1-1-2009

Image segmentation through the scale-space random walker

Richard Rzeszutek
Ryerson University

IMAGE SEGMENTATION THROUGH THE SCALE-SPACE RANDOM WALKER

by

Richard Rzeszutek, B.Eng
Bachelors of Engineering (B.Eng), Ryerson University, 2007

A thesis
presented to Ryerson University
in partial fulfillment of the
requirement for the degree of
Masters of Applied Science
in the Program of
Electrical and Computer Engineering.

Toronto, Ontario, Canada, 2009

Author's Declaration

I hereby declare that I am the sole author of this thesis.
I authorize Ryerson University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize Ryerson University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Instructions on Borrowers

Ryerson University requires the signatures of all persons using or photocopying this thesis. Please sign below, and give address and date.

Abstract

Image Segmentation Through the Scale-Space Random Walker

Richard Rzeszutek

Masters of Applied Science, Electrical and Computer Engineering

Ryerson University, Toronto, Ontario, Canada, 2009

This thesis proposes an extension to the Random Walks assisted segmentation algorithm that allows it to operate on a scale-space. Scale-space is a multi-resolution signal analysis method that retains all of the structures in an image through progressive blurring with a Gaussian kernel. The input of the algorithm is setup so that Random Walks will operate on the scale-space, rather than the image itself. The result is that the finer scales retain the detail in the image and the coarser scales filter out the noise. This augmented algorithm is referred to as "Scale-Space Random Walks" (SSRW) and it is shown in both artificial and natural images to be superior to Random Walks when an image has been corrupted by noise. It is also shown that SSRW can improve the segmentation when texture, such as the artificial edges created by JPEG compression, has made the segmentation boundary less accurate.

This thesis also presents a practical application of the SSRW in an assisted rotoscoping tool. The tool is implemented as a plugin for a popular commercial compositing application that leverages the power of a Graphics Processing Unit (GPU) to improve the algorithm's performance so that it is near-realtime. Issues such as memory handling, user input and performing vector-matrix algebra are addressed.

Acknowledgements

First and foremost, I'd like to thank my supervisor, Dr. Dimitrios Androutsos for all of the support and guidance that he has provided over the past two years of my MASc studies. The respect that he shows towards his students is admirable and an important quality in a supervisor. In my time here, Dr. Androutsos has never put his personal interests above those of his students, instead finding us research and internship opportunities whenever possible.

I would like to acknowledge Dr. Thomas El-Maraghi, the Post Doctoral Fellow I worked along with on the IMAX project. He provided crucial feedback during the development of my thesis and developed the labelling scheme that I later then used to develop a rotoscoping plugin in Adobe After Effects. I would also like to acknowledge Samuel Zhou, Gary Tsui and Ryo Sato from IMAX Corporation, with who I've had the pleasure of working with. They were instrumental in helping me to understand how the rotoscoping process is performed and where improvements can (and need to) be made. I also acknowledge the contribution from Professor Bruce Elder with the School of Image Arts in the Faculty of Communications and Design here at Ryerson. After all, it was the initial collaboration with Professor Elder that resulted in this thesis.

Finally, I want to acknowledge my friends and family. Without my friends, procrastinating and venting my frustrations would be that much less interesting and enjoyable. I thank my brother for graciously donating one of his own personal images to use as a test image. Lastly, but most importantly, my family has always supported me and was always the first to tell me, repeatedly, that getting into Ryerson six years ago was not the end of the world.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

IMAGE segmentation is somewhat of a strange thing. Consider Figure 1.1. A person should have no problem identifying the various cables, the laptop in the right corner and the electronic device in the centre. Whether or not a person would necessarily know what each item was is irrelevant. The fact that segmenting an image is mostly trivial to humans *is* relevant because segmentation is a *non-trivial* task for a computer.

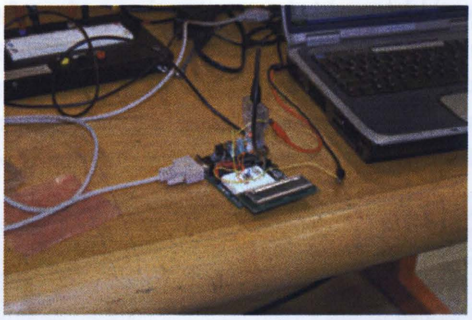


Figure 1.1: A photo of one part of the author's undergraduate design project.

How is this problem considered non-trivial? Note that the laptop on the right side of the image is visually composed of several parts: a black keyboard, a grey mouse pad, a navy "face-plate", a grey rim and a black underside. To a person, even if they do not know *what* the object is, they are still able to see it as one object. For a computer, the fact that these disparate parts constitute one object is not as obvious. An edge-detection algorithm might find the boundaries between these parts but how does the computer know that they are all part of one object? A programmer might specify that a certain distribution of edges and colours represent a particular object, but that still does not guarantee that the computer sees the laptop as one single object.

Conceptually, image segmentation is rather simple: given some image, cut it up so that every object is separated from every other object in the image and the background. While simple in concept, it is quite difficult to execute in practice and many different methods for segmentation have been proposed. Each method has its various strengths and weaknesses and, as a result, one method that may be appropriate for one situation may not always be appropriate for other situations.

What makes image segmentation a desirable research topic, despite being computationally difficult, is because it is so useful. In computer vision, more specifically, object recognition, segmentation provides a pre-processing step that can allow for extraction of an object's shape. In digital media, segmentation provides a way for extracting complicated objects more quickly and accurately then doing so manually.

Regardless, what is important is that a) image segmentation is useful and b) image segmentation is difficult. Because of these two things, it is important to develop segmentation algorithms that are appropriate for the situation in which they will be used. Understanding where the method will be used is key to ensuring that it does a good job. It also provides a convenient way to side-step the entire problem of segmentation being non-trivial for a computer.

1.1 Automated vs. Supervised

Before proceeding any further, it is important to better define, formally, what is meant by "image segmentation". At a very general level, there are two types of segmentation: automated and supervised. Automated segmentation techniques are ones where an image, $I[x, y]$, is partitioned into N regions such that $\mathcal{R}_0 \cap \mathcal{R}_1 \cap \ldots \mathcal{R}_{N-1} = \emptyset$. The entire point of automated segmentation is to allow a system to separate a scene into distinct elements much like how a human does.

An example of where this would be used is in counting the number of objects in an image. Ideally, the objects would be distinct from one another and a segmentation of a scene with N objects would produce a segmentation with $N + 1$ regions; the extra region is the background. However, the downside to automated segmentation is that it is very hard to define what a "good" segmentation is. Often some heuristics are used to define the segmentation quality but, for the most part, the question is pretty much open-ended.

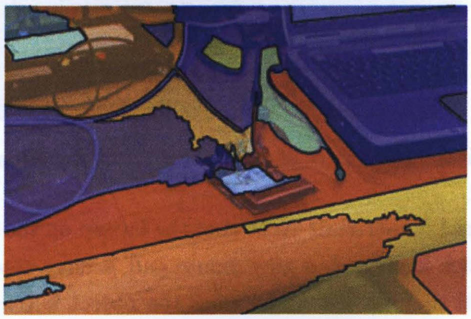


Figure 1.2: An example of automated segmentation performed on Figure 1.1

Figure 1.2 gives an example of what an automated segmentation algorithm would return. This is just a simple example created by modifying one of the demonstration scripts that is provided by MATLAB and is not an indication of automated segmentation algorithms as a whole. However, note the fact that, as far as the algorithm is concerned, the laptop is part of the back of the workbench. Again, this drives the point that segmentation is a difficult computing task.

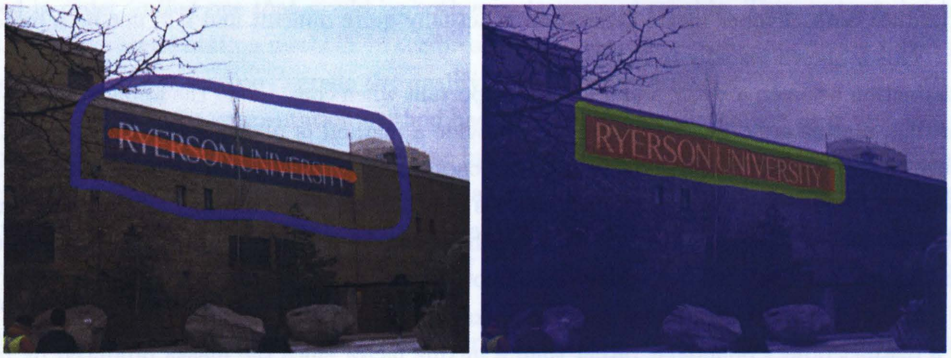
Supervised segmentation, on the other hand, is somewhat of a more straightforward problem. Again, there is some image, $I[x, y]$, that has an object to be extracted. Now, however, the number of partitions are, for all intents and purposes, restricted to two: the foreground, $\mathbb{F}$, and the background, $\mathbb{B}$. Unlike automated segmentation, these two regions are known *a priori* by some user-specifed labelling. A third, *unknown* region, $\mathbb{U}$, exists where the boundary between $\mathbb{F}$ and $\mathbb{B}$ is unknown. A supervised segmentation algorithm operates *only* on $\mathbb{U}$ because that is where the user has indicated that is where the boundary *should* be.

This type of segmentation is used whenever the user needs to maintain a high level of control over the results of the segmentation. The user will provide an initial labelling that specifies where the object being extracted is located in the image. It is up to the particular segmentation algorithm to figure out where the object ends and the background begins. How much a user interacts with the system, how amenable it is to correcting the results and so on all depends on the algorithm being used.

There are a number applications where this sort of control is necessary. For example, in medical imaging, a specialist is more interested in isolating a specific object in an image, such as an abnormal growth, rather than a semi-arbitrary segmentation of the entire image. Similarly, an artist doing compositing and special effects only needs to extract certain objects from the background and not partition it. There are more examples but in any instance where only a certain object, or certain objects, need to be extracted, supervised segmentation is probably the most appropriate tool.

1.2 Trimaps and Masks

What the user is defining, either explicitly or implicitly, in supervised image segmentation is a **trimap**. A trimap is defined as the set $\{\mathbb{F}, \mathbb{B}, \mathbb{U}\}$, where $\mathbb{F} \cap \mathbb{B} \cap \mathbb{U} = \emptyset$. This is not strictly true for every algorithm, but it is generally the case. Figure 1.3 shows the two general types of labelling that can be used for supervised segmentation.



(a) Strokes (b) Border

Figure 1.3: The two different types of labelling generally used for supervised segmentation.

The first style, in Figure 1.3a, is a very rough labelling of the object being extracted (the Ryerson banner). Here, a user has simply painted very rough strokes corresponding to the background and foreground locations. The other labelling style is in Figure 1.3b. This is a much more precise labelling because $\mathbb{U}$, is relatively small with respect to the rest of the image. While there is really no conceptual difference, other then the accuracy of the labelling, the difference in labelling can make a difference in how a given algorithm performs.

In practice, the labelling in Figure 1.3b will always give better results but is not necessarily as desirable from a user-interface point of view.

Once the image has been labeled and the algorithm has been run, something has to be produced. That "something" is an **alpha matte**. In computer graphics, there exists the concept of "compositing", where one image is mixed with another. The degree of mixing is usually denoted by a value, α, that ranges from 0 to 1. The relationship between two images, $I_A[x,y]$ and $I_B[x,y]$, is given by the linear compositing equation

$$I_{comp}[x,y] = \alpha I_A[x,y] + (1-\alpha) I_B[x,y]. \tag{1.1}$$

This is not the only type of compositing, also known as "blending", equation but it is the one that is the most useful for segmentation. The reason is that the output of the segmentation, the matte, is not a single constant but a greyscale image, $\alpha[x,y]$. So, if $\mathbb{F}$ is represented by the image $F[x,y]$ and $\mathbb{B}$ by $B[x,y]$, then the original image is really a composition of two, i.e.

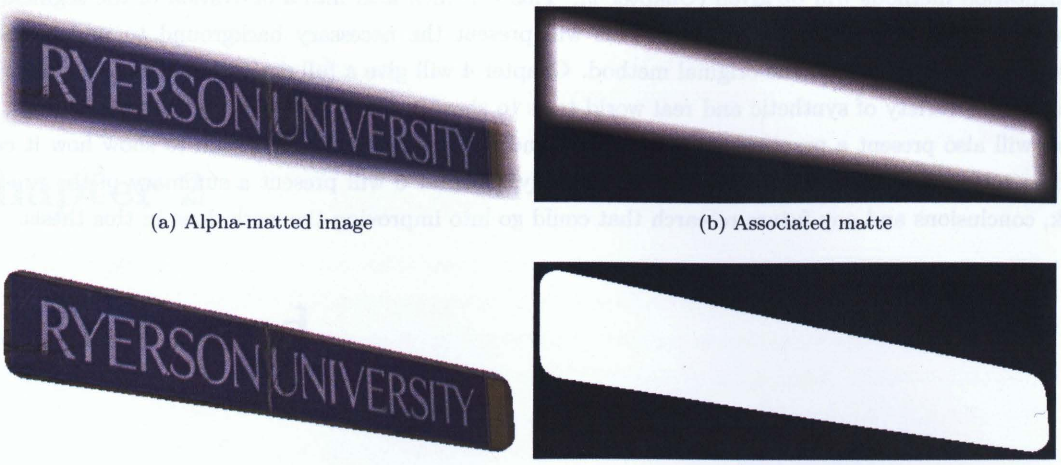
$$I[x,y] = \alpha[x,y]F[x,y] + (1-\alpha[x,y])B[x,y]. \tag{1.2}$$

While the definition of $\alpha[x,y]$ started out as a continuous value in the range of 0 to 1, it does not have to be a continuous value. In fact, $\alpha[x,y]$ can be defined as

$$\alpha[x,y] = \begin{cases} 1 & [x,y] \in \mathbb{F} \\ 0 & [x,y] \in \mathbb{B} \end{cases}. \tag{1.3}$$

By restricting $\alpha[x,y]$ to only take discrete values, it is now referred to as a **mask**. This distinction between a matte and a mask is important because some segmentation algorithms can only return masks while others can only return mattes. Some other algorithms, such as the one presented in this thesis, can actually produce both. Proper matte generation is actually quite difficult and will not be a major topic of discussion. This is shown visually in Figure 1.4.

The distinction between a *matte* and a *mask* is important when considering the application. For example, in compositing, mattes are often used since they provide a method of allowing fine featured or translucent objects, such as hair, to blend in with the composited background. The classical example of this is chromakey, better known as "green screening", where a subject is filmed in front of a brightly coloured background that is quite distinct from the subject. A variety of methods can be employed to obtain a continuous α value for the subject so that they can be overlaid onto another background. For rotoscoping, which is the act of manually extracting elements in a film sequence, masks are often preferred. The reason is that it is important to obtain a distinct object boundary, rather than a mixing factor since the masks produced by rotoscoping are typically used to modify the extracted object, rather then simply blending it into another scene.



(a) Alpha-matted image

(b) Associated matte

(c) Alpha-masked image

(d) Associated mask

Figure 1.4: These images show the difference between an alpha *matte* and an alpha *matte*. A matte represents varying levels of transparency while a mask only indicates if a pixel is transparent or opaque.

1.3 Motivation and Goals

This thesis was, in part, motivated by a separate research project where the goal was to animate a figure inside of a painting. The project was a collaboration with a professor from the School of Image Arts, part of the Faculty of Communication and Design, at the Ryerson. Naturally, the first stage of any animation system would be to extract the object being animated. From there, a number of segmentation methods were investigated in order to find one that would be both easy to use and reliable. This, in turn, led to research into how to improve segmentation methods in the presence of noise, since many images are subject to JPEG compression artifacts that tend to degrade the quality of the segmentations.

Originally, one particular segmentation method had been chosen for the project as it was able to quickly provide accurate segmentations when given a trimap by the user. From a theoretical standpoint, this method was conceptually simple and easier to implement than the majority of the other methods. However, the nature of the algorithm meant that when an image was corrupted by some process, be it noise, compression artifacts or even texture in the image, the quality of the segmentation quickly degraded. It became clear that this segmentation algorithm could be improved.

This thesis' goal was to develop a robust, but theoretically and computationally simple, image segmentation algorithm that would work "well" in a variety of different conditions. Specifically, this thesis will address how an existing segmentation method can be augmented to work for very noisy images. Furthermore, this extension will allow the segmentation method to work in situations where it was unable to work before. Through a simple, though non-trivial, extension to the original method, the new and improved method can return more accurate segmentations when the image has been heavily corrupted by noise. The novelty of this extension is that it does not, in any way, actually modify the algorithm. Rather, it modifies the *input* to the algorithm so that the algorithm is able to better handle noise.

Before any derivation of the new segmentation method is presented, a literature review of pre-existing segmentation methods will be given (Chapter 2). This will then lead into a derivation of the segmentation algorithm being improved (Chapter 3). This will present the necessary background to understand the advantages, and failures, of the original method. Chapter 4 will give a full derivation of the new algorithm, along with a variety of synthetic and real world tests to show it can improve on the original method. This thesis will also present a practical implementation and application of the algorithm to show how it can be used to solve an actual problem (Chapter 5). Finally, Chapter 6 will present a summary of the presented work, conclusions and any future research that could go into improving the work done in this thesis.

Chapter 2

Background

THERE are a number of image segmentation techniques [1]. They can be loosely grouped into two categories: statistical algorithms and graphical algorithms. These categories are by no means exclusive and there are a number of algorithms that are a combination of the two. There are even segmentation methods that are neither graphical nor statistical and use information such as the image gradient.

A statistical algorithm would be any algorithm that uses statistical measurements about the distribution of colours in the image to produce a segmentation. For example, some algorithms create Gaussian Mixture Models (GMMs) [2, 3] of the foreground and background regions. The models give an idea of how the colours are distributed in a colour-space and unknown pixels can be compared to the models to see how they are best classified. Expanding on the example, a mask could be obtained by using the models as a classifier. If an unknown pixel is closer to the foreground model, then it will be a foreground pixel and vice-versa. But, if the distance is used to measure the *relative* membership, it provides one way to produce an alpha matte.

