_2}$.

Remark A.25. As can be seen from the previous two subsections, one advantage of homogenous coordinates is that everything can be done with cross products, without needing to worry about special cases. See the next remark for an example of a "special" case that is handled naturally using homogenous coordinates.

Remark A.26. In the line intersection case, what happens if the lines are parallel? It turns out that the point generated is of the form $\mathbf{p} = (x, y, 0)$, which represents a vector, or a "point at infinity". This point can be manipulated in the same way as other points; for example, if we take two parallel lines, take their intersection (the "point at infinity"), and compute the line running through this intersection and another point $\mathbf{x}$, the resulting line in homogenous coordinates is the line passing through $\mathbf{x}$ that was parallel to the first two lines, which is the expected solution.

Appendix B

Statistics and Statistical Algorithms

This chapter introduces some of the statistics, estimation and inference procedures that are relevant to this thesis. A more detailed overview of statistics, estimation and inference can be obtained from [13] and [23]. We assume a basic knowledge of elementary probability (measure theory not required); an overview can be found in [13].

B.1 Basic Statistics

The following definitions are given under the assumption that $x_1, \ldots, x_n$ are a sequence of scalar-valued samples.

Definition B.1. The **sample mean** is its average: $\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$.

Definition B.2. The **sample variance**[1] is the average squared deviation from the mean: $\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2$.

[1] As an aside, the *unbiased estimator for the variance* uses the exact same formula, except that the normalizing factor is $\frac{1}{n-1}$ instead of $\frac{1}{n}$. In practice, however, the difference between the normalizing factors is negligible.

B.1.1 Multivariate Statistics

In this section, we will assume that the samples are k-dimensional: $\mathbf{x_1}, \ldots, \mathbf{x_n}$ are a series of k-dimensional samples, with $\mathbf{x_i} = (x_{i,1}, x_{i,2}, \ldots, x_{i,k})$.

Definition B.3. The **sample mean** is defined the same way as in the univariate case (see definition B.1): $\mu = \frac{1}{n}\sum_{i=1}^{n} \mathbf{x_i}$. For each dimension $1 \leq j \leq k$, we have $\mu_j = \frac{1}{n}\sum_{i=1}^{n} x_{i,j}$.

Definition B.4. The **sample covariance** between two dimensions j and k is defined as $\sigma_{jk} = \frac{1}{n}\sum_{i=1}^{n}(x_{i,j} - \mu_j)(x_{i,k} - \mu_k)$. When $j = k$, the formula gives the sample variance of the jth dimension.

Definition B.5. The **sample covariance matrix**, also referred to simply as the covariance matrix, is defined as

$$\Sigma = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{x_i} - \mu)(\mathbf{x_i} - \mu)^T \tag{B.1}$$

For k-dimensional samples, Σ is a $k \times k$ matrix. The diagonal entries correspond to the variance of each of the individual components, and the $\Sigma_{i,j}$ entry corresponds to the covariance between dimension i and dimension j.

From definition B.4, we see that covariance is a symmetric relation: The covariance between dimensions j and k is the same as the covariance between dimensions k and j. Since these quantities are represented by the $\Sigma_{j,k}$ and $\Sigma_{k,j}$ entries respectively in the covariance matrix, we immediately obtain the following property:

Property B.6. The covariance matrix is symmetric.

Definition B.7. Let $\mathbf{v}$ be a k-dimensional vector of unit magnitude. Suppose we project all the points $\mathbf{x_1}, \ldots, \mathbf{x_n}$ onto $\mathbf{v}$. We can replace each sample by a number that denotes its position along the line, corresponding to its distance from an arbitrary point. The **projection variance** is the variance of these values. Note that the arbitrary reference point, or the points $\mathbf{v}$ passes through does not matter, as the variance accounts for the squared deviation from the mean.

To derive a formula for the projection variance, we shall without loss of generality (as noted in the previous paragraph) assume that $\mathbf{v}$ passes through the origin and the reference point is the origin. Then the distance of each point's projection onto $\mathbf{v}$ from the origin is given by equation (A.1), with the reference point $\mathbf{p}$ equal to the origin. This gives the distance (the denominator in equation (A.1) is equal to 1 as $\mathbf{v}$ has unit magnitude)

$$t_i = \mathbf{x_i^T v}$$

Since projection is a linear operation, the mean of these values is given by

$$t_\mu = \mu^{\mathbf{T}}\mathbf{v}$$

where μ is the sample mean as given in definition B.3. Applying the formula given in definition B.2 for single variable variance, we obtain the formula

$$\begin{aligned}\frac{1}{n}\sum_{i=1}^{n}\left(\mathbf{x_i^T v} - \mu^{\mathbf{T}}\mathbf{v}\right)^2 = \frac{1}{n}\sum_{i=1}^{n}\left[(\mathbf{x_i}-\mu)^{\mathbf{T}}\mathbf{v}\right]^T\left[(\mathbf{x_i}-\mu)^{\mathbf{T}}\mathbf{v}\right] = \frac{1}{n}\sum_{i=1}^{n}\mathbf{v}^T(\mathbf{x_i}-\mu)(\mathbf{x_i}-\mu)^{\mathbf{T}}\mathbf{v} \\ = \mathbf{v^T\Sigma v} \qquad \text{(B.2)}\end{aligned}$$

following the formula for the covariance matrix given in equation (B.1). Since variance is always non-negative, and since $\mathbf{v}$ is arbitrary up to a scaling factor, we can conclude from property B.6 and definition A.19 the following:

Property B.8. The covariance matrix is positive semidefinite. Furthermore, if there is no direction in which the projection variance is zero, then the covariance matrix is positive definite.

B.1.2 Weighted Statistics

It is often the case that not every sample is assigned equal weight: We may wish to consider some samples more important than others. In this section, we remain within the multi-dimensional framework, but each sample is additionally given a weight w_i.

Definition B.9. The **equivalent number** of samples is given by $W = \sum_{i=1}^{n} w_i$.

We can define the mean and covariance matrix (and hence the covariance and variance) for a weighted set of samples as follows:

Definition B.10. The **mean** of a weighted set of samples is given by $\mu = \frac{1}{W} \sum_{i=1}^{n} w_i \mathbf{x_i}$.

Definition B.11. The **covariance matrix** of a weighted set of samples is given by

$$\mathbf{\Sigma} = \frac{1}{W} \sum_{i=1}^{n} w_i (\mathbf{x_i} - \mu)(\mathbf{x_i} - \mu)^T$$

with μ following definition B.10. As before, we can obtain the variance of any dimension or the covariance between two dimensions from the elements of the covariance matrix (variance from the diagonal elements, covariances from the off-diagonal elements).

B.2 The Normal (Gaussian) Distribution

The univariate Normal or Gaussian distribution is parameterized by two parameters: μ and σ^2, and has probability density

$$f(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

This probability density is shown in figure B-1 for a distribution with $\mu = 0$ and $\sigma^2 = 1$. The normal distribution has the following two important properties:

Property B.12. The normal distribution has mean μ and variance σ^2. Furthermore, the mean and the variance completely characterize a normal distribution: Two normal distributions with the same mean and variances are identical.

Property B.13. A normally distributed random variable that is scaled by a constant factor or has constants added remains normally distributed. Furthermore, any

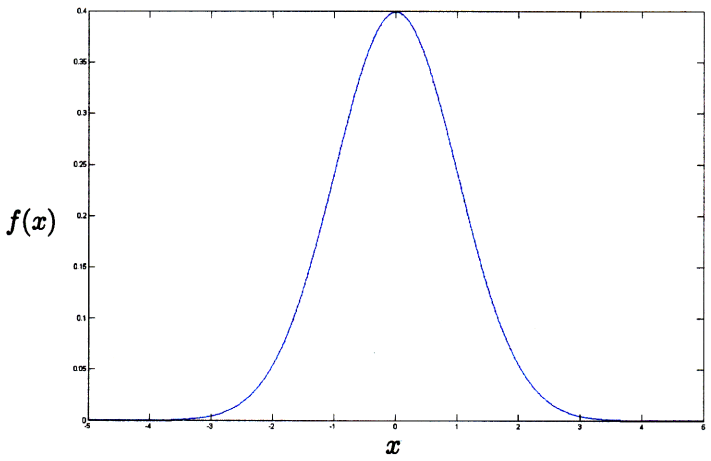


Figure B-1: An Example of a Univariate Normal Distribution.

linear combination of independent normally distributed random variables is normally distributed.

The normal distribution is important in statistics because of a Central Limit Theorem: Put informally, it states that if samples are drawn independently and identically from a probability distribution with finite variance[2], the sample mean converges in distribution to a normal distribution. Therefore, for most purposes, if the number of samples drawn is sufficiently large, we may assume that the distribution of the sample mean is normal.

Definition B.14. The **standard normal distribution** is the normal distribution with mean $\mu = 0$ and variance $\sigma^2 = 1$.

The standard normal distribution has been widely studied and its cumulative distribution – for which no closed form formula exists – has been tallied in tables. For example, it is well known that there is roughly a 95% probability that a value sampled from the standard normal distribution lies in the range $[-1.96, 1.96]$. As such, it is useful to transform any given normal distribution into the standard normal distribution. This can be done using the following property:

[2]The assumption of finite variance is necessary. For example, the Cauchy distribution, which has an infinite variance, does not obey the Central Limit Theorem: The mean of samples drawn from identical Cauchy distributions is distributed according to that same Cauchy distribution.

Property B.15. If a random variable X is normally distributed with mean μ and variance σ^2, then the random variable $\frac{X-\mu}{\sigma}$ follows the standard normal distribution.

This is easily shown to be true: Following property B.13, the random variable $\frac{X-\mu}{\sigma}$ is also distributed normally. It can easily be verified that the mean of this random variable is 0 and its variance is 1. The result then follows from property B.12.

B.2.1 The Multivariate Gaussian

We now consider the Normal distribution in multiple dimensions. The k dimensional *multivariate normal distribution* is parameterized by a $k \times 1$ vector μ and a $k \times k$ symmetric positive semidefinite matrix Σ, and has probability density

$$f(\mathbf{x}; \mu, \Sigma) = \frac{1}{(2\pi)^{k/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu)\right)$$

Figure B-2 shows the probability density function of a two dimensional multivariate normal distribution with $\mu = [0 \quad 0]^T$ and $\Sigma = \begin{bmatrix} 3 & -1 \\ -1 & 4 \end{bmatrix}$. Like the univariate

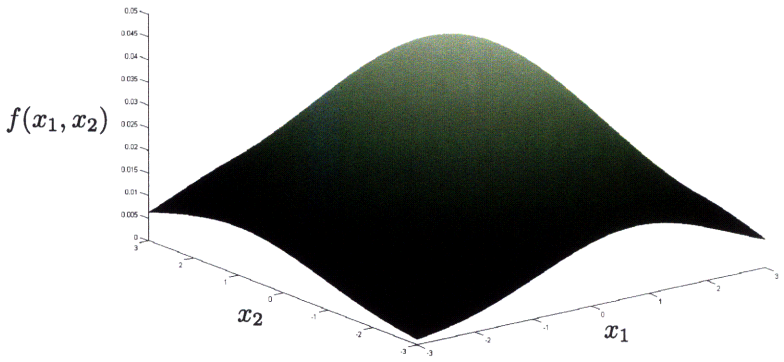


Figure B-2: An Example of a Multivariate Normal Distribution.

normal distribution, the multivariate normal distribution has several important properties. We will first state the analogues of properties B.12 and B.13:

Property B.16. The multivariate normal distribution has mean μ and covariance matrix Σ. Furthermore, the mean and covariance matrix completely characterize

the multivariate normal distribution: Two multivariate normal distributions with the same means and covariance matrices are identical.