Graphical algorithms treat the image as one big graph. Each pixel is a node in the graph and it is connected by weighted edges to its neighbors. The advantage, usually, of graphical methods is that they take the location of each pixel into account. Imagine that an image has two similarly coloured objects on opposite sides of the image, separated by the background. A statistical algorithm would look at *only* the colour distribution so some methods might pick both objects as the foreground when only one object was selected. A graphical algorithm would also use some image statistics to decide how nodes are related but this would take the form of edge weights.

The distinction being made between the two categories is somewhat arbitrary because a graphical algorithm will use statistical information about the image and a statistical algorithm will, in one way or another, take the pixel locations into account. However, this distinction is important because an algorithm that works primarily on image statistics will behave differently then one that treats the image as a graph. An understanding of these differences also helps to provide an understanding of the current state of image segmentation research.

2.1 Traditional Segmentation

For the purposes of this thesis, "traditional" segmentation techniques are the automated techniques alluded to in Chapter 1. This type of segmentation requires little, if any interaction from the user. Some methods rely on edge information about the image while others attempt to classify different parts of the image using unsupervised classification (i.e. clustering) techniques from machine learning [4]. These are very broad generalizations but they loosely classify how these algorithms are designed.

- **Cluster-based Segmentation** is any type of automated segmentation technique that utilizes an unsupervised clustering techniques such as *K-means* [3] or GMMs. This particular type of segmentation requires that a user assume the number of clusters, or distinct objects, in a scene. The primary drawback to these methods is that there is no way guess how many distinct objects are in a scene. Furthermore, visually distinct objects can be composed of several colour groupings in a colourspace and an unsupervised classifier has not means in which to determine which clusters are actually part of one distinct cluster.
- **Thresholding** methods are primarily used for greyscale images since the concept of thresholding has little meaning for colour images. These methods make the assumption that distinct objects in a scene will present as distinct modes in a histogram of the image. Otsu's method [5] is a good example of a thresholding method. It works under the assumption that the foreground and background in any scene will present as distinct peaks in a histogram and it seeks to find a threshold that will best separate the peaks. The obvious draw back for any thresholding method is that it cannot be applied to colour images.
- **Morphological Processing** can be used to remove small-scale image features and identify the positions of large-scale (i.e. objects) in the image. From there, algorithms such as Topological Watershed [6] can be used to obtain the actual object edges from the image gradient. This type of algorithm was used to produce the example in Chapter 1.
- **Normalized Cuts** [7] by Shi and Malik is an interesting segmentation technique that relies on the spectral properties of a *N*-connected graph constructed from the image data. The basic premise is that when the image is considered as a large graph, the edge weights can be used to encode information regarding the similarities between pixels. Groups of pixels that close to one another and similar in colour are considered to be part of the same objects so a cut along the strongest edges will lead to a "proper" segmentation. The technique assumes that the eigenvectors of the graph will contain information on where to make the most effective cuts. It does so in a recursive manner since large objects can be further segmented into smaller partitions.

This list is by no means exhaustive. Many other automated segmentation techniques exist that are designed for specific tasks (such as analyzing cell cultures [8, 9]). However, these techniques are difficult to apply for situations where the user *must* have some control over the segmentation. They segment the entire image and, as such, are useful when the image itself needs to be partitioned. However, if a user knows where the object boundary is, then is trivial for the user to mark where the boundary is. Please note that "trivial" does not imply "easy to do". This contradictory statement simply means that it is easy for a user to determine where the boundary is but that it is rather time-consuming and tedious to actually trace it out. Generally, the more complex the boundary, the more tedious it is to manually segment out the object. Supervised segmentation methods simplify this task but letting an algorithm extract the boundary with some initial constraints provided by the user.

2.2 Statistical and Other Methods

All of the methods outlined here rely either on statistical methods or some other image information in order to determine pixel membership with respect to the foreground. These methods are being separated from the graphical approaches because they do not treat the image as a graph. However, all of these methods require a trimap to be specified so the pixel location information is being taken into account. Therefore, pixels outside of the unknown region are not being processed by the algorithm. The following list, in no particular order, provides a quick summarization of these segmentation methods:

- **Bayesian Matting** [10], proposed by Chuang *et al*, builds a series of GMMs from the colour distribution in the neighborhood of an unknown pixel. The neighborhood is large enough so that it encompasses both the foreground and background. Their algorithm also builds a GMM for the unknown region. It uses that information to project the colour vector of the unknown pixel onto a line that connects the three distributions together. The relative position on that line is the α-value of that pixel.

- **Hillman et al** [11] built on the work of Ruzon and Tomasi [12] to produce mattes for high-resolution images and videos. Their method works under the assumption that colours fall into "cigar shaped" clusters in the RGB colourspace. By finding the projection of the current vector onto the line joining the two distributions, they are able to estimate α.

- **Poisson Matting** [13] by Sun *et al* is not a "statistical" method. Rather, it operates on the gradient of the image being segmented. They define the alpha matte as the solution to

$$\nabla^2 \alpha[x,y] = \operatorname{div}\left(\frac{\nabla I[x,y]}{F[x,y]-B[x,y]}\right). \tag{2.1}$$

  This method relies on the premise that an image's gradient will contain information on object boundaries. Furthermore, how strong the boundary is an indication of how likely a pixel is to be a member of the foreground.

- **Robust Matting** [14] was proposed by Wang and Cohen as a method to improve on the actual sampling techniques, rather than the distribution modeling itself. Their method samples points along the borders of the unknown region and builds an initial alpha matte. They then use a graphical optimization technique to improve on the matte. However, this method cannot be really considered to be graphical since the optimization stage is to smooth out the initial estimate matte.

The commonality between these methods is that they all attempt to find an alpha *matte*. However they do it, they all operate under the assumption that α is a continuous value in the range of $[0, 1]$. These methods can afford to do this because statistical methods can not only assign membership, they can also be used to determine the degree of membership.

However, many statistical methods exist because, often there is no "good" way to determine the proper matte. Generally, the closer in colour distribution two objects are, the less likely that a statistical method will work. The same logic also applies to methods such as Poisson matting. If the gradient has very little information, i.e. weak object edges, then it will also produce less accurate mattes.

A variety of extensions to the alpha matting methods have been proposed in order to improve the segmentation under certain circumstances. While this does restrict the situations that the method may be used in, it means that it can be used more effectively if these special conditions are met. Here are a few methods that extend Bayesian Matting:

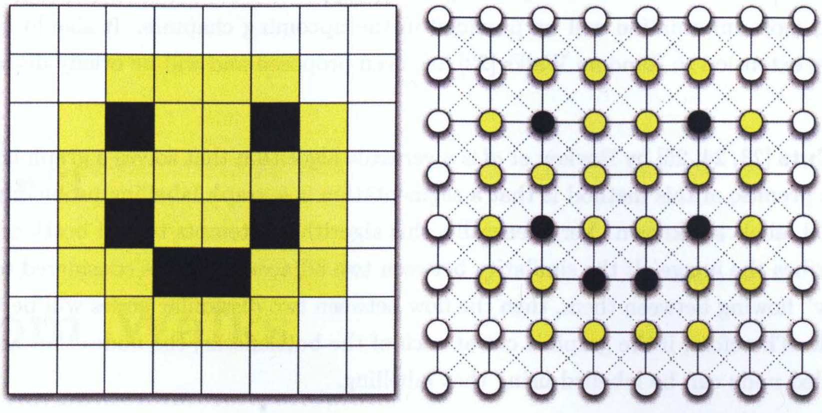
- **Flash Matting** [15] by Sun *et al* applies a modified version of Bayesian Matting to flash-image pairs. A "flash-image pair" is a pair of photographs where one was taken with a camera flash and one without. Flash Matting relies on the observation that, in most cases, the major difference between an image without a flash and with a flash is the illumination of the foreground object. Because the background is too far away, there will be little to no change in illumination. While restricted to flash-image pairs, this method was able to improve Bayesian Matting in circumstances were the edge boundary was quite weak.

- **Video Matting** [16], by Chuang *et al*, extends Bayesian Matting to video applications. A user specifies an initial trimap in one frame, referred to a as "garbage matte" and optical tracking algorithms interpolate the trimap across the video sequence. The matting algorithm is then performed for each frame in the sequence using the interpolated trimap. The result is a video matte, rather then a single image.

There are many more matting algorithms that are based on the statistics of an image. It is always possible to perform some sort of analysis on an image and make an estimate of the object's matte. However, statistical analysis methods have a tendency to be affected by noise and outliers. Theses methods, in one way or another, try to deal with this problem. Often, it is up to the user to correct any mistakes made in the matte. One such example is the "Extract" filter provided in Adobe Photoshop. The basic output of the filter is somewhat similar to Bayesian Matting but there is no way to know for sure if this is what is used since the algorithm is not named and is most likely proprietary. For images where the object boundary is ill-defined, the matte tends to have many errors so matte-correction tools are provided so that the user can improve the matte.

2.3 Graphical Methods

As previously mentioned, it is natural to consider an image as a very large graph. The pixels are nodes in the graph and the weighting of the edges in the graph determines the relationship between a pixel and its neighbors. This is a very loose definition because it does not consider *how* the nodes are connected to each other. The simplest graph one can construct is a 4-connected graph where a pixel is connect to its two horizontal and two vertical neighbors. Therefore, the distance between nodes is described using a Manhattan metric. If the graph is 8-connected, then the distance is described by a Chebyshev metric because the node's diagonal neighbors are also being considered. Figure 2.1 illustrates this graphically.



(a) Image as a Grid

(b) Image as a Graph

Figure 2.1: An example of how an image can be represented by both a pixel grid or a N-connected graph. The solid lines are the edges in a 4-connected graph while the dashed lines are those in a 8-connected graph. In both cases, the edges are weighted so that the weighting represents how similar or dissimilar two adjacent nodes (pixels) are.

The choice of the graph connections depends entirely on the problem being solved. Commonly, 4 or 8-connected graphs are used because they take into account all possible horizontal, vertical and diagonal differences in a small neighborhood around any given pixel. However, other types of graphs can be constructed where, for instance, the pixel is connected to 22 of its neighbors. In this case, the segmentation algorithm is now taking in account variations in the image over large distances. This can be important in case where the object boundary is very weak.

Here are several notable segmentation algorithms that operate on a graph:

- **Intelligent Scissors** [17, 18] by Mortensen and Barret provides the user with a simple and effective segmentation system. The method is designed to be interactive and all that is required of the user is to place down points the boundary of the object being extracted. The system tries to find the "shortest path" between the previous point and the current point. The distances between pixels are defined by their gradient magnitudes and the greater the magnitude, the "closer" two nodes are. Therefore, the shortest path found by the algorithm should lie on an object's edge.

- **Unified Image Segmentation and Matting** [19] is another segmentation algorithm developed by Wang and Cohen that utilizes belief propagation to iteratively guess an image's true trimap. The user simply provides a series of strokes that roughly denote the object's location and the background. They define an energy function to be minimized by their algorithm using a combination of edge weights and GMMs. This information allows them to guess at each iteration what the matte should be for each pixel. Eventually, the number of unknown pixels will vanish and the final "true" matte with be the result.

- **Random Walks** [20, 21] is a newer segmentation algorithm developed by Grady *et al* that determines the probability of a random walker visiting every node in a graph. Those probabilities can be used to determine the mask or matte, depending on the application. Because this thesis is *based on* this algorithm, more information will be provided in the upcoming chapters. It should be noted that an alternative extension to Random Walks [22] has been proposed and will be briefly discussed in Chapter 4.

- **Graph Cuts** [23, 24, 25] by Boykov *et al* is a versatile algorithm that solves a graph labelling problem. The basic premise of this method is that a segmentation is a graph labelling problem where only some of the pixel labels are known. Very generally, this algorithm attempts to find bottlenecks in the graph that describes the image. If the similarity between two adjacent nodes is considered to be the amount of "energy" flowing between them, then the flow between two dissimilar nodes will be very small, i.e. a bottleneck. Therefore, if the graph is cut at each of the bottlenecks, the nodes that are still connected to a labelled node can be labelled using that labelling.

An interesting property of graph-based segmentation algorithm is that they do not necessarily have to be applied to segmentation problems. Graphical algorithms attempt to minimize some energy function over a graph and label the nodes based on some *a priori* information. Graph Cuts, while being applicable to segmentation, can also be used in application such as image stitching [26] and stereo-matching [27]. Image segmentation is simply one type of graph labelling problem and similar techniques can be used to solve other labelling problems.

Chapter 3

Random Walks

THE Random Walks algorithm provides a very simple way to segment an image. It was originally proposed by Grady *et al* in [21] as a way to address some of the problems with the Graph Cuts algorithms. Namely, Graph Cuts is a very computational expensive algorithm to run and it suffers some something known as the "small-cut problem". Random Walks was designed with these problems in mind and the way in which it was defined allows it avoid these issues.

This chapter provides a complete discussion on the Random Walks algorithm. This discussion is important to understand the problems that plague Random Walks and what steps can be taken to fix or mitigate them. The algorithm itself is conceptually simple but a full derivation will be presented so as to show how the algorithm itself works. The simplicity of the algorithm belies the algorithm's versatility in a number of situations and applications. However, like all algorithms, it is not without its faults and this chapter will also show how these faults can seriously hamper the algorithm's performance.

3.1 The Problem With Graph Cuts

The specific problem that Graph Cuts solves is known as "max-flow/min-cut". The problem goes as stated: "Find all of the edges in a graph that can be cut in such a way that all of the remain edges are at maximum capacity." Nodes can be labelled as "source" nodes by connecting them to virtual nodes with an edge weighting indicating maximum similarity. The maximum flow is considered to be the flow of "energy" (a loosely defined term that is dependent on the context of the application) from one source node to another. Figure 3.1 shows how this graph structure is constructed and what the algorithm is actually doing. Algorithms such as the Ford-Fulkerson algorithm [28] can be used to find the maximum flow in the graph and other algorithms can be used to find the appropriate cut. Unfortunately, this type of problem falls under the class of problems known as "NP-Hard".

NP-Hard is a term reserved for any problem whose solution is easy to verify but very difficult and time-consuming to find. The most intuitive example is the "subset-sum" problem. The problem is, "given a set of number, what subset of those numbers will add up to zero?" For example, assume that the set of numbers is $\{2, -3, 1, -2\}$. It is trivial to verify that the subset $\{2, -3, 1\}$ adds up to zero. The problem is finding that

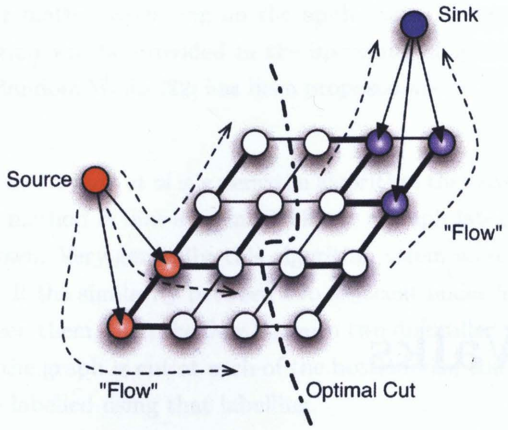


Figure 3.1: Graph structure typically used for a Graph Cuts image segmentation. The thick lines represent high capacity edges while the thing lines are low capacity edges. The two labels, "source" and "sink" indicate the two regions that the algorithm is supposed to find the optimal boundary between. Note that these virtual nodes can be connected to the unknown nodes with weights describing the affinity of a pixel to a particular label.

subset in the first place. The naive way would be to test out *every* combination, so for N elements, you have to test $(N-1)!$ cases (since the single element subsets do not count). For many NP-Hard problems, there is no "good" way to solve them. Often some sort of heuristics are needed in order to find a "good enough" solution.

A result of all of this is that Graph Cuts is fairly slow. A number of methods that use Graph Cuts in segmentation perform a number of steps before applying the algorithm to speed it up. Lazy Snapping [29] is one such Graph Cuts based method that utilizes a pre-segmentation stage. Prior to running the algorithm, Lazy Snapping performs an initial watershed segmentation to over-segment the image. The watershed segmentation breaks the image up into very small, nearly homogeneous groups that are referred to as "super nodes". These super nodes represent areas of the image that are *almost* the same and the actual Graph Cuts segmentation is performed on these nodes. Because there are less super nodes then there are pixels, Graph Cuts runs noticeably faster. In fact, the authors reported execution times that allowed for realtime user interaction. However, the authors trade the accuracy of a full Graph Cuts segmentation for the speed provided by the initial over-segmentation.

The small-cut problem is another issue that plagues Graph Cuts. Because Graph Cuts can utilize statistical information based on image features, such as colour distribution, it does a reasonably good job at segmenting an image using very little labelling. However, sometimes Graph Cuts can be too aggressive and it produces segmentations that are closer to the initial seeds than desired. To solve this, more seeds need to be placed in order to inform the algorithm where there was a mistake in the segmentation. The Grab Cuts [30] algorithm is a partial solution to this problem. It repeatedly performs an unsupervised Graph Cuts segmentation on a particular region of an image that is specified by the user to contain an object of interest. The algorithm stops once some energy function has been minimized. However, sometimes certain portions of the object are not included and have to be manually relabelled by the user.

3.2 Random Walks on a Graph

Random Walks is a graphical algorithm that tries to determine the probability that a random walker, starting at some source node and travelling to a sink node, will visit every node in the graph. The edges are weighted so that the walker is more, or less, likely to cross some edges rather than others. The result is that the walker will be more likely to visit some edges as opposed to others. If the weights are based, through whatever means, on the relationship between pixels then probabilities that are returned represent how likely a pixel is to be related to a source pixel.

The derivation that Grady provides in [21] is from a statistical viewpoint. However, he also mentions the equivalence to the Random Walker and an electrical circuit. In fact, the circuit analogy provides a rather intuitive way to derive the Random Walker. It requires little more than an understanding of some electrical theory, namely Ohm's Law and Kirchoff's Current Law (KCL), and linear algebra. Figure 3.2 shows how the Random Walker graph is laid out. This is *practically* identical to what is shown in Figure 3.1 but *conceptually* different. Here, the nodes are simply nodes in a circuit and all of the edges are the resistors connecting those nodes. The entire image is one, very large, resistor network.

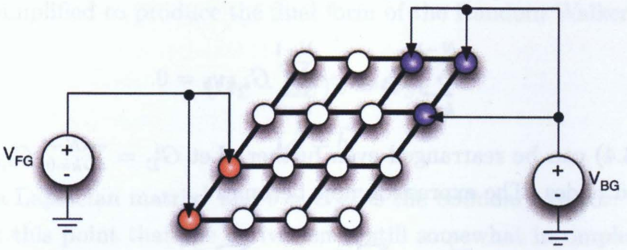


Figure 3.2: Graph structure as used by Random Walks. This is functionally identical to the structure shown in Figure 3.1 but the different problem formulation means the solution is different. The two voltage sources represent the different voltages that will represent each label. In a two-label case, one voltage is always ground to simplify the calculations.

Figure 3.3 shows a single node of a 4-connected graph as an electrical circuit. The resistors represent the dissimilarity between the current node, i, and its neighboring nodes.

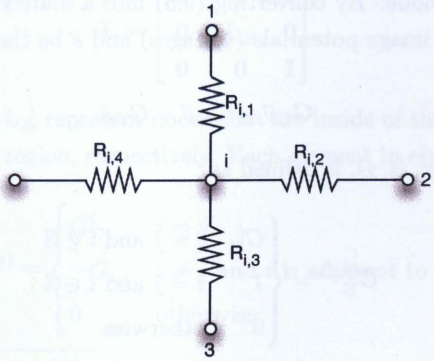


Figure 3.3: A node in the Random Walker graph as an electrical circuit.

The voltage at node i, v_i, can be found by applying Kirchoff's Current Law to that particular node. KCL states that the sum of all of the currents going out of the node must be equal to zero. Therefore, for an N-connected graph, the sum of the currents going from i to k, $I_{i,k}$ is

$$\sum_{k=0}^{N-1} I_{i,k} = 0. \tag{3.1}$$

By Ohm's Law, $I = V/R$ and therefore equation (3.1) becomes

$$\sum_{k=0}^{N-1} \frac{v_i - v_k}{R_{i,k}} = 0. \tag{3.2}$$

To simplify the math, rather then using the resistance value, the conductance, $G_{i,j} = 1/R_{i,j}$ can be used instead. Therefore, (3.2) can be rewritten and rearranged so that

$$\sum_{k=0}^{N-1} G_{i,k} \left(v_i - v_k\right) = 0 \tag{3.3}$$

$$\sum_{k=0}^{N-1} G_{i,k} v_i - \sum_{k=0}^{N-1} G_{i,k} v_k = 0. \tag{3.4}$$

The expression in (3.4) can be rearranged even further. Let $G_{\Sigma}^{i} = \sum_{k=0}^{N-1} G_{i,k}$ and $\mathbb{S}$ represent the set containing *all* of the seed nodes. The expression now becomes

$$G_{\Sigma}^{i} v_i - \sum_{k \notin \mathbb{S}} G_{i,k} v_k = \sum_{k \in \mathbb{S}} G_{i,k} v_k. \tag{3.5}$$

The notation "$k \in \mathbb{S}$" refers to any of the neighboring nodes, k, that are also a member of the seeds sets. In other words, the summation is split up to accommodate the situation when a node is bordering nodes with known values.

It is somewhat clear from (3.5) that what is being described is a system of linear equations. However, right now this is only describing the voltage at each individual node. It would be more convenient if this described the voltages at *every* node. By converting (3.5) into a matrix expression, this becomes possible. Therefore, let $\vec{v}$ be the vector of image potentials (voltages) and $\vec{s}$ be the seed vector so that

$$\mathbf{G}_{\Sigma} \vec{v} - \mathbf{G}_{U} \vec{v} = \mathbf{G}_{S} \vec{s}, \tag{3.6}$$

where each element in the matrix, (i, j), is defined as

$$G_{\Sigma}^{(i,j)} = \begin{cases} G_{\Sigma}^{i} & i = j \text{ and } i \notin \mathbb{S} \\ 1 & i = j \text{ and } i \in \mathbb{S}\ , \\ 0 & \text{otherwise} \end{cases} \tag{3.7}$$

$$G_U^{(i,j)} = \begin{cases} G_{i,j} & j \notin \mathbb{S} \\ 0 & \text{otherwise} \end{cases} \tag{3.8}$$

and

$$G_S^{(i,j)} = \begin{cases} G_{i,j} & j \in \mathbb{S} \\ 0 & \text{otherwise} \end{cases} \tag{3.9}$$

The two matrices, $\mathbf{G}_U$ and $\mathbf{G}_S$, can be considered as "sub-adjacency matrices". They describe the graph connections all of the neighbors that are unknown ($\mathbf{G}_U$) and all of the neighbors that are seeds ($\mathbf{G}_S$). The complete adjacency matrix is $\mathbf{G} = \mathbf{G}_U + \mathbf{G}_S$. This matrix is always sparse and symmetric. If a regular lattice structure, such as a grid is used, then the matrix is also banded. This is important for computational reasons as it allows the matrix to be compressed and require less storage. The matrix $\mathbf{G}_\Sigma$ is the degree matrix of $\mathbf{G}$. Each element along the main diagonal of $\mathbf{G}_\Sigma$ is the sum of all of the edge weights connected to that particular node.

(3.6) can be further simplified to produce the final form of the Random Walker algorithm so that

$$(\mathbf{G}_\Sigma - \mathbf{G}_U)\,\vec{v} = \mathbf{G}_S\vec{s} \tag{3.10}$$

$$\mathbf{L}\vec{v} = \vec{b}, \tag{3.11}$$

where $\mathbf{L} = \mathbf{G}_\Sigma - \mathbf{G}_U$ is a Laplacian matrix[1] and $\vec{b} = \mathbf{G}_S\vec{s}$ is the boundary vector.

It should be noted at this point that the derivation is still somewhat incomplete. As defined in (3.11), the Random Walker is a solution to a linear system. However, this system contains a fair bit of redundant information. The potential at many of the nodes is already known since some have already been designated as source or sink nodes. Therefore, the values along the main diagonal of $\mathbf{L}$ that are known will be '1' and the remaining values along the row will be '0'. Similarly, if i is in $\mathbb{S}$, then the value in $\vec{b}$ will be its equivalent value in $\vec{s}$, or $b_i = s_i$. The system is over determined and there are nodes that do not need to be solved for.

To simplify the problem and remove redundant information, $\mathbf{L}$ can be decomposed such that

$$\mathbf{L} = \begin{bmatrix} \mathbf{L}_U & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{L}_B & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} \end{bmatrix} \tag{3.12}$$

The two sub-matrices, $\mathbf{L}_U$ and $\mathbf{L}_B$ represent nodes that are inside of the unknown region and nodes that are on the border of the unknown region, respectively. Each element in either sub-matrix is defined as

$$L^{(i,j)} = \begin{cases} G_\Sigma^i & i = j \\ -G_{i,j} & i \neq j \text{ and } i \text{ is adjacent to } j\ . \\ 0 & \text{otherwise} \end{cases} \tag{3.13}$$

[1] A Laplacian matrix is defined as $\mathbf{L} = \deg(\mathbf{A}) - \mathbf{A}$, where $\mathbf{A}$ is the adjacency matrix of the graph and $\deg(\mathbf{A})$ is the degree matrix.

Also, the boundary vector, $\vec{b}$ can be written as

$$\vec{b} = -(\mathbf{L}_B)\,\vec{s}. \tag{3.14}$$

Since it is not necessary to solve for the already known nodes, the final solution to system is simply

$$\vec{v} = (\mathbf{L}_U)^{-1}\,\vec{b}. \tag{3.15}$$

By decomposing the system matrix, the size of the system being solved is limited. Rather than solving for every node in the image, which is not necessary, only the unknown nodes are solved for. If a stroke-style labelling is being used (Figure 1.3a in Chapter 1) then this does not necessarily have much of an impact. But, if a trimap-style labelling is used (Figure 1.3b) then this *will* make a difference since only a small percentage of the nodes have to be solved for.

The final result of the algorithm is a **probability map**, $P[x, y]$, where $0 \leq P[x, y] \leq 1 \forall [x, y]$. However, the term "probability" is somewhat of a misnomer that arises from the algorithm's connection to the probability of a random walker visiting a node. Treating the image as an electrical circuit means that $P[x, y]$ contains the voltages at each node. In reality, $P[x, y]$ represents likelihoods but, out of convenience, it is considered to contain probabilities.

Normally, when using the Random Walks algorithm, the foreground, or source, is chosen to be 1.0V and the background, or sink, is chosen to be ground (0V). This is done to maintain the relationship with probability but it is not that important. All of the unknown node values will range from (x_{BG}, x_{FG}), where x_{FG} is the assigned foreground value and x_{BG} is the assigned background value.

3.3 Edge Weighting

Choosing the proper edge weighting is important. The edge weighting determines how similar, or dissimilar, two pixels are. Originally, when considering the image as a circuit, the pixels were connected by resistors. The resistors represented how different two nodes were from one another. A resistance of zero indicates that the two nodes are identical while an infinite resistance implies the two nodes are completely dissimilar.

The statistical equivalent of a resistor in this problem would be some distance function, $d(i, j)$. Distance has the convenient property of being zero when two objects are "close together" (the same) and being very large when they two objects are "far apart" (very different). For pixels, these objects would be the colour vector, $\vec{c}_i$. There are a number of different distance measures to choose from however, the distance function

$$d(i, j) = \|\vec{c}_i - \vec{c}_j\| \tag{3.16}$$

works very well in most cases, where $\|\cdot\|$ is the Euclidean norm.

However, in (3.3), the expression was changed to use *conductance*, which is a measure of similarity. Therefore, the weighting should describe similarity, not dissimilarity. The inverse of the distance cannot be used because the result will be infinity if the two data points are the same and therefore results in computational problems. Grady's choice of a weighting function is

$$G_{i,j} = \exp\{-\beta d(i, j)\}, \tag{3.17}$$

where β is a free parameter that is usually set to 90. The distance term is normalized to be in the range of $[0, 1]$ so that the similarity function performs the same regardless of the input data.

The problem with using an exponential is that it decays very quickly the moment that $d(i, j) > 0$. If the two data points being considered are very similar, but slightly different to noise, then the weighting may not necessarily reflect the *true* similarity. A better choice would be to use a weighting function that is "flat" in the range of $[0, \epsilon]$, where ϵ is some arbitrarily small value, and does not decay as quickly. Fortunately, a sigmoidal weighting function such as

$$G_{i,j} = \frac{2}{1 + \exp\{\beta d(i, j)^{\gamma}\}} \tag{3.18}$$

satisfies this condition. The β and γ terms can be used to control the decay and "flatness", respectively. This provides a weighting function that is much more flexible and provides a degree of noise rejection.

Figure 3.4 shows a comparison between the exponential weighting function and the sigmoidal weighting function in the range of $[0, 0.15]$. This subset of the total range of $[0, 1]$ was chosen because of how quickly the two function must decay in order to prevent the weighting function from becoming too inclusive.

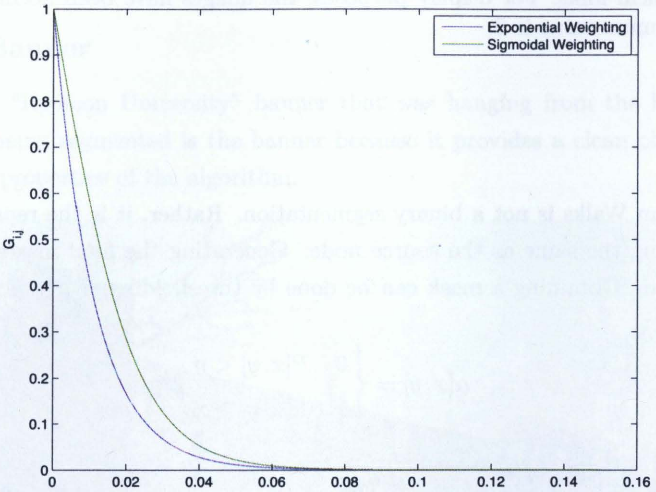
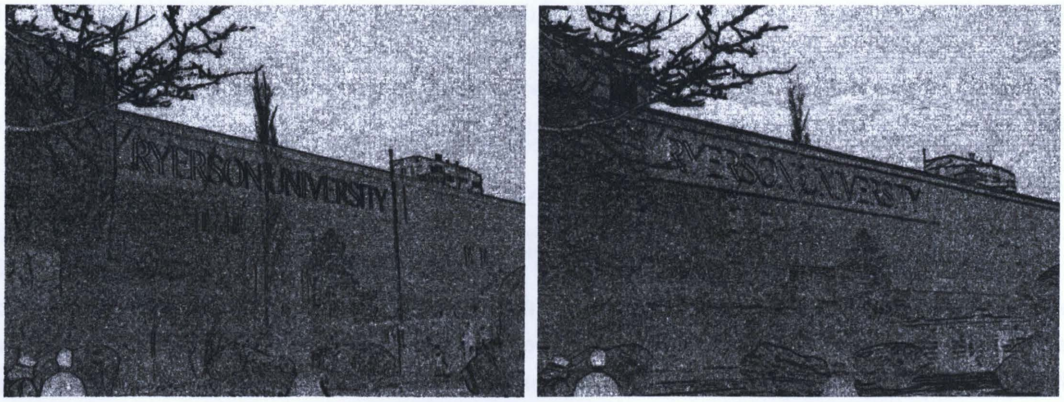


Figure 3.4: Comparison between the exponential and sigmoidal weighting functions. A value of $\beta = 90$ was chosen for the exponential function and values of $\beta = 90$ and $\gamma = 1$ were chosen for the sigmoidal function.

The weights of the sigmoidal function were chosen to resemble the exponential since the performance of the exponential weighting function is well established. Empirically, it was found that the sigmoid provided better distinction between edges than the exponential. This is due to the fact that the sigmoid does not decay as rapidly as the exponential and provides better noise rejection. Figure 3.5 shows what the weighting scheme appears when applied to an image. Brighter pixels are links where the similarity is high while darker pixels represent links between dissimilar pixels. This is a sort of "reverse edge detector" in that pixels on an edge are represented by small values (close to zero) and pixel not on an edge are represented by large values (close to one).



(a) Horizontal Weights

(b) Vertical Weights

Figure 3.5: Images showing the edge weighting function applied to an input image. Note that there are two "images", a horizontal and vertical image, respectively. This is to remain consistent with the fact that there are horizontal and vertical links. For display purposes, the images have been normalized so the maximum value is 1 and the minimum value is 0.

3.4 Matting

The solution to Random Walks is not a binary segmentation. Rather, it is the representation of the probability of one node being the same as the source node. Generating the final matte depends on how these probabilities are treated. Obtaining a mask can be done by thresholding the resulting **probability map**, $P[x, y]$, so that

$$\alpha[x, y] = \begin{cases} 0 & P[x, y] < \eta \\ 1 & P[x, y] \geq \eta \end{cases}. \tag{3.19}$$

There is no optimal way to choose a value for η. One way is to treat η as a fixed constant. If $P[x, y]$ is interpreted to represent probabilities then $\eta = 0.5$ is usually a good choice. The reasoning is that any node (pixel) with $P[x, y] = 0.5$ is equally likely to be in the foreground or background. The result is that any pixel with less than a 50% chance of being in the foreground is rejected. Another way is to use automated greyscale thresholding methods, such as Otsu's method [5], to pick a value for η. The benefit to using an automated method is that it can provide a better segmentation but the quality of the thresholding algorithm now affects the performance of Random Walks. All of the segmentation examples provided in this thesis use $\eta = 0.5$ so that all of the results remain consist and there are no external factors affecting the quality of the segmentation.

Another option, if an alpha matte is desired, is to "filter" $P[x, y]$ in one way or another. This means that

$$\alpha[x, y] = f(P[x, y]), \tag{3.20}$$

where $f(\cdot)$ is the filtering function. How simple or complex $f(\cdot)$ is depends on how much effort will be devoted to extracting a matte from the map. For example, a sigmoidal function could be used to suppress very low probabilities without creating a binary mask. This particular issue is unimportant to this thesis and will not be addressed any further. As previously mentioned, all of the segmentations are produced using a fixed threshold to remain consistent.

3.5 Some Examples

While the mathematical derivation of the algorithm is important, it is just as important to show what exactly the algorithm *does.* The following examples are segmentations of natural images (i.e. photographs) using Random Walks. The parameters used for the Random Walks are: $\beta = 90$, $\gamma = 1$ and $\eta = 0.5$. The sigmoidal weighting function is used since it provides slightly better segmentations but it does not provide a dramatic, and in many case even a noticeable, improvement. Each image was converted from the RGB colourspace to the CIE L*a*b* colourspace [31] prior to the segmentation due to the fact that the L*a*b* colourspace is perceptually linear. Other colour transformations could also be used, such as what was done in [20], but, in practice, the L*a*b* colourspace is a suitable choice for the colourspace.

3.5.1 Ryerson Banner

This is an image of a "Ryerson University" banner that was hanging from the back of the Image Arts building. The object being segmented is the banner because it provides a clean object to extract. It also illustrates some of the properties of the algorithm.