Property B.17. If the k-dimensional random variable X is distributed multivariate normal, then the random variable $\mathbf{c} + \mathbf{A}X$, where $\mathbf{c}$ is a $n \times 1$ constant vector and $\mathbf{A}$ a $n \times k$ constant matrix, is also distributed multivariate normal.

We now define the analog of definition B.14:

Definition B.18. The **standard multivariate normal distribution in k dimensions** is the multivariate normal distribution with mean $\mu = \mathbf{0}$ and covariance matrix $\Sigma = I_k$, the $k \times k$ identity matrix.

Unlike the univariate standard normal, there have been less tables computed for the multivariate standard normals; however, it has been equally well-studied and there are known efficient computations for it. For example, we might compute that a random variable sampled from the 3-dimensional standard multivariate normal distribution is likely to be in a sphere of radius 2.8 around the origin approximately 95% of the time. Hence, it is also useful to convert a distribution to the standard multivariate normal distribution. This procedure is known as *whitening.*

Whitening is only possible if Σ is strictly positive definite rather than simply positive semidefinite; Σ being only positive semidefinite is similar to $\sigma^2 = 0$ in the univariate case, in which case the procedure outlined in property B.15 will fail as well. Hence, we will assume that Σ is strictly positive definite, in which case one possible whitening procedure is as follows:

Property B.19. Suppose X is a multivariate normal random variable with mean μ and covariance matrix Σ. Since Σ is symmetric positive definite, we can perform the Eigenvalue Decomposition outlined in section A.3.1: $\Sigma = \mathbf{QSQ^T}$, where $\mathbf{Q}$ is an orthogonal matrix with the eigenvectors of Σ as its columns and $\mathbf{S}$ is a diagonal matrix with the eigenvalues of Σ along its diagonal. Let $\mathbf{S^{-1/2}}$ be a diagonal matrix where every diagonal element is the corresponding element of $\mathbf{S}$ taken to the $-1/2$

power. Then $\mathbf{S^{-1/2}Q}^T(X-\mu)$ is distributed according to the standard multivariate normal distribution.

We can verify this: Following property B.17, the random variable $\mathbf{S^{-1/2}Q}^T(X-\mu)$ is distributed according to a multivariate normal distribution. It is easily checked that it has mean $\mathbf{0}$ and it has covariance matrix equal to

$$\begin{aligned}\left[\mathbf{S^{-1/2}Q}^T(X-\mu)\right]\left[\mathbf{S^{-1/2}Q}^T(X-\mu)\right]^T &= \mathbf{S^{-1/2}Q}^T(X-\mu)(X-\mu)^T\mathbf{QS^{-1/2}} \\ &= \mathbf{S^{-1/2}Q}^T\Sigma\mathbf{QS^{-1/2}} \\ &= \mathbf{S^{-1/2}Q}^T\mathbf{QSQ}^T\mathbf{QS^{-1/2}} \\ &= I\end{aligned}$$

where in the first equality we note that $S^{-1/2}$ is symmetric and is thus equal to its transpose, and in the last equality we have used the fact that $\mathbf{Q}^T\mathbf{Q} = \mathbf{S^{-1/2}SS^{-1/2}} = I$. Since the means and covariance matrices are identical, the result follows from property B.16.

Remark B.20. It is usually convenient to visualize the multivariate normal distribution as an ellipse, which is a reflection of the two-dimensional case but remains a useful tool for obtaining intuition about the general multivariate normal distribution. In the two-dimensional case, the axes of the ellipse point in the direction of the eigenvectors of the covariance matrix, and the length of the axes is proportional to the magnitude of their respective eigenvalues. Why the eigenvectors and eigenvalues? We will obtain a clearer understanding in Appendix C.

B.3 Testing if a Single Sample belongs to a Given Normal Distribution

In this thesis, we will encounter the following problem: Given a normal distribution parameterized by μ and Σ, and a sample measurement $\mathbf{x}$, we wish to test if $\mathbf{x}$ came

from this distribution. This is one example from a very large class of problems known as *hypothesis testing.*

We present a very basic overview of hypothesis testing with regards to this context. In hypothesis testing, we have an assumed hypothesis we wish to test, known as the **null hypothesis**. In our context, our assumed hypothesis is that $\mathbf{x}$ comes from the given distribution. We test the null hypothesis against an **alternate hypothesis**, which in our context is that $\mathbf{x}$ does not come from the given distribution. We shall refer to the null hypothesis as H_0 and the alternate hypothesis as H_A.

Our aim is to determine if we will reject or not reject the null hypothesis[3]. The hypothesis testing framework proceeds as follows (here we assume that $\mathbf{x} \in \mathcal{R}^k$ and is thus in k-dimensional space):

1. Select a region $C \subset \mathcal{R}^k$ which is termed the **critical region**.

2. If $\mathbf{x} \in C$, reject the null hypothesis. Otherwise, do not reject the null hypothesis.

There are generally two types of errors with any such statistical test:

- **Type 1 error**. This is an error in which we reject the null hypothesis when it is true. In our context, a type 1 error occurs when we claim that $\mathbf{x}$ does not come from the distribution when it actually does so.

- **Type 2 error**. This is an error in which we do not reject the null hypothesis when it is incorrect. In our context, a type 1 error occurs when we claim that $\mathbf{x}$ comes from the distribution when it does not.

Generally, there is no way to minimize both type of errors simultaneously; we can reduce one at the expense of the other: For example, we may minimize type 1 error by choosing C as the empty set, but this maximizes type 2 error. One approach used

[3]In classical hypothesis testing, we never *accept* the null hypothesis, but rather we do not reject it. The null hypothesis is assumed to be true, but we can never actually determine if it is actually correct; rather we can only determine if we should reject it because of statistical reasons.

is to fix the probability of type 1 error allowed and find a test that minimizes the probability of type 2 error, subject to the constraint that the probability of type 1 error cannot exceed the fixed amount. We denote the fixed probability of type 1 error as α. To ensure that the probability of type 1 error is α, we need the region C to have probability α under the normal distribution with mean μ and covariance Σ – so that if the null hypothesis was true, that $\mathbf{x}$ was really drawn from such a distribution, there would only be an α probability of incorrectly rejecting the null hypothesis.