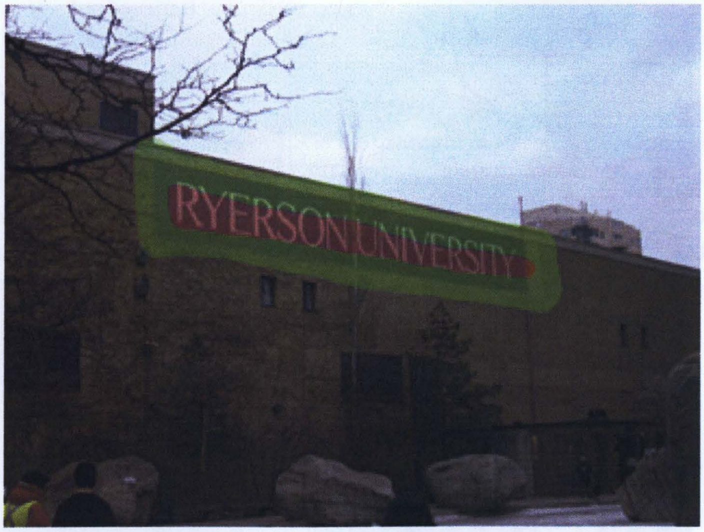
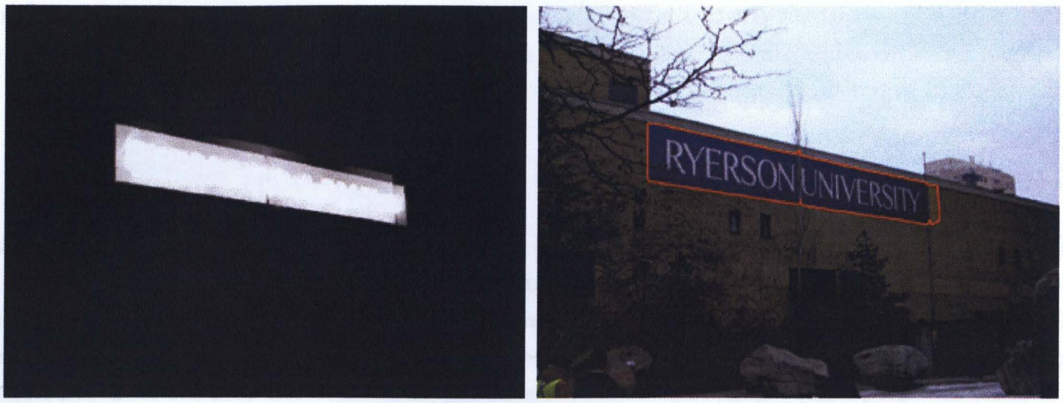


Figure 3.6: Ryerson University banner with the overlaid trimap. Green is the unknown region and red is the foreground.

The unknown region is made to be quite wide in order to show what happens when the algorithm is run. Figure 3.7 shows the resulting Random Walks segmentation. This image is fairly noise free so it provides a good indication of how the algorithm operates on a clean image.



(a) Probabilities

(b) Segmentation

Figure 3.7: Segmentation results, and corresponding probabilities, of the segmentation of the Ryerson banner in Figure 3.6.

Overall, the segmentation result is quite good. The object has, for the most part, very strong edges that make it "easy" for the algorithm to find boundaries. However, the rightmost portion of the banner, i.e. the yellow rectangle fades into the background and the algorithm does not detect the edges as easily. Similarly, the midpoint of the banner where the tree covers the banner also causes an error in the edge. Figure 3.8 provides a close-up to these two areas to show what exactly is happening.

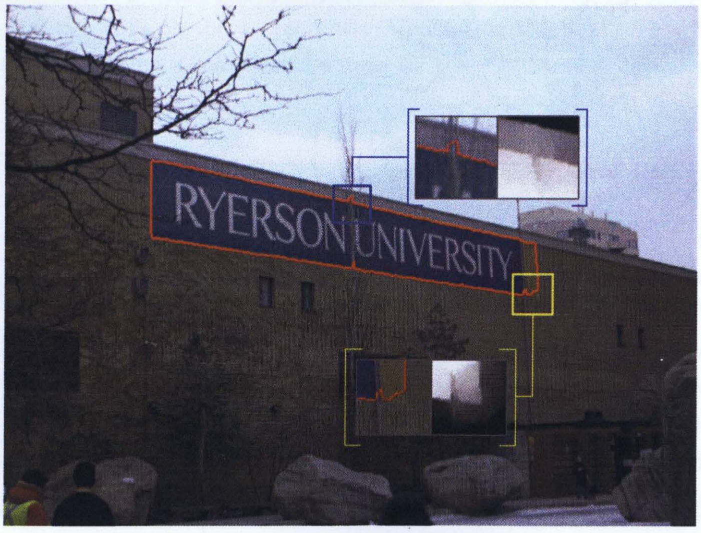


Figure 3.8: Comparison between two regions in the banner image where the segmentation was less than ideal.

Here is an explanation of *why* these two failures happened. For the first case – denoted by the blue box – the tree branch is breaking the edge of the banner into two parts. If the edge was continuous then there would be a large, and noticeable, drop in the potentials from one side of the edge to another. But, since the edge is split into two parts, the potentials "leak" where the break takes place.

The other case – denoted by the yellow box – occurs because of the colour of the bricks and the yellow portion of the banner are similar in colour. The edges are very weak and, by design, Random Walks is somewhat conservative when estimating the edge locations. Therefore, Random Walks picks an edge somewhere in the middle of the foreground and background labelling.

Formally, these failures can be explained through electromagnetic field theory. Consider a rectangular sheet of some sort of uniformly semi-conductive material. Now, imagine that the sheet has been cut in half *except* for a tiny strip in the middle. A voltage source is connected to one corner of the sheet and another corner, on the other side of the cut, is connected to ground. The measured voltages across the cut will be relatively high; almost that of the voltage source. However, if the voltage is measure across the small strip, they will be much lower because the two sections are this point are connected to one another so the resistance between them is much lower. This is what is happening, from an electrical viewpoint, in the first failure case.

Now, imagine that the sheet has not been cut; it is just one large sheet of semi-conductive material. Because the one corner is connected to a voltage source and the other is connected to ground, there *has* to be a linear decrease in potential between the two points (the resistance of the material at any given point is constant). Therefore, a voltage value half that of the source has to be halfway between the source and ground. This is the situation that occurs when there is little to no variation between the foreground and background labelling. All of the weights are roughly the same so the potential (probability) decreases linearly from the foreground to the background.

3.5.2 Basement Shelf

The "Basement Shelf" photo shown in Figure 3.9 is an example of a "bad" image. It was taken under poor lighting conditions using the camera inside an Apple iPhone 3G. Under low illumination, the sensors used to capture images tend to be very sensitive to noise. As a result, this picture suffers from all of the problems that are present in all cellphone camera photos: noisy, high JPEG quantization and ill-defined edges. It provides a good test of the performance of Random Walks on a "less than ideal" image.

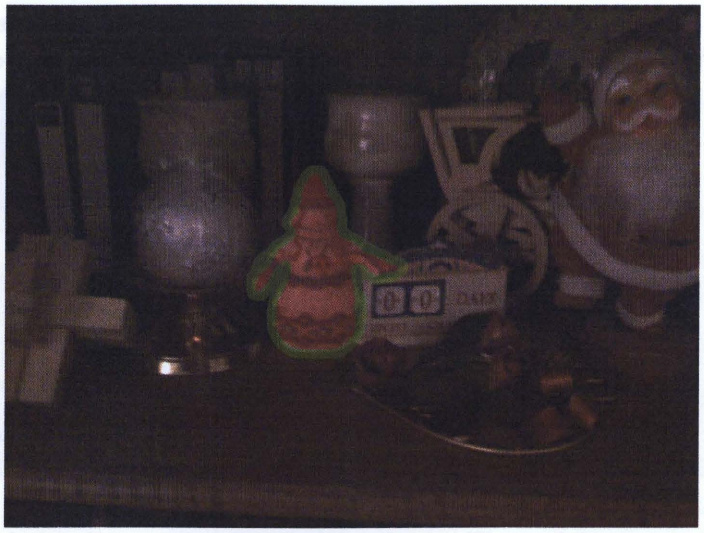
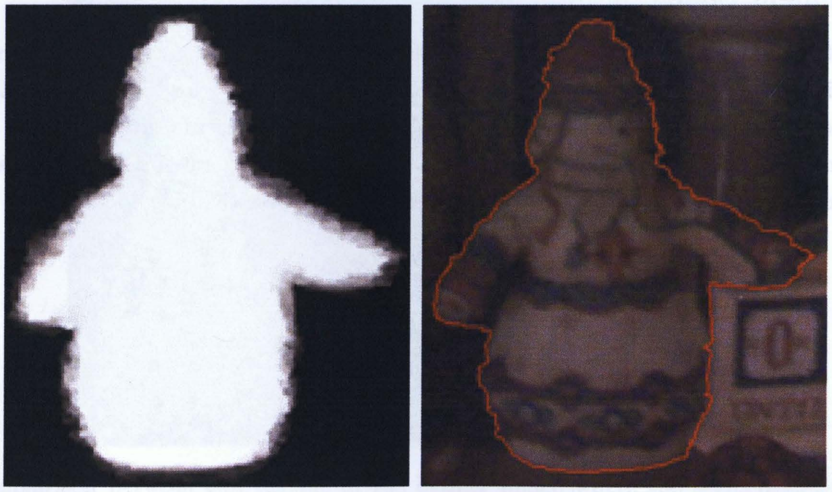


Figure 3.9: Photograph of some Christmas decorations sitting on a shelf, taken with a cellphone camera. The labelling used in the segmentation is overlaid with red representing the foreground and green representing the unknown region.

The labelling used for this segmentation is much more precise than the labelling in the banner image (Figure 3.6). The principle reason is that the edges are quite weak as a result of the poor lighting. The goal of this particular example is to show the effect of noise on the segmentation so the unknown region is much smaller, relative to the overall size of the image. This ensures that the boundary is somewhat accurate while still demonstrating how noise affects the segmentation.



(a) Probabilities

(b) Segmentation

Figure 3.10: Segmentation of the shelf image (Figure 3.9) with the corresponding probabilities. The images have been cropped for clarity.

The results of the segmentation, shown in Figure 3.10, are not particularly impressive. Because the weight calculation in the Random Walks algorithm is essentially an edge-detection operation, noise results in pixels being considered dissimilar that are not actually dissimilar. Noise is simply unwanted high-frequency information that comes from some external source. Figure 3.11 shows the segmentation outline and walker probabilities for one particularly noisy section of the image.

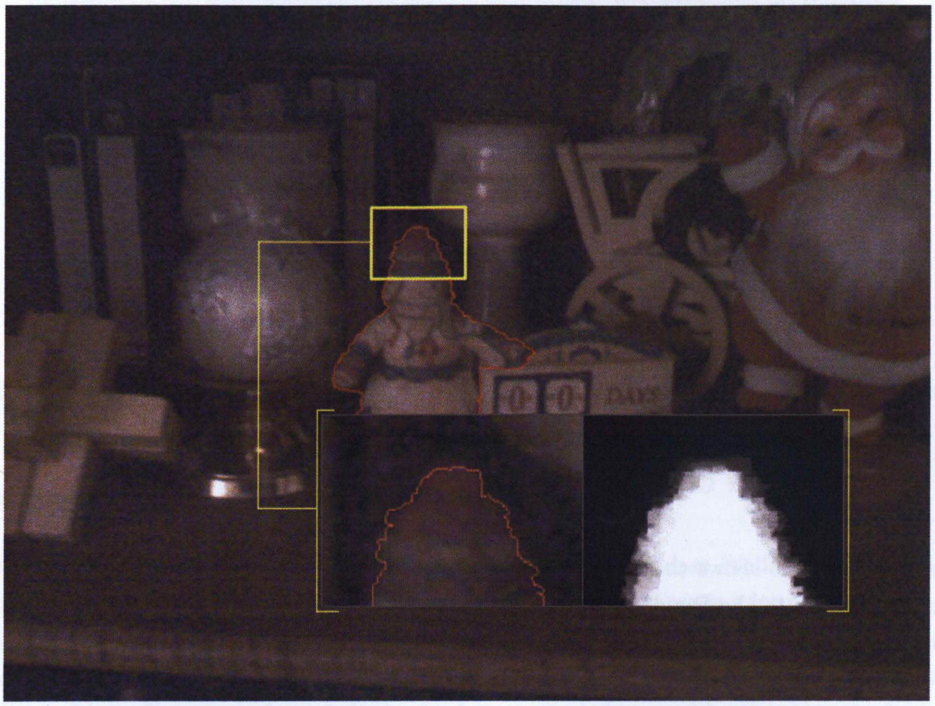


Figure 3.11: A close-up of one particularly noisy region of the shelf image. The jagged outline and the blocky probabilities show how the Random Walker is affected by heavy noise.

3.6 Comparison with Graph Cuts

For completeness, the two examples shown in 3.5.1 and 3.5.2 have been segmented using Graph Cuts through the algorithm provided by Olga Veksler [32, 33, 34]. The graph used in the segmentation is set up exactly as it was for the Random Walks segmentations. The unknown nodes are not assigned any affinity weighting to either of the labels in order to keep the comparison consistent. Furthermore, the inter-pixel weights remain the same (i.e. using a sigmoidal weighting function). The only difference is that the Graph Cuts algorithm is used rather than Random Walks. This provides a direct "algorithm-to-algorithm" comparison.

Figure 3.12 shows the results of the segmentation of the "Banner" image. The results are comparable to the Random Walks segmentation, with a few exceptions. The mask returned by Graph Cuts is much more severe than the Random Walks mask. What is meant by that is that the mask corners are much more abrupt and there are a lot more "thorns", i.e. very small, sharp offshoots from the main outline. However, the overall segmentation is very close. This is expected as the setup of the graph structure is identical.

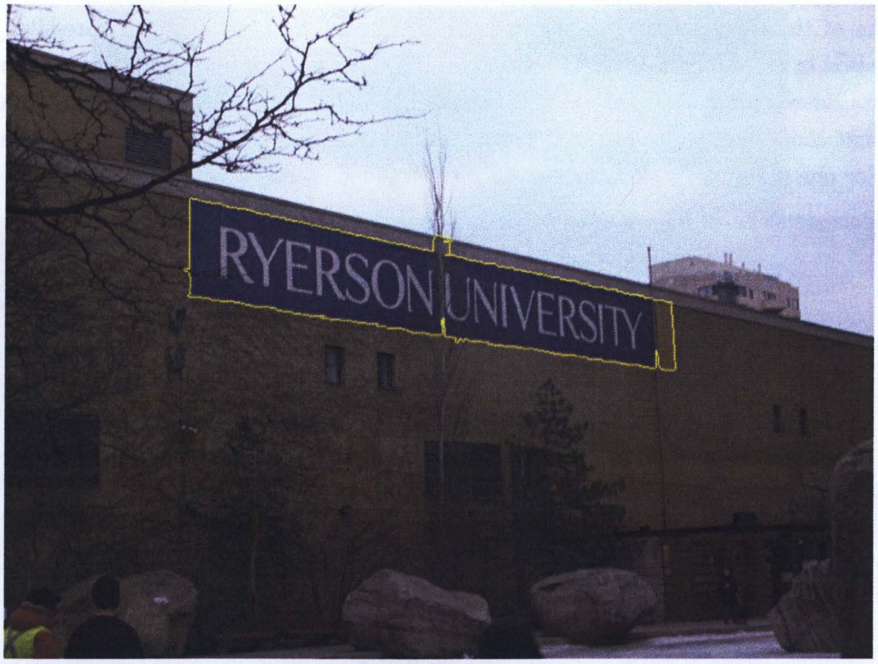


Figure 3.12: Graph Cuts segmentation of the image shown in Figure 3.7.

Similarly, Figure 3.13 shows the Graph Cuts segmentation of the "Basement Shelf" image. Again, the results are comparable to the Random Walks segmentation in Figure 3.10. However, the resulting outline is much more jagged than the Random Walks outline. This difference can be explained by the fact that Random Walks respects weak boundaries better than Graph Cuts [20] and, as such, handles noisy and ill-defined boundaries a little more gracefully. Since the two algorithms do essentially the same thing on the same graph structure, the results are comparable.

It should be mentioned that it is possible to improve Graph Cuts performance, but that requires the application of several types of heuristics to the algorithm and would invalidate the comparison. The purpose of this comparison was to only to show the relative differences between the two algorithms. Keeping the operating conditions of the two algorithms as close as possible allows the relative performance of each algorithm to be measured.

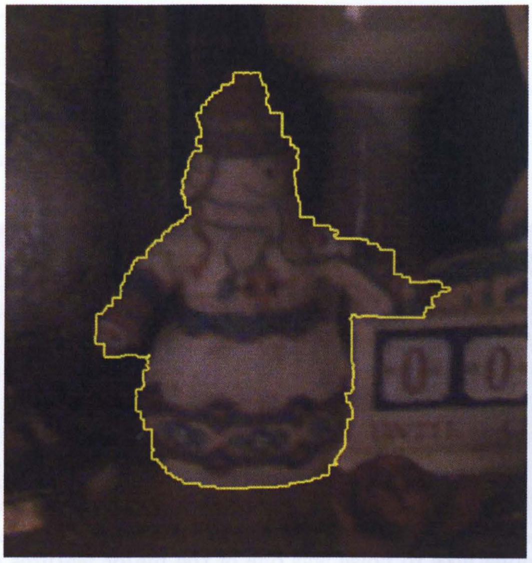


Figure 3.13: Graph Cuts segmentation of the image shown in Figure 3.10. The image has been cropped for clarity.

3.7 Issues with Random Walks

The two example images convey some of the issues that plague the Random Walks algorithm. Namely, the algorithm has difficulty when the edges are weak or broken and when there is a lot of texture or noise. The weak edge issue is one that is, for all intents and purposes, impossible to fix. Unlike a human, a computer has no way to effectively determine where an edge should be. As discussed in Chapter 1, a human can usually extract distinct objects from a scene, even if they've been occluded or are blending into the background. A computer would have to have some way of knowing how to separate the different objects in a scene before it could even being to try to deal with the weak edge problem.

The noise problem is much more tractable, though. Plenty of signal (and image) processing research has gone into removing noise from a signal. Generally, these methods rely on the fact that the noise is somehow different from the signal. System modelling techniques can be used to identify the statistics of the noise if there is some way to model the desired signal. Filtering techniques rely on the fact that most noise tends to be high-frequency and so they usually employ low-pass filters to retain all of the desired, low-frequency information.

Chapter 4

Scale-Space Random Walks

NOISE was identified as the biggest problem, aside from ill-defined edges, that affects Random Walks. Ironically, Random Walks' robustness to noise was one of the major features of the algorithm, as presented in [21]. This robustness arose not from how accurate the segmentation was, rather it arose from the fact that Random Walks will *always* provide a segmentation. The most important feature to Random Walks is that, unlike other algorithms, it will always return a segmentation, though not necessarily one of any real quality.

Finding a way to make Random Walks more resilient to noise and other undesired image features is *the* central goal of this thesis. Improving Random Walks so that it returns more faithful segmentations in noisy conditions would greatly expand the range of situations where the algorithm could be applied. Completely removing the effect of noise is an impossible task; it is only possible to mitigate it. Fortunately, as it has been shown, that is not necessary since the algorithm can handle *some* noise.

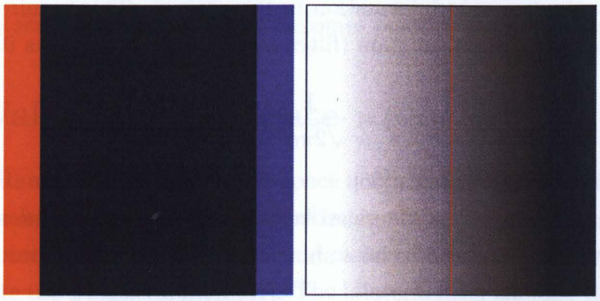
It was mentioned briefly in Chapter 2 that an extension to Random Walks has already been proposed in [22]. However, this modification significantly alters the algorithm by changing the graph edges from resistors to resistor-diode circuits. Singaraju *et al* also use an iterative method to generate asymmetric weights that are optimal for the given image. The result is a significant departure from the original algorithm that Grady had proposed but does prove to be more effective in a number of situations. Please refer to [22] for more information.

The major failing of this extension is that it can only be applied to grey-scale images. Here, the concept of "directionality" can be easily defined as an edge is simply a rapid change from a dark region to a light region. Traveling from a dark to light region is different than traveling from a dark to light region. Unfortunately, this cannot be easily translated to colour images and such this extension is limited to only grey-scale images. Singaraju *et al* restricted their algorithm to examples from medical imaging since those images are typically devoid of colour information (e.g. x-rays, ultrasound, Computer Aided Tomography and etc.).

The extension to Random Walks, as presented in this chapter, makes no assumptions on the image being segmented. There are no restrictions to whether or not the image is in colour or in grey-scale. Furthermore, this extension does not modify the core Random Walks algorithm. Rather, it modifies the *input* to the algorithm so that the algorithm itself is untouched and retains all of the properties that were defined in [21].

4.1 The Problem of Noise

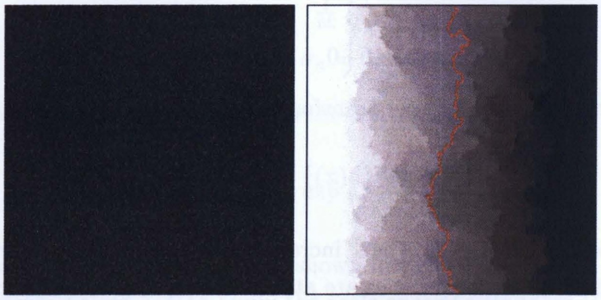
Consider Figure 4.1 below. It is the Random Walks segmentation of a blank, completely empty image. The resulting probabilities, as expected, from a smooth gradient from left to right, with the segmentation boundary exactly in the centre of the image.



(a) Blank Image With Labels (b) Resulting Probabilities

Figure 4.1: The Random Walks segmentation of a completely blank image. The resulting probabilities are the *ideal* segmentation of the image

Now consider Figure 4.2, where the image in Figure 4.1a has been seeded with Gaussian noise with a mean of 0 and a variance of 0.01. The resulting probabilities in Figure 4.2b are noticeably worse. Rather than being a smooth gradient from left to right, the probabilities drop in almost discrete steps. The segmentation boundary no longer runs straight through the centre of the image but is instead a jagged line.



(a) Noisy Image (b) Probabilities

Figure 4.2: A segmentation, using the same labels as Figure 4.1a, on an image seeded with Gaussian noise.

In trying to find a way to make Random Walks more robust against noise, a few requirements were identified. The first requirement is that structural information about the image is preserved. Simply blurring the image is not effective because that removes edge information and makes the segmentation less precise. Second, the new version should retain most, if not all, of the properties of the original Random Walks algorithm. Modifying the algorithm to make it more robust but losing its characteristic of acting locally is not desirable. Any new algorithm should act like the old algorithm, only providing a better segmentation when the image has a fair degree of noise.

4.2 Scale-Space

The solution to the noise problem turns out to be to use a scale-space. Scale-space is a form of multi-resolution signal analysis that can be used to examine the features of a signal (image) at various scales. The various scales are produced by successively filtering the signal through a series of low-pass filters in order to extract information about coarser and coarser scales [35]. An important condition of the low-pass filter is that it must not add any extraneous information in the process. This requirement results in a series of axioms that any scale-space generation function (filter kernel) must satisfy. As it so happens, the Gaussian kernel of the form

$$h(x|\sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-x^2}{2\sigma^2}\right) \tag{4.1}$$

is the *only* kernel that satisfies these axioms [36].

There are two ways to explain why a Gaussian kernel must be used as the scale-space generator. The first is that scale-space and be considered to be a simulation of the diffusion process, given by

$$\frac{\partial\phi(\vec{x},t)}{\partial t} = \frac{1}{2}\nabla^2\phi(\vec{x},t), \tag{4.2}$$

where $\phi(\vec{x},t)$ is the signal at time t. Because the signal can be described over multiple dimensions, such as an image, the parameter $\vec{x}$ is used to indicate this. Using the diffusion "analogy", the Gaussian kernel appears in the solution to (4.2). Therefore, a grey-scale image can be considered to be the initial temperature distribution at $t = 0$ and the scale-space describes how the temperature evolves as $t \to \infty$.

The other way of understanding why a Gaussian kernel is necessary is from a signal processing perspective. For a filter not add extraneous information, it must not include information from undesired frequencies. Consider the normalized box filter

$$h(x) = \begin{cases} \frac{1}{2a} & -a \le x \le a \\ 0 & \text{otherwise} \end{cases}, \tag{4.3}$$

where a is the width of the filter. It's Fourier transform will be

$$H(\omega) = \mathcal{F}\{h(x)\} = \frac{1}{|a|}\text{sinc}\left(\frac{\omega}{2\pi a}\right). \tag{4.4}$$

The sinc function both oscillates and decays as ω increases. As a result, all frequencies at a multiple of a are *completely* suppressed but all other frequencies are attenuated. Because the function decays, it does act as a LPF but the periodic frequency rejection and the attenuation in between causes information from higher frequencies to "leak through".

The Gaussian function is different; it has the unique property of being its own Fourier transform. For any function in the form

$$h(x) = \exp\left(-ax^2\right), \tag{4.5}$$

its Fourier transform is

$$H(\omega) = \mathcal{F}\{h(x)\} = \sqrt{\frac{\pi}{a}}\exp\left(-\frac{\omega^2}{4a}\right). \tag{4.6}$$

Effectively, the function is its own Fourier transform (i.e. it is self-reciprocal). For both functions, $h(x)$

and $H(\omega)$ decrease monotonically as $|x| > 0$ and $|\omega| > 0$, respectively. Therefore, unlike the box filter, the Gaussian filter does not allow high frequency information to leak through into the filtered signal.

In the end, filtering a signal by a Gaussian will remove *all* of the high frequency information beyond a designated frequency (controlled by σ). Therefore, this filtering process gradually removes the high-frequency, or fine, detail while retaining the overall structure. The greater the value of σ, the more coarse the image representation because only the large structures, i.e. blobs, will still be present after the filtering. By producing a number of scale representations, it becomes possible to build up a scale-space.

4.3 Random Walks in Scale-Space

The motivation for using Random Walks on a scale-space is fairly straightforward. Random Walks relies on the differences between pixels (i.e. edges) in order to estimate a pixel's foreground/background membership. Simply blurring an image destroys these differences and makes the image more and more homogeneous.

One solution would be to use a bilateral filter [37]. The bilateral filter changes the filtering kernel so that it respects strong edges in an image while smoothing out more noisy regions. However, this means that quality of the Random Walks segmentation is now dependent on the quality of the bilateral filter implementation. Furthermore, the bilateral filter may unintentionally blur an edge crucial to the segmentation, degrading the final results.

In any case, the best solution would be to preserve as many of the image features as possible. This means trying to avoid removing information from the image or adding extra information that was not originally present in the image. This type of problem seems very well suited to a scale-space since the purpose of a scale-space is to avoid adding extra information and preserving existing image information.

Before the segmentation is performed, the first step is to generate a N-level scale-space, $\mathcal{S}$, for the image, I, such that

$$\mathcal{S} = \{I[x,y] * h[x,y|\sigma_n] : \sigma_n = 0, 1, 2, 4, \ldots, 2^{N-1}\}, \tag{4.7}$$

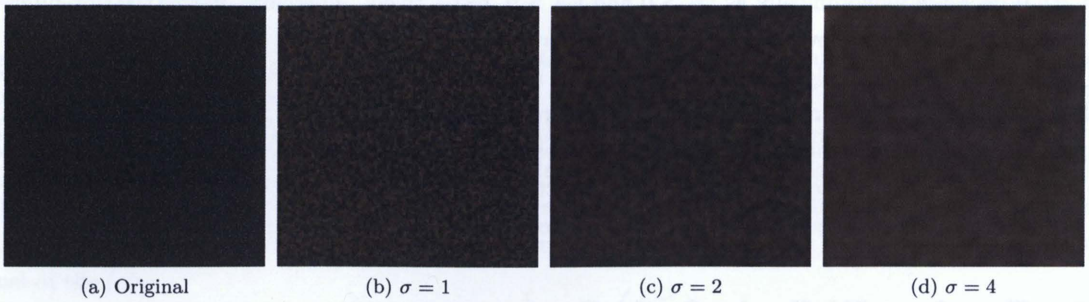
where '$*$' is the convolution operator and $h[x,y|\sigma_n]$ is the convolution kernel

$$h[x,y|\sigma_n] = \frac{1}{2\pi\sigma_n^2}\exp\left(-\frac{x^2+y^2}{2\sigma_n^2}\right). \tag{4.8}$$

The scale-space for the noisy image in Figure 4.2a is shown in Figure 4.3.

However, it is not necessarily clear *how* to apply the scale-space to Random Walks. Intuitively, Random Walks should be performed on the scale-space and that the graph structure should somehow incorporate the scale-space. There are two basic methods of applying the scale-space that can be identified: a per-level segmentation or a "volumetric" segmentation.

The per-level segmentation treats each level independently of one another. Random Walks is run separately on each level to produce a "probability scale-space", $\mathcal{P} = \{P[x,y|n] : n = 0, 1, 2, \ldots, N-1\}$, where $P[x,y|n]$ is the probability mapping for the n-the scale. The other option constructs a six-connected graph where each pixel connects to its corresponding neighbors in the scale-space. The labelling remains the same for each scale and Random Walks is performed on the *entire* three-dimensional graph. The result is also a probability scale-space, $\mathcal{P}$.


(a) Original

(b) $\sigma = 1$

(c) $\sigma = 2$

(d) $\sigma = 4$

Figure 4.3: Scale-space for the noisy image shown in Figure 4.2a with $\sigma = \{0, 1, 2, 4\}$. The images have been contrast-enhanced to better bring out the structures in the image (the actual segmentation was done on the unprocessed images).

For both methods, the scales are combined using a geometric mean so that the final probability mapping, $P[x, y]$ is

$$P[x,y] = \left(\prod_{i=0}^{N-1} P[x,y|i] \right)^{1/N} \quad . \tag{4.9}$$

The geometric mean was chosen, as opposed to the arithmetic mean, because of it's ability to suppress outliers more effectively than the arithmetic mean. Specifically, it was more important that the probability at a point, $P[x, y]$, converge to the value at the majority of the scales than the average value of all of the scales. This can be justified through the observation that $P[x, y|k] \leq 1 \, \forall \, k$.

For example, the probability at any particular point at the topmost scale, $P[x, y|N - 1]$, will be greater than the probability at a point at a lower scale, $P[x, y|N - k]$, where $k < N - 1$. This follows naturally from the effect of weak edges on the Random Walks algorithm. Therefore, the contribution of $P[x, y|N - 1]$ should be less than that of $P[x, y|N - k]$. Fortunately, since $P[x, y|i] \in [0, 1] \forall \, x$ and y, multiplying the values together does just that by resulting in $P[x, y] \to 0$ if the majority of the values for $P[x, y|N - k] \ll 1$. By taking the n-th root of the product, the final probability map is produced.

4.3.1 Per-level Segmentation

The simplest method of producing the scale-space random walks is to perform Random Walks on each each level of the scale-space separately. This method treats each level of the scale-space independently, effectively ignoring the fact that the levels *are not* independent. The principle advantage to this method is that the resources required to perform this segmentation are the same as performing Random Walks on the original image. However, because the levels are being treated separately, it becomes possible to perform the per-level segmentations in parallel.

Unfortunately, the cost of overlooking the dependance of each level on the previous level can have serious consequences, as illustrated in Figure 4.4. This demonstrates the "failure case" for this particular approach. The example image happens to be a 256x256 pixel image and, for $\sigma = 4$, the width of the filter used to produce the scale is $6\sigma + 1$ or 25 pixels. This means that the largest features in the image happens to be one-tenth the size of the image, producing a large number of false contours in the final probability map, as seen in Figure 4.5.

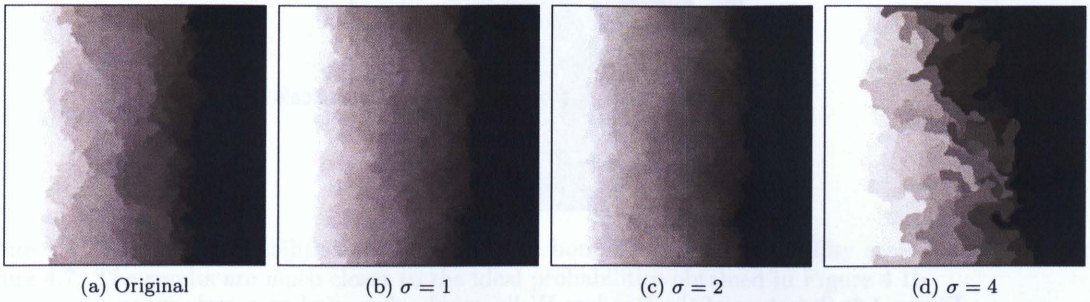

(a) Original (b) $\sigma = 1$ (c) $\sigma = 2$ (d) $\sigma = 4$

Figure 4.4: The resulting scale-space produced by applying Random Walks to each level of the scale-space shown in Figure 4.3.

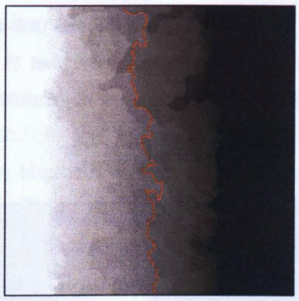


Figure 4.5: The resulting segmentation and probabilities for Figure 4.4. The "error" in the probabilities for $\sigma = 4$ result in an erroneous probability map and corresponding segmentation boundary.

Therefore, the simplest solution, a per-level segmentation, does not work. This is especially true if the blurring filter becomes very wide and therefore produces artificial structures in the probability maps. The algorithm does a very good job as picking up blobs in the image, particularly if they are visually distinct. Because the features are relatively large, they produce artificial structures in the probability map, which is not consistent with the definition of a scale-space. The returned probabilities must respect the scale-space axioms in order to the algorithm to operate properly.

4.3.2 Volumetric Segmentation

As demonstrated in 4.3.1, ignoring the relationship between scales results in artifacts appearing the probability maps of the coarser scales. Therefore, the relationship between the scales must be taken into account. Each scale, with the exception of the original image, is *directly* related to the previous scale in the space; this can be understood by considering the analogy with diffusion. Through a simple modification of the graph, as shown in Figure 4.6, this relationship can be accommodated. As before, the labelling is applied to each level so that the unknown nodes form a volume in the scale-space.

Each node at scale k is linked to it's corresponding node at scale $k-1$ and $k+1$. This allows the filtered, relatively noise-free, coarse scales to influence and mitigate the effect of noise at the finer, more

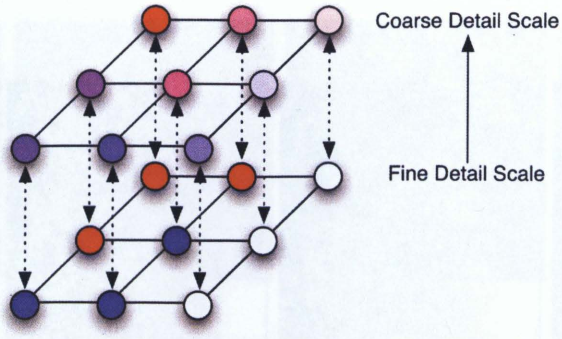


Figure 4.6: Structure of the Random Walks graph when used on a scale-space.

noisy scales. Conversely, fine detail information propagates to the coarse scales and prevents the situation where imaginary structures appear due to the blurring process. This type of segmentation is referred to as a "volumetric" segmentation because the unknown region carves out a *volume* in scale-space. Figure 4.7 shows the probabilities returned by using the modified graph structure. As a result, the combined probabilities are much cleaner and more precise. The probability map and segmentation boundary, shown in Figure 4.8, are very close to the ideal segmentation in Figure 4.1b.

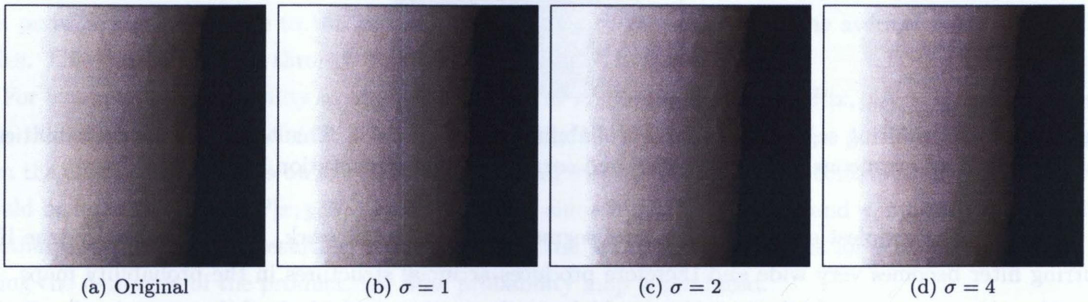

(a) Original (b) $\sigma = 1$ (c) $\sigma = 2$ (d) $\sigma = 4$

Figure 4.7: Volumetric segmentation of the scale-space presented in Figure 4.4. There are no more artifacts at the higher scales and they appear much more uniform.