There exists many possible regions C with the property we are considering; however, we wish to choose the region that minimizes type 2 error. We will note that the two hypotheses we are distinguishing between – $\mathbf{x}$ coming from the given distribution and not – are considered *simple point* hypotheses. In this case, the celebrated Neyman-Pearson lemma (see [13]) tells us how to construct C: Consider the statistic

$$LR(\mathbf{x}) = \frac{P_{H_0}(\mathbf{x})}{P_{H_A}(\mathbf{x})} \tag{B.3}$$

where $P_{H_0}(\mathbf{x})$ is the probability of sampling $\mathbf{x}$ under the null hypothesis – in our context, the probability of obtaining $\mathbf{x}$ from the given distribution – and $P_{H_A}(\mathbf{x})$ is the probability of sampling $\mathbf{x}$ if it does not come from the given distribution. The critical region C – the region in which we reject the null hypothesis – consists of the region where $LR(\mathbf{x}) < k$, for the correct value of k such that the probability of C under the given distribution is α. The statistic LR is known as a **Likelihood Ratio Statistic** and the test we have described is an example of a **Likelihood Ratio Test**.

In our context, we will assume that if H_A is true, then $\mathbf{x}$ could have been sampled uniformly everywhere, and therefore $P_{H_A}(\mathbf{x})$ is a constant for all $\mathbf{x}$. Similarly, if H_0 is true, then the probability of sampling $\mathbf{x}$ is simply the probability of $\mathbf{x}$ under the given distribution, and therefore $P_{H_0}(\mathbf{x}) = f(\mathbf{x}; \mu, \Sigma)$. Hence, the likelihood ratio test

simply becomes the following test:

Reject H_0 iff	$f(\mathbf{x}; \mu, \Sigma) < k$
Do not reject H_0 iff	$f(\mathbf{x}; \mu, \Sigma) \geq k$

where we have to determine k such that the critical region of rejection has probability α under the normal distribution. Now consider the standard normal distributions (see definitions B.14 and B.18). Since these distributions are spherically symmetrical and decay exponentially away from the origin, the above test under these distributions is equivalent to choosing a radius r and setting the critical region to be all points further than r from the origin – in such a way that the critical region has a probability of α under the distribution we are considering. We illustrate this with two examples that are relevant to this thesis; in both cases, we have $\alpha = 0.05$, and we will assume standard normal distributions, since it is always possible to convert an arbitrary normal distribution to these standard normal distributions (see properties B.15 and B.19):

Univariate standard normal distribution. In this case, we choose $r = 1.96$ as it is well known that the region $[-1.96, 1.96]$ has probability 0.95 under the standard normal distribution; thus the critical region, with probability $\alpha = 0.05$, is $(-\infty, -1.96) \cup (1.96, \infty)$.

Remark B.21. Since $1.96 \approx 2$, a general rule of thumb is to reject the null hypothesis for a general (not necessarily standard) univariate normal distribution if $\mathbf{x}$ is two standard deviations or more away from the mean.

3-dimensional standard normal distribution. The sphere with radius 2.8 around the origin has probability approximately 0.95 under the standard 3-dimensional standard normal distribution; hence $r = 2.8$ and the critical region is the area outside the sphere with this radius.

B.4 Maximum Likelihood Estimation

In many cases, we model data using a distribution – such as the multivariate normal distribution – but do not have the parameters (μ, Σ in the multivariate normal case) to complete its specification. In this case, we will have to perform **parameter estimation** using the data. One popular estimation technique is Maximum Likelihood Estimation.

The main idea of Maximum Likelihood Estimation is to find the parameter settings that **maximize the likelihood** of the observed data. The likelihood is simply the probability of observing the given data. For example, for a multivariate normal distribution with fixed Σ and observed data $\mathbf{x}$, the likelihood as a function of the unknown parameter μ is simply

$$l(\mu; \mathbf{x}, \Sigma) = f(\mathbf{x}; \mu, \Sigma)$$

In practice, it is often more convenient to maximize the *log likelihood* rather than the likelihood. Let $L(\mu; \mathbf{x}, \Sigma) = \log l(\mu; \mathbf{x}, \Sigma)$; following our example with data $\mathbf{x}$, fixed Σ and unknown μ, the **maximum likelihood estimate** of μ is given by

$$\hat{\mu}_{ML} = \arg\max_{\mu} L(\mathbf{x}; \mu, \Sigma) \tag{B.4}$$

Not surprisingly, in the multivariate Gaussian case, it will turn out that the maximum likelihood estimate of μ from a given data sample will be the sample mean. Maximum Likelihood estimation is popular for several reasons: In a practical sense, even when no closed form solution for (B.4) exists, there usually exist fairly efficient numerical methods to obtain the estimate. In a theoretical sense, maximum likelihood estimators exhibit a variety of desirable properties, which we will briefly mention informally (although many of these properties require far more mathematical rigor – beyond the scope of this thesis – to define and specify precisely):

1. An **unbiased estimator** is an estimator whose expected value is the value of

the parameter itself. An **efficient unbiased** estimator is the unbiased estimator that has a lower variance than all other unbiased estimators *regardless of the value of the actual parameter.* Note that there may not exist an efficient unbiased estimator. However, if there exists one, that estimator is identical to the Maximum Likelihood estimator[4]. We note that in the Gaussian case an efficient unbiased estimator exists and thus the Maximum Likelihood estimator is efficient.

2. The maximum likelihood estimator is *usually* consistent; by consistency we mean that the maximum likelihood estimate converges[5] in probability to the actual value of the parameter as the number of samples increase.