By linking the layers together, important structural information is propagated upwards (from finer to coarser) in the scale-space while the filtered coarse scales suppress the noise at the finer scales. This results in a substantially larger system then the original Random Walks, but without changing the overall properties of the algorithm. This is still the solution to a linear system; it is merely performed on a 3D graph, rather than a 2D graph. All of the properties of Random Walks are retained with the added benefit of being able to better cope with noise in the signal. It is this graph structure that is referred to as "Scale-Space Random Walks" (SSRW).

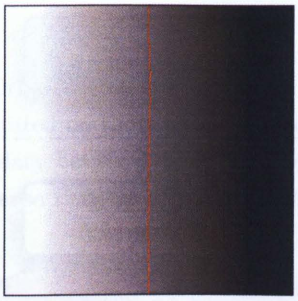


Figure 4.8: The final probabilities and segmentation boundary of the probability scale-space generated in Figure 4.7. The results are much closer to the ideal probabilities obtained in Figure 4.1b

4.4 Algorithm

The complete SSRW algorithm is a straightforward modification to the Random Walks algorithm. The core algorithm, the linear system solver, remains the same. However, a "pre-processing" stage is added where the scale-space is generated and the system is assembled before being passed to the solver. A "post-processing" stage then takes the probability scale-space, $\mathcal{P}$, and generates the probability map, $P[x, y]$ using (4.9). Figure 4.9 show the various stages of the system.

From a user-interaction point of view, the SSRW algorithm is identical to Random Walks. The algorithm takes two inputs, the image and the labelling, and produces a probability image that can then be used to find the segmentation boundary. Because Random Walks has no *a priori* knowledge of the graph that it operates on (this is determined when the system matrix, $\mathbf{L}$, is generated), no modification of the original Random Walks code has to be made. All that is required is to add the system generation and the geometric average portion.

It is clear from Figure 4.9 that the algorithm's major bottleneck is the linear system solver. The method used to solve $\mathbf{L}\vec{x} = \vec{b}$ is entirely dependent on the application. Some solvers are very fast but require very large amounts of memory to run while other solvers can run under very constricted memory conditions but are much slower. The scale-space filtering and final probability generation are both very fast operations. As such, the choice of the solver can dictate how fast the algorithm will run. Chapter 5 will examine these topics in more detail.

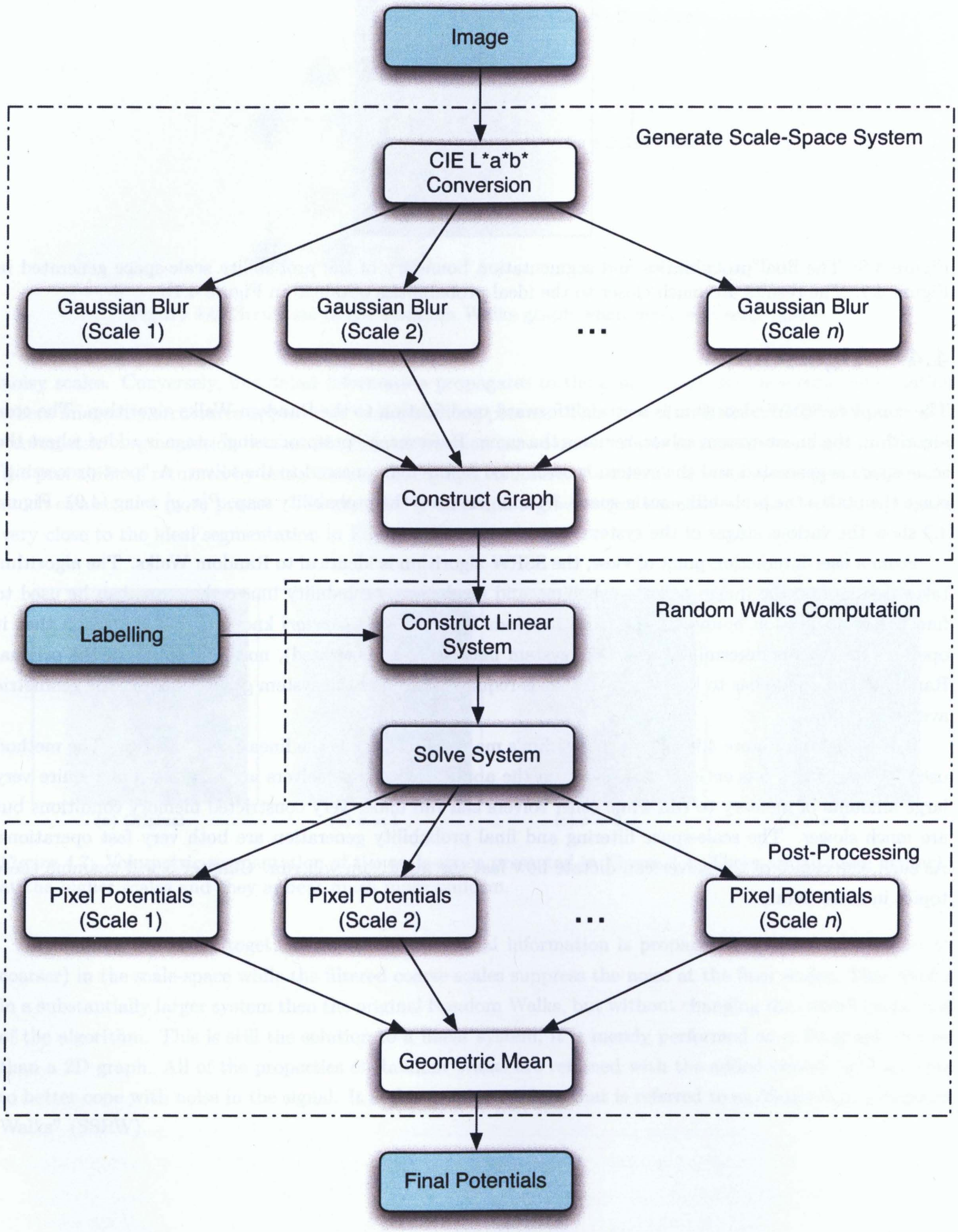


Figure 4.9: A flowchart showing the individual stages of the SSRW algorithm.

4.5 Noise Performance

A novel feature of the SSRW is that, with one exception, it does not care what kind of noise is present in the image. The effects of noise are demonstrated on the artificial image in Figure 4.10. This two colour image has sharp edges with a very clear boundary between them. Because this image is so simple, the resulting segmentation will be "perfect" in the sense that the boundary between the two colours will be identified exactly.

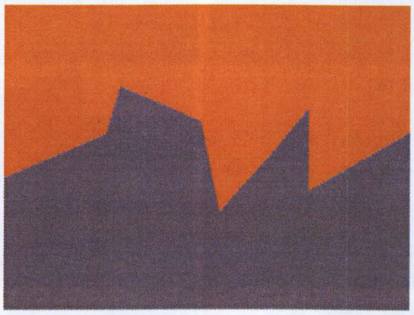


Figure 4.10: The artificial test image that will be used to demonstrate the robustness of the SSRW to noise.

On a noise-free image, the Random Walks and SSRW segmentations are *almost* identical (Figure 4.11). The primary difference between the two is that Random Walks tends to respect the sharp edges better than the SSRW. This is understandable since the SSRW is essentially a low-pass filter. Some edge information, particularly sharp corners, will be lost. However, the overall loss is negligible.

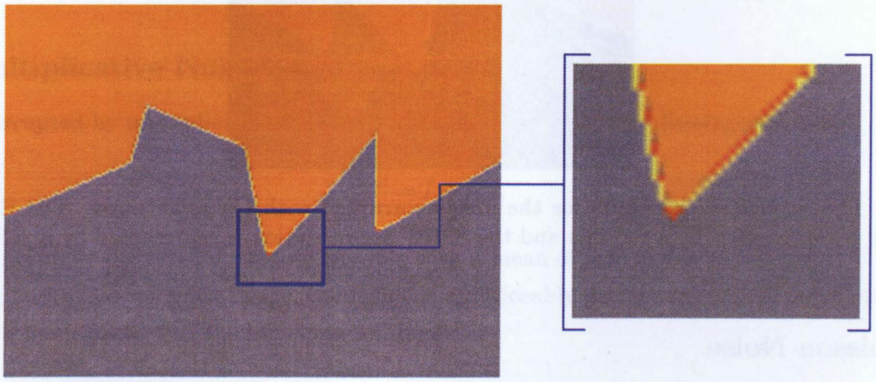


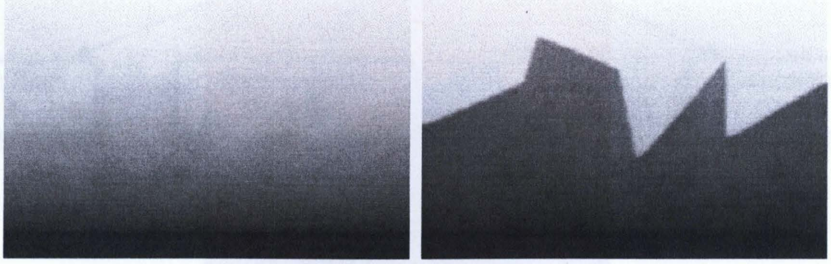
Figure 4.11: A comparison between the Random Walks segmentation and the SSRW segmentation on a synthetic image. The image has no noise and very sharp, well-defined, edges.

4.5.1 Gaussian Noise

For the initial test, the image was corrupted by additive Gaussian noise with a mean of zero and a variance of 0.01. Therefore, if $\mathcal{X}$ represents a Gaussian distributed random variable with $N(0, 0.01)$ then the corrupted image is

$$I' = I + \mathcal{X}. \tag{4.10}$$

These are the same parameters for the noise image shown in Figure 4.2a. However, rather than segmenting an information-less image, this test image has a well-defined boundary. As expected, the SSRW far outperforms Random Walks in the quality of the segmentation.



(a) Random Walks

(b) SSRW

Figure 4.12: The returned probabilities for the test image corrupted with Gaussian noise.

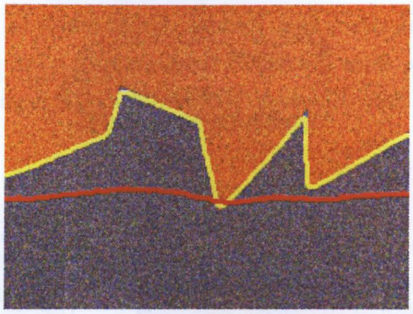


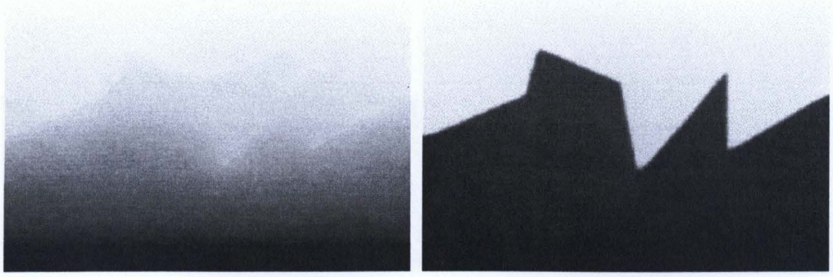
Figure 4.13: The segmentation results for the image corrupted with Gaussian noise. The Random Walks segmentation is represented by a red line and the SSRW segmentation is represented by a yellow line.

4.5.2 Poisson Noise

Poisson noise is any noise that is distributed according to the Poisson distribution, which defined as

$$f(x|\lambda) = \frac{\lambda^x}{x!}e^{-\lambda}. \tag{4.11}$$

The λ term is both the mean and variance of the distribution. The MATLAB `imnoise` function, which was used to generate the noisy image, bases the value of λ on the value of the current pixel. For an RGB image, each of the three colour channels is treated independently. As with the Gaussian noise, the SSRW outperforms Random Walks. Again, like the Gaussian noise, it was additive.



(a) Random Walks (b) SSRW

Figure 4.14: The returned probabilities for the test image corrupted with Poisson-distributed noise.

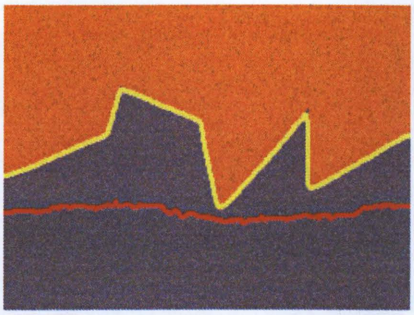


Figure 4.15: The segmentation results for the image corrupted with Poisson noise. The Random Walks segmentation is represented by a red line and the SSRW segmentation is represented by a yellow line.

4.5.3 Multiplicative Noise

The image corrupted by multiplicative noise was generated through the following operation:

$$I' = I + \mathcal{X}I, \tag{4.12}$$

where $\mathcal{X}$ is a normally distributed random variable with a mean of zero and a variance of 0.04. Because the noise is being multiplied with the image, the result is a noticeably greater amount of corruption. However, the SSRW still manages to find the segmentation boundary.

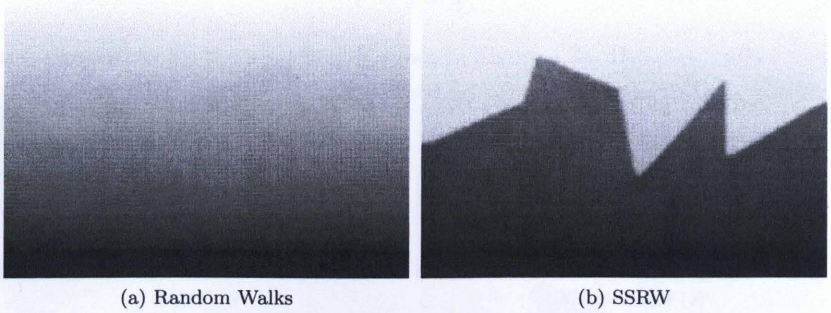


Figure 4.16: The returned probabilities for the test image corrupted with normally-distributed multiplicative noise.

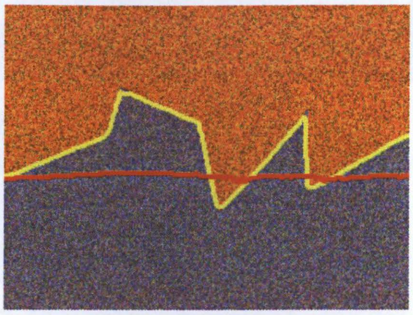


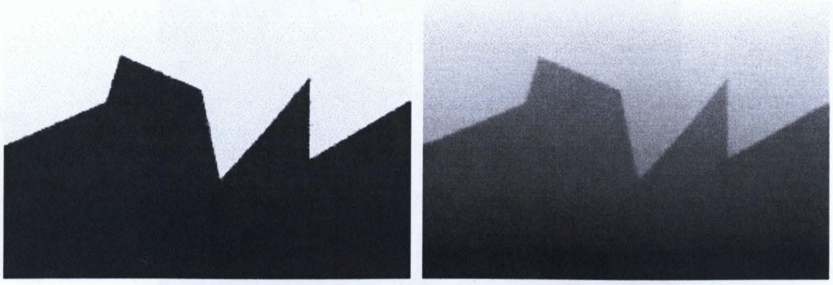
Figure 4.17: The segmentation results for the image corrupted with normally-distributed multiplicative noise. The Random Walks segmentation is represented by a red line and the SSRW segmentation is represented by a yellow line.

4.5.4 Impulsive Noise

Impulsive noise is really the only type of noise that will cause problems for the SSRW. In fact, impulsive noise is the only instance where Random Walks performs *better* in the presence of noise than the SSRW. Figures 4.18 and 4.19 demonstrate this fact.

Impulsive noise is noise where the state of pixel is randomly changed to either "on" or "off" with equal probability. For a colour image, the individual colour channels are corrupted independently so that for a particular pixel $\vec{p}[x, y]$, the probability of any particular colour component, $p_c[x, y]$, where c is the colour channel, being 0 or 1 is 50%. This randomness means that each corrupted pixel is, for all intents and purposes, unique.

Because the state of each pixel is assigned randomly, a corrupted pixel is completely *dissimilar* from its neighbors. The result is that, more often than not, the weighting for the corrupted pixel, i, to a neighbor, j, will be $G_{i,j} \approx 0$. The basic Random Walks algorithm will ignore this node because the effect is that of completely removing the node from the graph and, as such, any disturbance by the impulsive noise. However, the nature of the SSRW makes it susceptible the the effects of this noise, as will be explained in 4.7.



(a) Random Walks

(b) SSRW

Figure 4.18: The returned probabilities for the test image corrupted with impulsive noise.

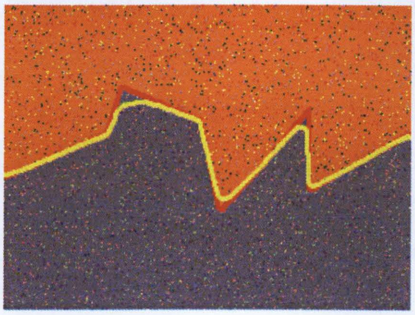
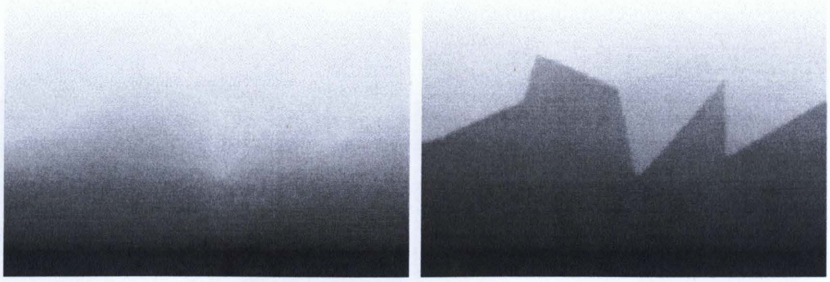


Figure 4.19: The segmentation results for the image corrupted with impulsive noise. The Random Walks segmentation is represented by a red line and the SSRW segmentation is represented by a yellow line.