3. Under certain conditions that are usually met in practice, the maximum likelihood estimator exhibits several fairly desirable asymptotic properties, such as asymptotic unbiasedness and asymptotic efficiency.

B.5 Bayes' Law and Maximum A Posteriori Estimation

In many cases, when performing estimation, we have some prior knowledge or beliefs about the parameter we are trying to estimate, and we would like to incorporate these beliefs into our estimation framework. The key instrument we will use is Bayes' law. Let θ denote the set of parameters we are trying to estimate and $\mathbf{x}$ the data we observe. We have a prior belief $P(\theta)$ on the parameters, and the probability of observing the data we observed is $P(\mathbf{x})$. Let $P(\theta|\mathbf{x})$ denote the updated probabilities – incorporating our beliefs – on the parameters after observing $\mathbf{x}$. Bayes' law tells us

[4]More precisely, if there exists an estimator that satisfies the famous Cramer-Rao bound, that estimator is the Maximum Likelihood estimator. See [30] for details.

[5]We will not discuss the different kinds of probabilistic convergence here; this is a fairly advanced topic that requires some understanding of measure theory.

that

$$P(\theta|\mathbf{x}) = \frac{P(\mathbf{x}|\theta)P(\theta)}{P(\mathbf{x})} \tag{B.5}$$

The value $P(\mathbf{x}|\theta)$ denotes the probability of observing $\mathbf{x}$ under a given set of parameters. For example, if our model is the multivariate normal, and $\theta = (\mu, \Sigma)$, then $P(\mathbf{x}|\theta) = f(\mathbf{x}; \mu, \Sigma)$. Note that the term $P(\mathbf{x})$ in the denominator of (B.5) accounts for the probability of observing $\mathbf{x}$ weighted over *all* possible values of θ; in other words $P(\mathbf{x}) = \int_\theta P(\mathbf{x}|\theta)P(\theta)d\theta$.

We now wish to estimate the parameters θ given our prior beliefs and observed data. The technique we introduce here is **Maximum a Posteriori (MAP)** estimation, which is extremely similar in spirit to Maximum Likelihood Estimation, which was introduced in section B.4. In essence, the key idea is to find θ that maximizes the right hand side of equation (B.5).

As with Maximum Likelihood Estimation, it is often convenient to take logarithms and maximize the log a posteriori likelihood. Furthermore, it is usually the case that we may ignore the $P(\mathbf{x})$ term; since $\mathbf{x}$ is observed and is constant regardless of the value of θ chosen, $P(\mathbf{x})$ often plays no role in the estimation process. With these modifications, and using $L(.)$ to denote $\log P(.)$, the MAP estimate of θ is then given by

$$\hat{\theta}_{MAP} = \arg\max_\theta L(\mathbf{x}|\theta) + L(\theta) \tag{B.6}$$

If we have no prior on θ – or if we have a uniform prior on θ – such that $L(\theta)$ is a constant across all possible θ, then the MAP estimate and the Maximum Likelihood estimate are identical.

Remark B.22. The MAP estimate takes the **mode** of the posterior probability given

by equation (B.5). Another common Bayesian estimator, the **Bayes Least Squares** estimator, uses the **mean** of this posterior probability as its estimate of θ.

B.6 Markov Random Fields

Occasionally we have a collection of variables that are related in a structured manner, and we wish to perform inference by taking advantage of this structure. One such class of structured models are the **Graphical Models**. Within this class of models, we will consider only one particular model, the **Markov Undirected Graphical Models**, or **Markov Random Fields (MRF)**.

In this model, every variable can be represented as a node in an undirected graph. Edges denote dependencies between variables; an example of a nine-node MRF can be found in figure B-3.

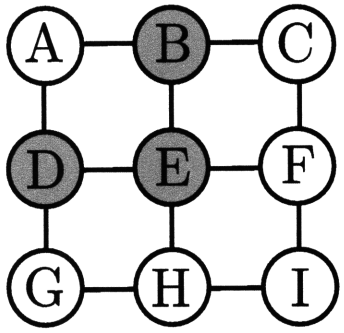


Figure B-3: A Markov Random Field.

The key feature of such a model is the **Markov** property: Given a set of nodes S whose removal partitions the graph into two separate components, the variables on these two components are independent given the values of the variables in the set S. A special case is when one component consists only of a single node; the set S is then denoted as the **Markov blanket** of the node. As an example, in figure B-3, the Markov blanket of node A consists of the shaded nodes B, D, and E; hence variable A is independent of all other variables in the graph conditioned on the values of

variables B, D and E.

In this graph, a **maximal clique** is a set of nodes that has the following two properties:

1. The nodes are all pairwise directly connected to each other.

2. No additional node can be added to this set while maintaining the first property.

In the MRF in figure B-3, the set of maximal cliques are every pair of adjacent vertices. The Hammersley-Clifford theorem (see [23]) states that if the Markov property holds and every assignment of values to variables has positive probability, then the probability distribution over the variables can be factored into potential functions over the maximal cliques. For example, using x_A to represent the value assigned to variable A, and so on, and using Ψ_{x_A,x_B} to represent the potential function over the pair of variables (A, B), we may factorize the probability density of the MRF in figure B-3 as

$$P(x_A, \ldots, x_I) \propto \prod_{(i,j) \text{ connected by an edge}} \Psi_{x_i,x_j}(x_i, x_j) \tag{B.7}$$

In this thesis, we will consider a special case: Regardless of the size of the cliques, we will assume that the probability distribution can be factored into potential functions that only include two variables which share an edge[6]. This condition automatically holds true for any graph with maximal cliques of size at most two; this includes, for example, the MRF in figure B-3 and trees – graphs without cycles.

We will also include potential functions for individual nodes e.g. $\Psi_{x_A}(x_A)$ is the potential function for node A. This is clearly not necessary – since these potential functions can always be incorporated into the pairwise potential functions – but it allows us to classify the potential functions into two categories:

[6]This will simplify our discussion of Belief Propagation in the next section; however, in most cases, by converting graphs that contain potential functions over more than two variables into factor graphs, Belief Propagation will also be applicable.