4.5.5 Mixed Noise

This following example (Figures 4.20 and 4.21) is a combination of the Gaussian noise from 4.5.1 and the impulsive noise from 4.5.4. In effect, this image has been corrupted twice by two very distinct types of noise. The result is that while the impulsive does in fact cause the SSRW to fail, it is still able to more effectively find the segmentation boundary than Random Walks. Therefore, the limiting factor in this particular example is the impulsive noise and *not* the Gaussian noise.

What is important to note is that the SSRW fails *gracefully*. The algorithm is still able to find a segmentation boundary; not a very good boundary but still significantly better than the result returned by Random Walks. This is useful in cases where the algorithm should try to return a "reasonable" result, which usually means to provide a segmentation through the centre of the unknown region. In this instance, even though the impulsive noise has an extreme effect on the SSRW, it can still recover from that and provide a semi-accurate segmentation.



(a) Random Walks (b) SSRW

Figure 4.20: The returned probabilities for the test image corrupted with both Gaussian *and* impulsive noise.

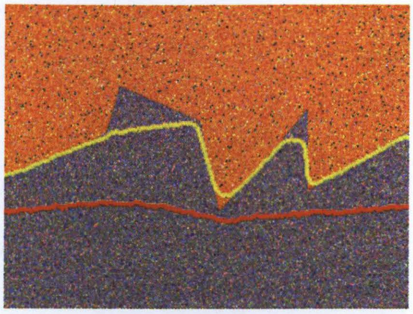


Figure 4.21: The segmentation results for the test image corrupted by both Gaussian and impulsive noise. The Random Walks segmentation is represented by a red line and the SSRW segmentation is represented by a yellow line.

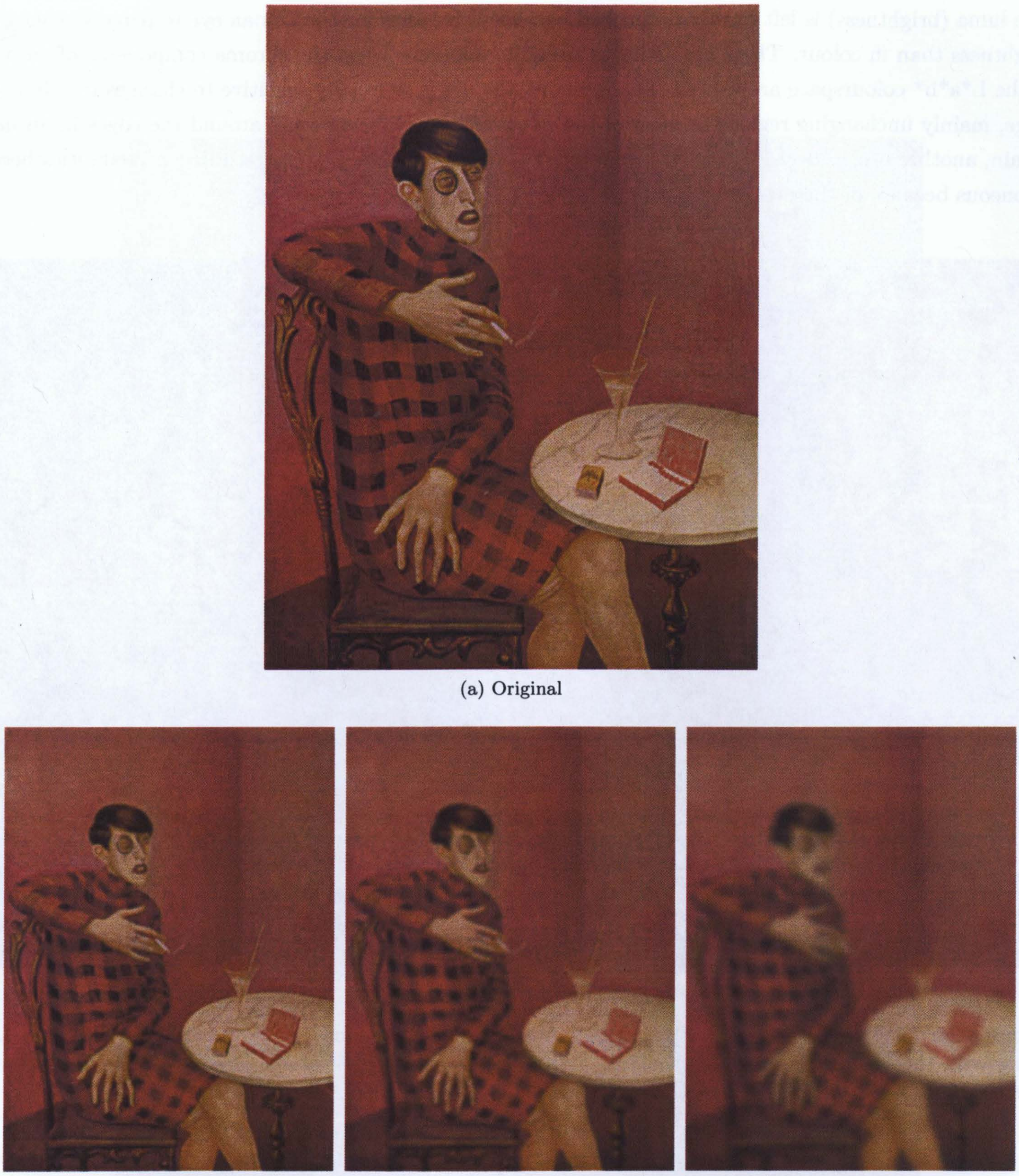
4.6 More Example Images

Just as in Chapter 3, a number of "real-world" example images will be provided to show the performance of the SSRW in a number of situations. These examples will compare the SSRW to Random Walks and show how the SSRW improves Random Walks performance in the presence of noise. For all of the examples, the scale-space used will be $\sigma = \{0, 1, 2, 4\}$. While this scale-space may not be appropriate for *all* images, it provides a standard for comparing the results and demonstrates the effect of the choice of scales on the segmentation.

4.6.1 Portrait of the Journalist Sylvia von Harden (1926)

The first examples is a painting by German artist Otto Dix entitled, "Portrait of the Journalist Sylvia von Harden" (1926). This image suffers from significant JPEG compression artifacts that *are not* immediately visible to the naked eye. These artifacts do, however, cause problems for the Random Walks segmentation. Figure 4.22 shows the scale-space that Random Walks will be operating on.

For reference, the Random Walks segmentation of the image is provided in Figure 4.23 to show the effects of the compression/quantization artifacts on the resulting probabilities. The segmentation boundary



(a) Original

(b) $\sigma = 1$

(c) $\sigma = 2$

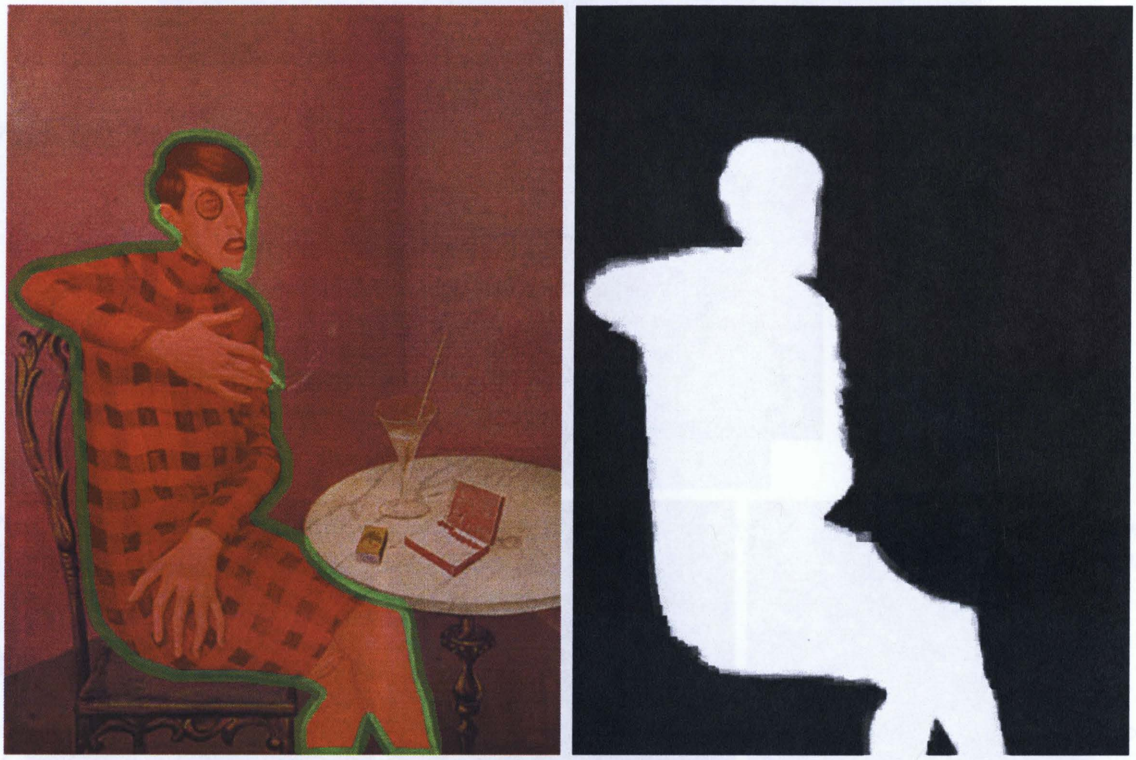
(d) $\sigma = 4$

Figure 4.22: Scale-space of the Otto Dix painting “Portrait of the Journalist Sylvia von Harden”. The set of scales used is $\sigma = \{0, 1, 2, 4\}$.

is shown in Figure 4.24, along with close-ups of some of the more affected regions. Note that while JPEG artifacts are difficult to see, they create blocky structures in the probability map. These structures are essentially false edges created by the JPEG quantization process.

By design, JPEG compression hides much of the artifacts in the chroma (colour) components of an image.

The luma (brightness) is left mostly untouched because it is easier for the human eye to perceive changes in brightness than in colour. These are artifacts are quite apparent when the chroma components of an image in the L*a*b* colourspace are viewed. Furthermore, the eye is especially sensitive to changes in colour over large, mainly unchanging regions so most of the compression artifacts occurs around the edges in an image (again, another design decision). Unfortunately, by corrupting the edges, the resulting probabilities become erroneous because of the artifacts (visible in Figure 4.24).


(a) Labelling

(b) Probabilities

Figure 4.23: The labelling used for the Random Walks segmentation of the painting shown in Figure 4.22a, along with the resulting probabilities. The painting has a high degree of JPEG compression artifacts that affect the probabilities returned by the algorithm.

Using a scale-space does a very good job of mitigating the artifact problem by blurring the false edges away. The false edges tend to be very small small structures because the JPEG compression block size is only 8x8 pixels. In the vast majority of JPEG compressed images, a single block comprises only a tiny portion of the image's overall area. The errors themselves occur at the block boundaries, structures that are only two-pixels across. Effectively, all of these artifacts are completely gone by the second or third scale due to the blurring process. The probability scale-space for the painting is shown in Figure 4.25 and the final result is shown in Figure 4.26.

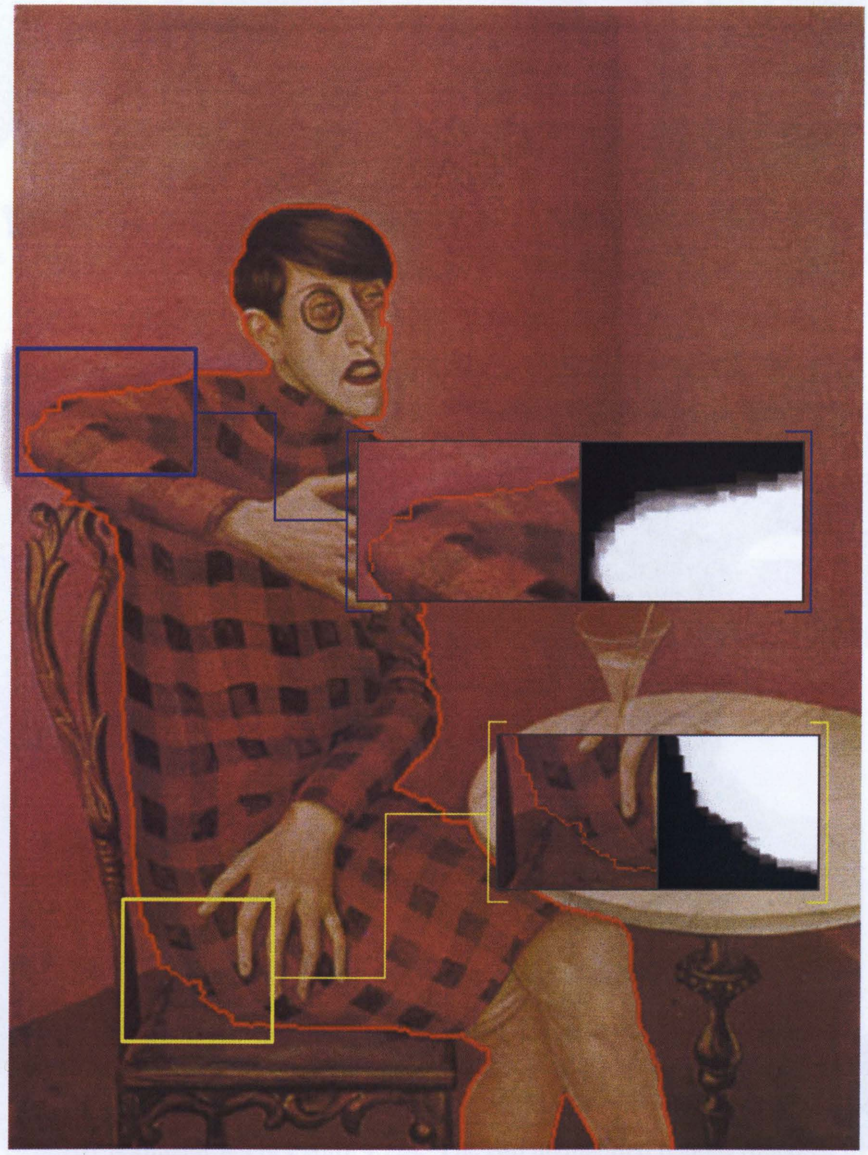
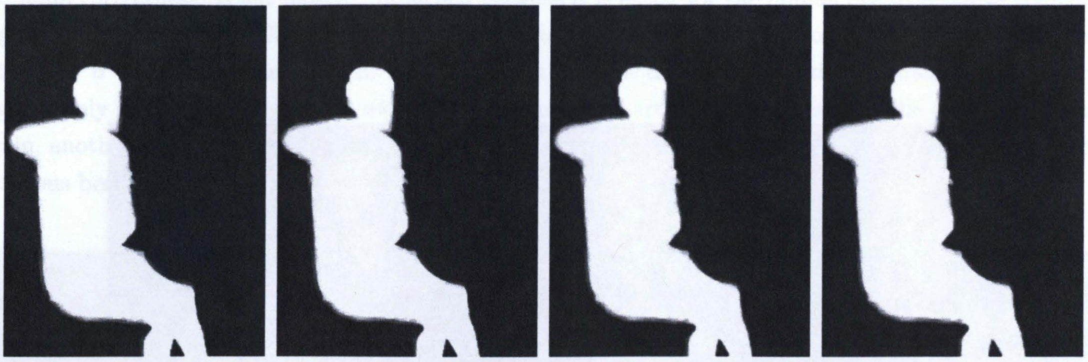
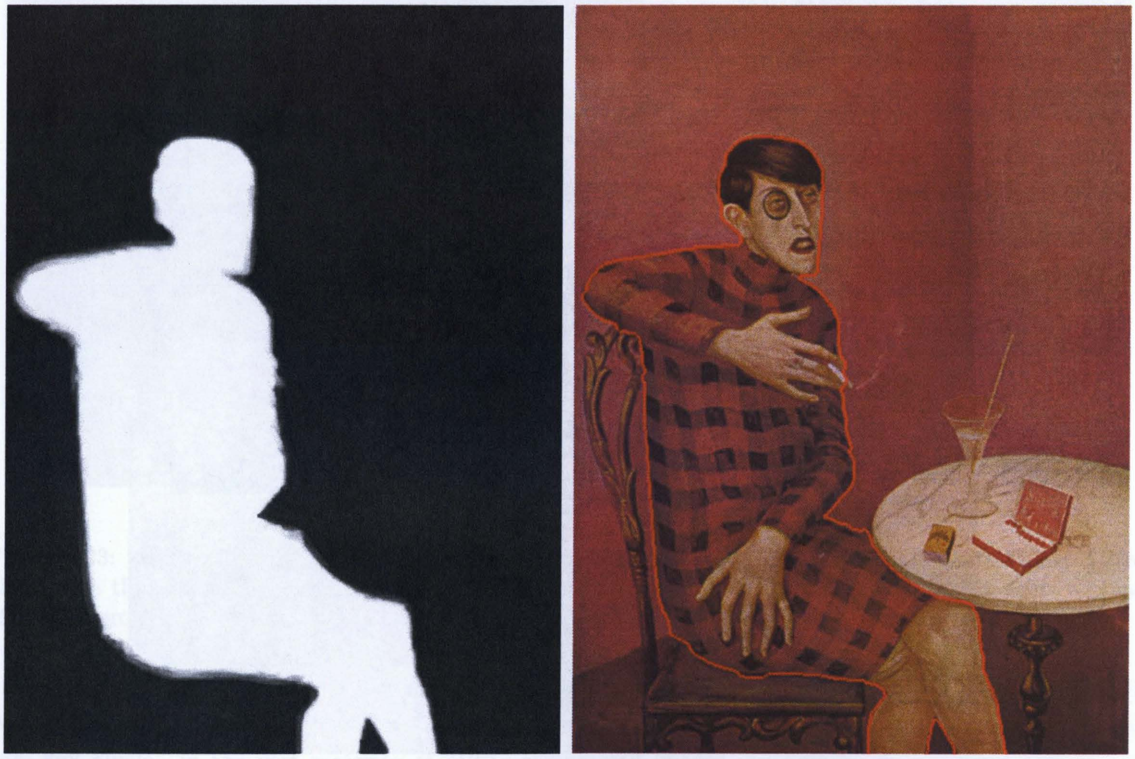


Figure 4.24: The Random Walks segmentation of Figure 4.22a. The segmentation boundary is a result of the probabilities that are displayed in Figure 4.23. The two regions (denoted by the blue and yellow rectangles) show how the JPEG compression results in a “blocky” probability map.