1. **Data term.** These are the individual potential functions which allow us to specify some prior beliefs on the probability of each value the variables can take.

2. **Connectivity term.** These are the pairwise potential functions which allow us to specify how neighboring variables can affect each other.

With this change, equation (B.7) now becomes:

$$P(x_A, \ldots, x_I) \propto \prod_{\text{All nodes } i} \Psi_{x_i}(x_i) \prod_{(i,j) \text{ connected by an edge}} \Psi_{x_i,x_j}(x_i, x_j) \tag{B.8}$$

B.6.1 Estimation using Belief Propagation Algorithms

We wish to perform estimation on our graphical model to solve the following problem: Find

$$(x_A, \ldots,) = \arg\max_{x_A,\ldots} P(x_A, \ldots)$$

That is, find the assignment of values to variables that has maximum probability. In this section, we will briefly introduce the **Belief Propagation** technique that solves the above problem exactly for graphs without cycles and can serve as an approximation for graphs with cycles. We will need the set of possible values taken by each variable to be discrete and have finite size[7].

Belief Propagation relies on *message passing* between adjacent nodes connected by an edge; intuitively, a node receives messages from its Markov blanket informing it about the nodes beyond it, and due to the Markov property it does not require further information. To build some intuition as we introduce Belief Propagation, we shall look at a simple three-node example, shown in figure B-4.

[7]Approximations such as Particle Filtering are available for the continuous case; however, we will not encounter them here.

Figure B-4: A Simple Three-Node MRF.

The probability factorization of this MRF is as follows:

$$P(x_A, x_B, x_C) = \Psi_{x_A}(x_A)\Psi_{x_B}(x_B)\Psi_{x_C}(x_C)\Psi_{x_A,x_B}(x_A, x_B)\Psi_{x_B,x_C}(x_B, x_C)$$

and thus our maximization problem can be rewritten as

$$\begin{aligned}\arg\max_{x_A,x_B,x_C} &\Psi_{x_A}(x_A)\Psi_{x_B}(x_B)\Psi_{x_C}(x_C)\Psi_{x_A,x_B}(x_A, x_B)\Psi_{x_B,x_C}(x_B, x_C) = \\ &\arg\max_{x_A} \Psi_{x_A}(x_A)\Big(\arg\max_{x_B} \Psi_{x_B}(x_B)\Psi_{x_A,x_B}(x_A, x_B) \\ &\qquad \Big[\arg\max_{x_C} \Psi_{x_C}(x_C)\Psi_{x_B,x_C}(x_B, x_C)\Big]\Big) \end{aligned} \tag{B.9}$$

where the term in the square brackets [] is a function of x_B and the term in the round brackets () is a function of x_A. This suggests the following message passing schedule:

1. In the first round, for each possible value of x_B, node C sends node B the message $m_{CB}(x_B) = \max_{x_C} \Psi_{x_C}(x_C)\Psi_{x_B,x_C}(x_B, x_C)$.

2. In the second round, for each possible value of x_A, node B sends node A the message $m_{BA}(x_A) = \max_{x_B} \Psi_{x_B}(x_B)\Psi_{x_A,x_B}(x_A, x_B)m_{CB}(x_B)$. Note that node B uses the messages that C passed to it initially.

3. Finally, node A takes $x_A = \arg\max_{x_A} \Psi_{x_A}(x_A)m_{BA}(x_A)$.

4. By sending the value of x_A to node B, node B can figure out $x_B = \arg\max_{x_B} \Psi_{x_B}(x_B)\Psi_{x_A,x_B}(x_A, x_B)m_{CB}(x_B)$, and similarly by sending the value of x_B to node C, the value of x_C can be figured out.

The above message passing schedule relies on a given message passing order. However, it is certainly possible for messages to be passed *simultaneously.* For example, nodes C and A could pass messages to node B at the same time. The only catch is that

when node B passes a message to node A, it should "forget" the message that node A previously passed to it. This suggests the following general simultaneous message passing algorithm for an acyclic MRF:

Algorithm B.23. The Belief Propagation Algorithm for Maxima Estimation. Let $m_{AB}(x_B)$ denote a message passed from node A to node B for a given value of x_B, and let N_A denote the set of neighbors of node A. We perform the following steps:

1. Initially, every node A passes to each of its neighbors $B \in N_A$ the message $m_{AB}(x_B) = 1$ for every possible value of x_B.

2. At every time step, every node A passes to each of its neighbors $B \in N_A$ the following message for every possible value of x_B:

$$m_{AB}(x_B) = \max_{x_A} \Psi_{x_A}(x_A)\Psi_{x_A,x_B}(x_A, x_B) \prod_{C \in N_A \setminus B} m'_{CA}(x_A)$$

   Here $m'_{CA}(x_A)$ is the message that node C sent node A in the *previous time step*. Thus, the message that A sends B takes into account the messages that A received from all its neighbors – except for B – in the previous time step.

3. After a sufficient number of iterations (one less than the diameter of the graph), each node A calculates

$$x_A = \arg\max_{x_A} \Psi_{x_A}(x_A) \prod_{C \in N_A} m_{CA}(x_A)$$

A quick word of warning: The final step of algorithm B.23 is not entirely accurate if there are multiple values of x_A that maximize $\Psi_{x_A}(x_A) \prod_{C \in N_A} m_{CA}(x_A)$; in this case, a directed propagation mechanism similar to the final step of our example three-node message passing schedule will work. However, in most cases algorithm B.23 will be accurate.

Algorithm B.23 will give an exact solution when the graph is acyclic; in practice, it can be a good approximation to apply it to graphs *with cycles*; this is referred to as **loopy belief propagation**. This no longer gives an exact answer, and in some cases can result in wildly incorrect estimates, but in most cases it gives a fairly satisfactory answer, and will be adequate for our purposes in this thesis.

Remark B.24. In practice, the messages that are passed can become extremely small, so to prevent computational error due to numerical issues, it is common to normalize the messages after each iteration: For every neighboring (A, B) node pair, set $m_{AB}(x_B) = \frac{m_{AB}(x_B)}{\sum_{x_B} m_{AB}(x_B)}$.

Appendix C

Principal Component Analysis

This section will briefly discuss the technique of **Principal Component Analysis (PCA)**, also known as the Karhunen-Loeve transform. A detailed description can be found in [15].

The primary motivation of PCA is of dimensionality reduction. Suppose we receive data that is d-dimensional in nature – perhaps each sample could be the age, height, weight, etc. of a person – and we want to perform some data analysis on it. If d is large, analysis might take too long, and we are interested in reducing the dimensionality of the data while still retaining its useful information so that our analysis will remain accurate.

Furthermore, perhaps some data is simply noise and conveys no useful information; for example, if in our data set the age variable is always the same value, we should not need it in our analysis. Perhaps the ratio of height to weight is extremely similar for every person in our data set – in this case we should merge the two variables into one variable.

These motivations are illustrated in figure C-1. Although the data points are two-dimensional, they are mainly scattered along the line with unit slope; it would perhaps be appropriate to project all the points onto this line and take their distances along

the line as their value – this reduces the dimensionality of the data from two to one.

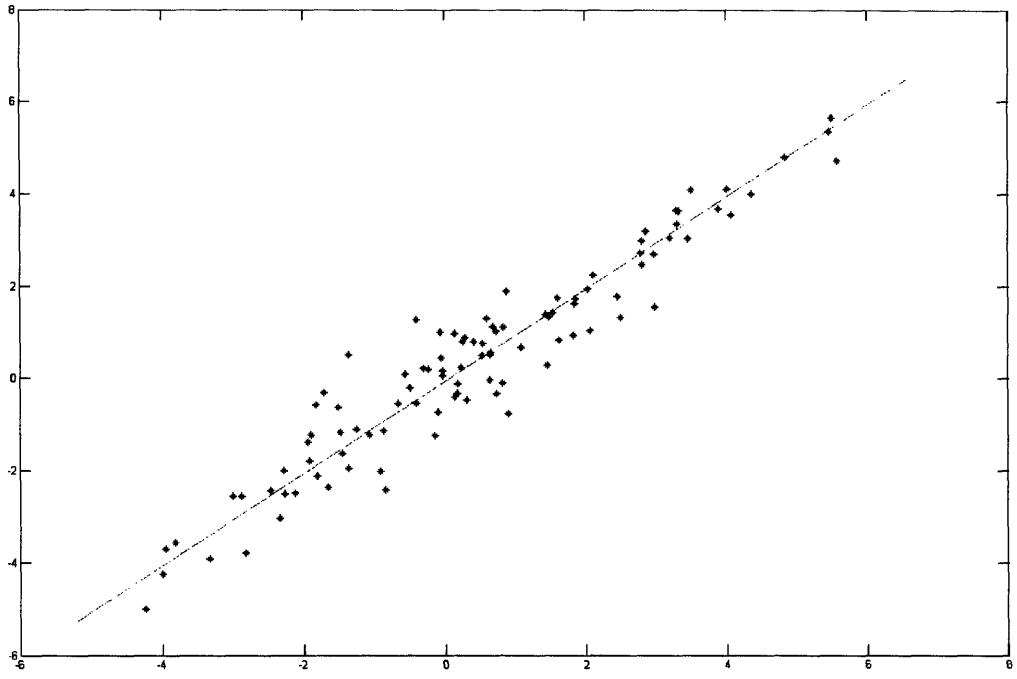


Figure C-1: A motivation for PCA.

C.1 Approach

PCA tries to find the directions that maximize the variance of the data; in figure C-1, this direction is the line with unit slope. One way to view PCA is as follows:

1. Initially, no directions have been found.

2. Find the direction that maximizes the variance of the data, subject to the condition that the new direction found is orthogonal to all previous directions found.

3. Repeat the previous step until we have found enough directions (if we want to reduce the data to d dimensions, we find d directions), or the variance in the directions found is too small.

4. Project each point onto every direction found. The distances along the projections correspond to the new values for the data; in this way the dimensionality of the data is reduced to the number of directions found.

However, it turns out that we can find all the relevant directions simultaneously; the techniques required are the projection variance discussion from definition B.7 and the Rayleigh quotient discussion from appendix A.3.2.

From the projection variance discussion, we know that the variance in the direction denoted by the unit vector $\mathbf{v}$ is given by $\mathbf{v^T \Sigma v}$, where Σ is the sample covariance matrix of our data.

From the Rayleigh quotient discussion, we know the following: The direction $\mathbf{v}$ that maximizes $\mathbf{v^T \Sigma v}$ is the normalized eigenvector corresponding to the largest eigenvalue of Σ; furthermore, to find the next direction orthogonal to all previous directions found, take the normalized eigenvector corresponding to the largest eigenvalue of Σ not already previously taken.

To determine the variance in the direction found, we can recalculate $\mathbf{v^T \Sigma v}$ for each vector $\mathbf{v}$ found, or using the fact that $\mathbf{v}$ is an eigenvector of Σ with corresponding eigenvalue λ, we have

$$\mathbf{v^T \Sigma v} = \lambda \mathbf{v^T v} = \lambda$$

since $\mathbf{v}$ is a vector of unit magnitude. Therefore, the variance in the direction of the eigenvector $\mathbf{v}$ of Σ is simply its corresponding eigenvalue. Since the directions (eigenvectors) found are orthogonal to each other, the total variance of the data is the sum of all the eigenvalues.

We summarize our discussion as follows:

- To perform PCA, calculate the eigenvalues and corresponding eigenvectors of

the covariance matrix. Sort the eigenvalues in decreasing order and take the eigenvectors corresponding to the d largest eigenvalues, where d is the desired dimensionality of the resulting data.

- The variance in the direction of a given eigenvector is its corresponding eigenvalue.

- The total variance of the data is the sum of all eigenvalues of the sample covariance matrix Σ.

Bibliography

[1] T. Asano, D. Chen, N. Katoh, and T. Tokuyama. Polynomial-time Solutions to Image Segmentation. In *Proc. 7th Annual SIAM-ACM Conference on Discrete Algorithms*, pages 104–113, 1996.

[2] X. Bai and G. Sapiro. A Geodesic Framework for Fast Interactive Image and Video Segmentation and Matting. In *Proc. ICCV 2007*, Oct 2007.

[3] M. Bertalmio, G. Sapiro, V. Caselles, and C. Ballester. Image Inpainting. In *Proc. SIGGRAPH 2000*, pages 417–424, 2000.

[4] D. Biedny, B. Monroy, and N. Moody. *Photoshop Channel Corps.* New Riders Publishing, 1998.

[5] Y. Boykov and M. Jolly. Interactive Graph Cuts for Optimal Boundary and Region Segmentation of Objects in N-D Images. In *Proc. ICCV 2001*, volume 1, pages 105–112, 2001.

[6] R. Brinkmann. *The Art and Science of Digital Compositing.* Academic Press, San Diego, CA, 1999.

[7] S. Buss. *3D Computer Graphics: A Mathematical Introduction with OpenGL.* Cambridge University Press, 2003.

[8] Y. Y. Chuang. *New Models and Methods for Matting and Compositing.* Ph.D thesis, University of Washington, 2004.

[9] Y. Y. Chuang, B. Curless, D. H. Salesin, and R. Szeliski. A Bayesian Approach to Digital Matting. In *Proc. CVPR 2001*, 2001.

[10] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms.* The MIT Press, 2nd edition, 2001.

[11] A. Criminisi, P. Perez, and K. Toyama. Object Removal by Exemplar-Based Inpainting. In *Proc. IEEE CVPR 2003.*

[12] A. Efros and T. Leung. Texture Synthesis by Non-parametric Sampling. In *Proc. IEEE ICCV 1999*, pages 1033–1038, September 1999.

[13] N. Giri. *Introduction to Probability and Statistics.* Marcel Dekker, New York, NY, 2nd edition, 1993.

[14] L. Grady. Random Walks for Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(11):1768–1783, November 2006.

[15] I. Jolliffe. *Principal Component Analysis.* Springer, 2nd edition, 2002.

[16] A. Levin, D. Lischinski, and Y. Weiss. A Closed Form Solution to Natural Image Matting. In *Proc. IEEE CVPR 2006*, pages 61–68, 2006.

[17] A. Levin, A. Racha, and D. Lischinski. Spectral Matting. In *Proc. IEEE CVPR 2007*, June 2007.

[18] D. Mackay. *Information Theory, Inference and Learning Algorithms.* Cambridge University Press, 2002.

[19] D. Martin, C. Fowlkes, D. Tal, and J. Malik. A Database of Human Segmented Natural Images and its Application to Evaluating Segmentation Algorithms and Measuring Ecological Statistics. In *Proc. ICCV 2001*, volume 2, pages 416–423, July 2001.

[20] R. Nickalls. A new approach to solving the cubic: Cardan's solution revealed. *The Mathematical Gazette*, 77:354–359, 1993.

[21] I. Omer and M. Werman. Color Lines: Image Specific Color Representation. In *Proc. IEEE CVPR 2004*, pages 946–953, June 2004.

[22] M. Orchard and C Bouman. Color Quantization of Images. *IEEE Transactions on Signal Processing*, 39(12):2677–2690, December 1991.

[23] J. Pearl. *Probabilistic Reasoning in Intelligent Systems.* Morgan Kaufmann, San Francisco, CA, 1988.

[24] P. Perona and J. Malik. Scale-Space and Edge Detection Using Anisotropic Diffusion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(7):629–639, July 1990.

[25] T. Porter and T. Duff. Compositing Digital Images. In *Proc. 11th Annual Conference on Computer Graphics and Interactive Techniques*, pages 253–259, 1984.

[26] C. Poynton. *Digital Video and HDTV Algorithms and Interfaces.* Morgan Kaufmann, San Francisco, CA, 2003.

[27] C. Rother, V. Kolmogorov, and A. Blake. GrabCut: Interactive Foreground Extraction using Iterated Graph Cuts. In *Proc. SIGGRAPH 2004*, volume 23, pages 309–314, 2004.

[28] M. Ruzon and C. Tomasi. Alpha Estimation in Natural Images. In *Proc. IEEE CVPR*, volume 1, pages 18–25, Hilton Head Island, SC, June 2000.

[29] P. Sahoo, S. Soltani, A. Wong, and Y. Chen. A survey of thresholding techniques. *Computer Vision, Graphics and Image Processing*, 41(2):233–260, February 1988.

[30] J. Shao. *Mathematical Statistics.* Springer, 2nd edition, 2007.

[31] J. Shi and J. Malik. Normalized Cuts and Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, August 2000.

[32] A. Smith and J. Blinn. Blue Screen Matting. In *Proc. SIGGRAPH 1996*, 1996.

[33] G. Strang. *Linear Algebra and Its Applications.* Brooks Cole, 4th edition, 2005.

[34] J. Wang and M. Cohen. An Iterative Optimization Approach for Unified Image Segmentation and Matting. In *Proc. IEEE ICCV 2005*, pages 936–943, 2005.

[35] J. Wang and M. Cohen. Optimized Color Sampling for Robust Matting. In *Proc. IEEE CVPR 2007*, June 2007.

[36] D. Ziou and S. Tabbone. Edge Detection Techniques – An Overview. *International Journal of Pattern Recognition and Image Analysis*, 8:537–559, 1998.