(a) Original (b) $\sigma = 1$ (c) $\sigma = 2$ (d) $\sigma = 4$

Figure 4.25: SSRW probability scale-space generated for the original image scale-space in Figure 4.22.



(a) Probabilities (b) Segmentation

Figure 4.26: The final probability map produced from the scale-space shown in Figure 4.25.

A close-up comparison of the two regions originally shown in Figure 4.24 is shown in Figure 4.27. The probability map is noticeably cleaner than its equivalent Random Walks version. The result is that the segmentation boundary is much smoother and cleaner. Unfortunately, an unintended consequence of the scale-space is that it can exaggerate weak edges, resulting in the segmentation boundary being erroneously placed. However, *on average*, the result is much better than the original Random Walks segmentation.

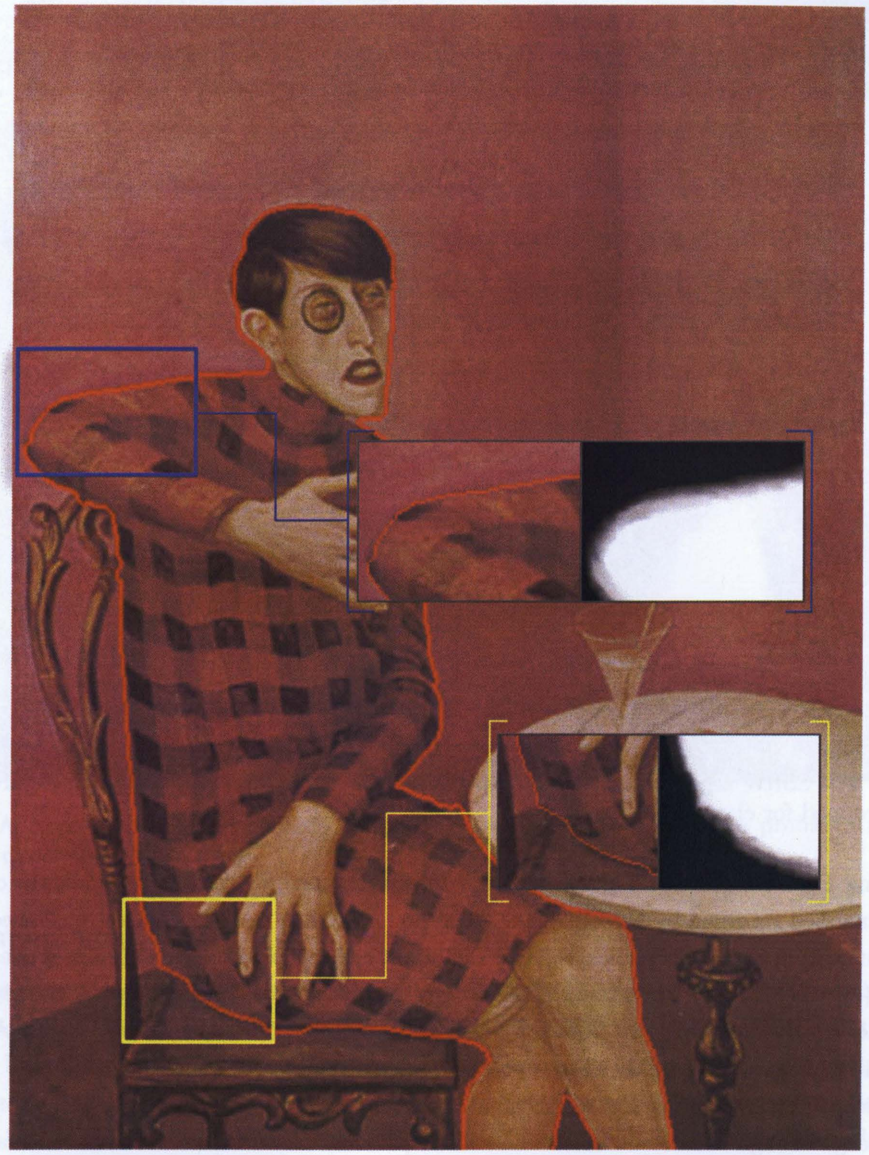
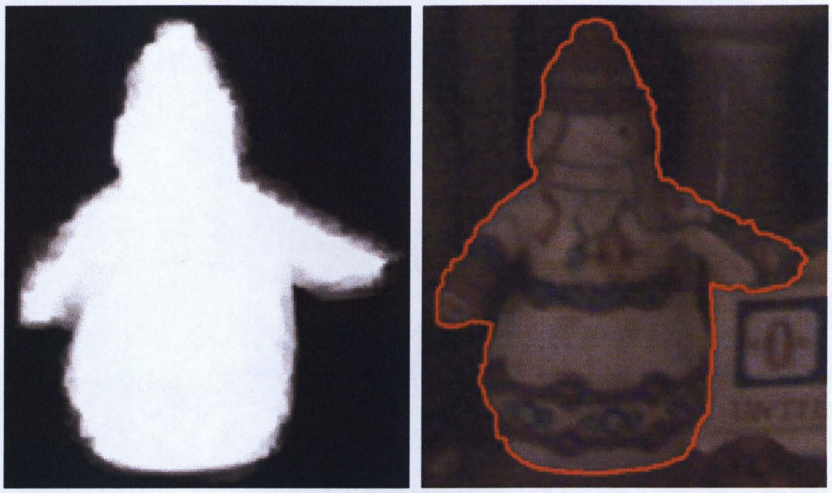


Figure 4.27: The SSRW segmentation of Figure 4.22a. The two highlighted regions (denoted by the blue and yellow rectangles) show how the SSRW improves on the original segmentation. In comparison with Figure 4.24, the results are much cleaner and more precise.

4.6.2 Basement Shelf

The shelf image, shown back in Chapter 3.5.2, was identified as a very poor-quality image. The results of the Random Walks segmentation (Figure 3.10) were equally as poor. While Random Walks was able to identify a very rough segmentation boundary, the boundary itself was very rough and inaccurate. The results of a SSRW segmentation, shown in Figure 4.28, are much better. The scale-space does a fairly good job at removing the compression artifacts and the camera sensor noise that arose from poor lighting. However, note that the boundary still is not without *any* error and it is still somewhat jagged.



(a) Probabilities

(b) Segmentation

Figure 4.28: The SSRW segmentation of the basement shelf image originally shown in Figure 3.10. Images have been cropped for clarity.

Comparing Figure 3.11 with Figure 4.29, it is clear that the SSRW has significantly cleaned up the probability mapped returned by the algorithm. The boundary has not been perfectly cleaned up, however, and this results from the choice of scales. The shelf image is a 1600x1200 pixel image while, for the set of scales used ($\sigma = \{0, 1, 2, 4\}$), the largest filter is only 25-pixels wide. Therefore, the blurring at even the coarsest scale is not that strong and some of the noise is still present at the coarser scales. As a result, the final probability map is still slightly noisy.

The choice of scales can be somewhat of a tricky problem. As demonstrated, if the scales are not sufficiently large enough then some noise will still be present. However, if there are too many scales then the system may be too large to store in memory and it will take a relatively long time to solve. Conversely, if the scales are chosen too far apart, the SSRW will become less effective. This results from the inter-scale weights becoming smaller (less similar) and making the random walker less likely to go between scales.

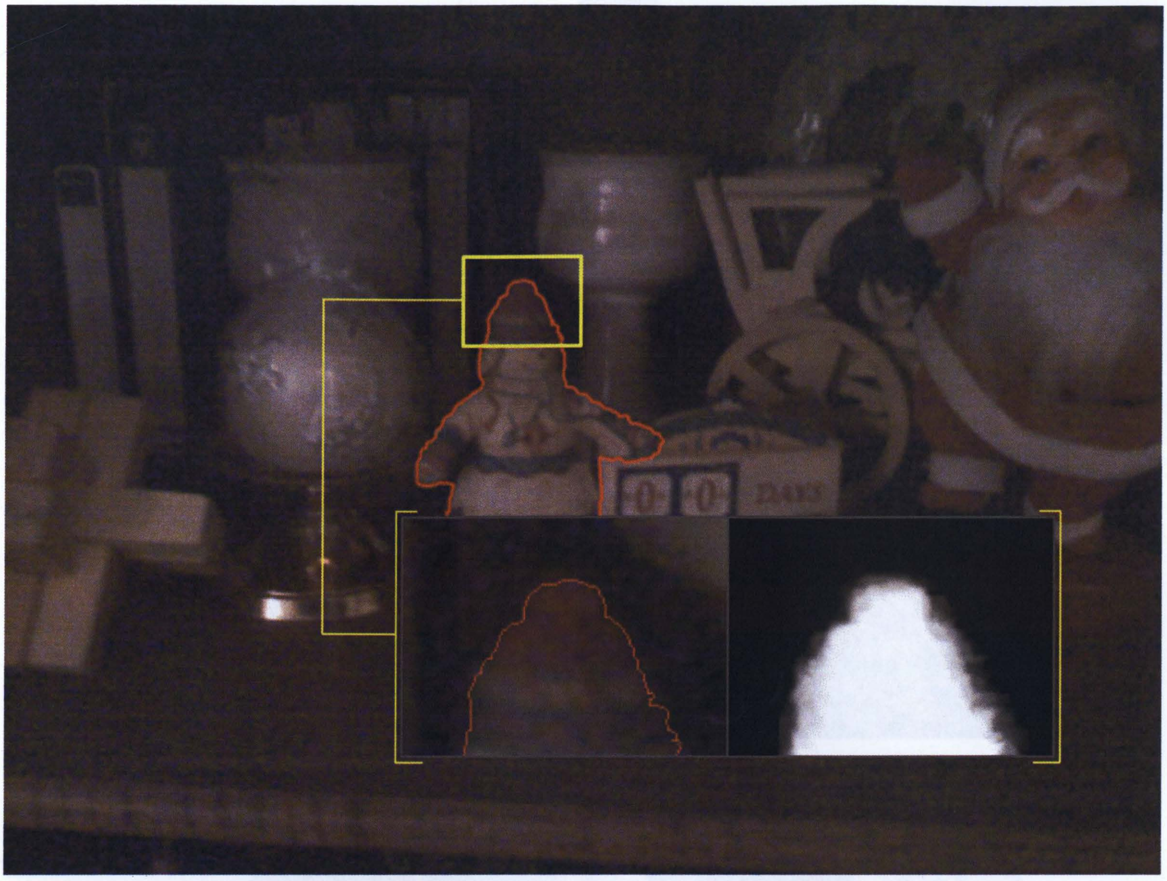


Figure 4.29: A close-up of one particularly affected region in Figure 4.28. The probabilities are much smoother and closer to the ideal.

4.6.3 Camouflaged Couch

This particular example is of a similar type to the "Basement Shelf" example. Here, an individual wearing Canadian Forces-issue CADPAT disruptive pattern camouflage is blending into a similarly patterned couch (Figure 4.30). Like the shelf image, the image was taken under indoor lighting conditions using a cellphone camera and, as a result, has a fair bit of noise and JPEG compression artifacts. Furthermore, because of the CADPAT, the boundary between the individual and the couch are quite indistinct in several places. In fact, the only visual cue to any sort of border are the shadows along the left side of the figure. This provides a unique challenge for a segmentation algorithm.

Figure 4.31 shows the labelling used and the results of the Random Walks and SSRW segmentation. In both cases, errors result from the weak boundaries but the SSRW is generally cleaner and closer to the perceived curve than Random Walks. As expected, the probabilities for the SSRW are cleaner and show more structural information than the Random Walks probabilities (Figure 4.32). Figure 4.33 shows a close-up of two regions in the image and their accompanying potentials, both SSRW and Random Walks.

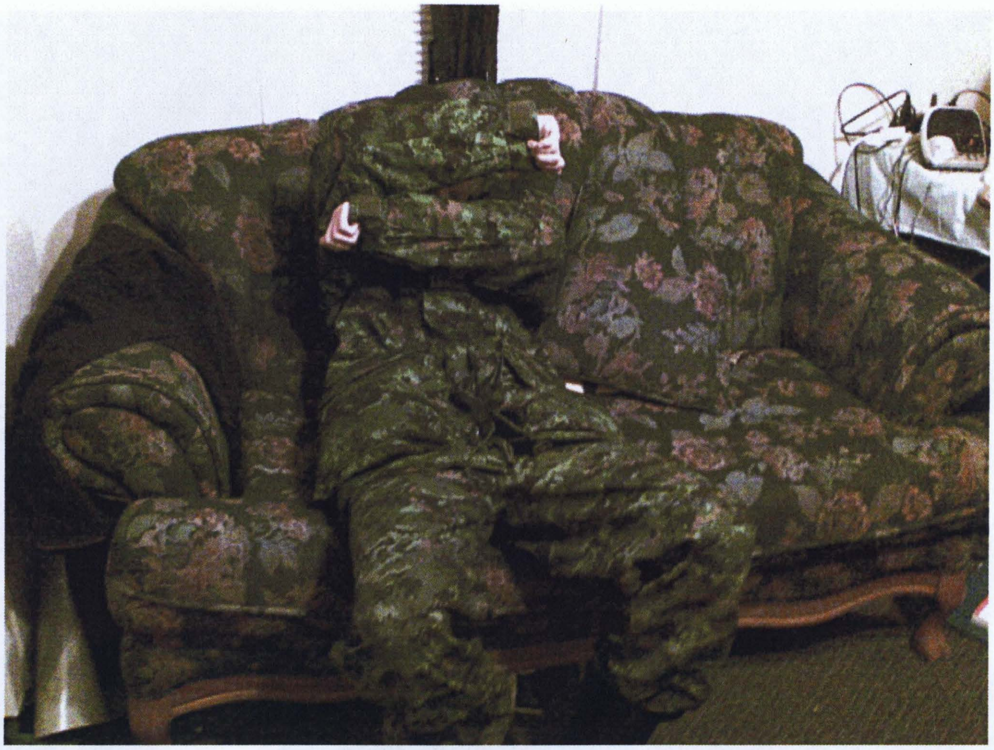
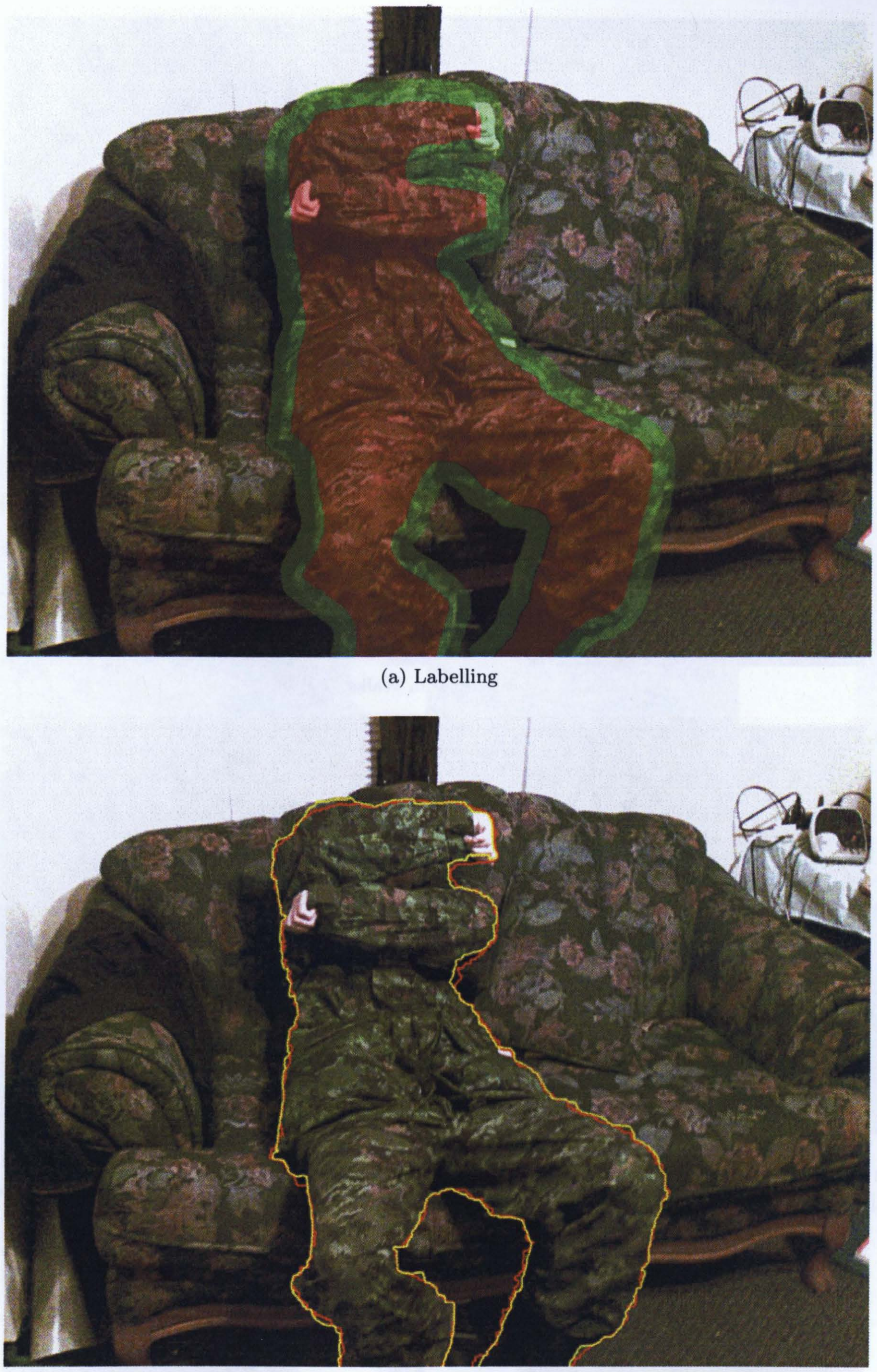


Figure 4.30: An individual wearing CADPAT blending into a similarly patterned couch.

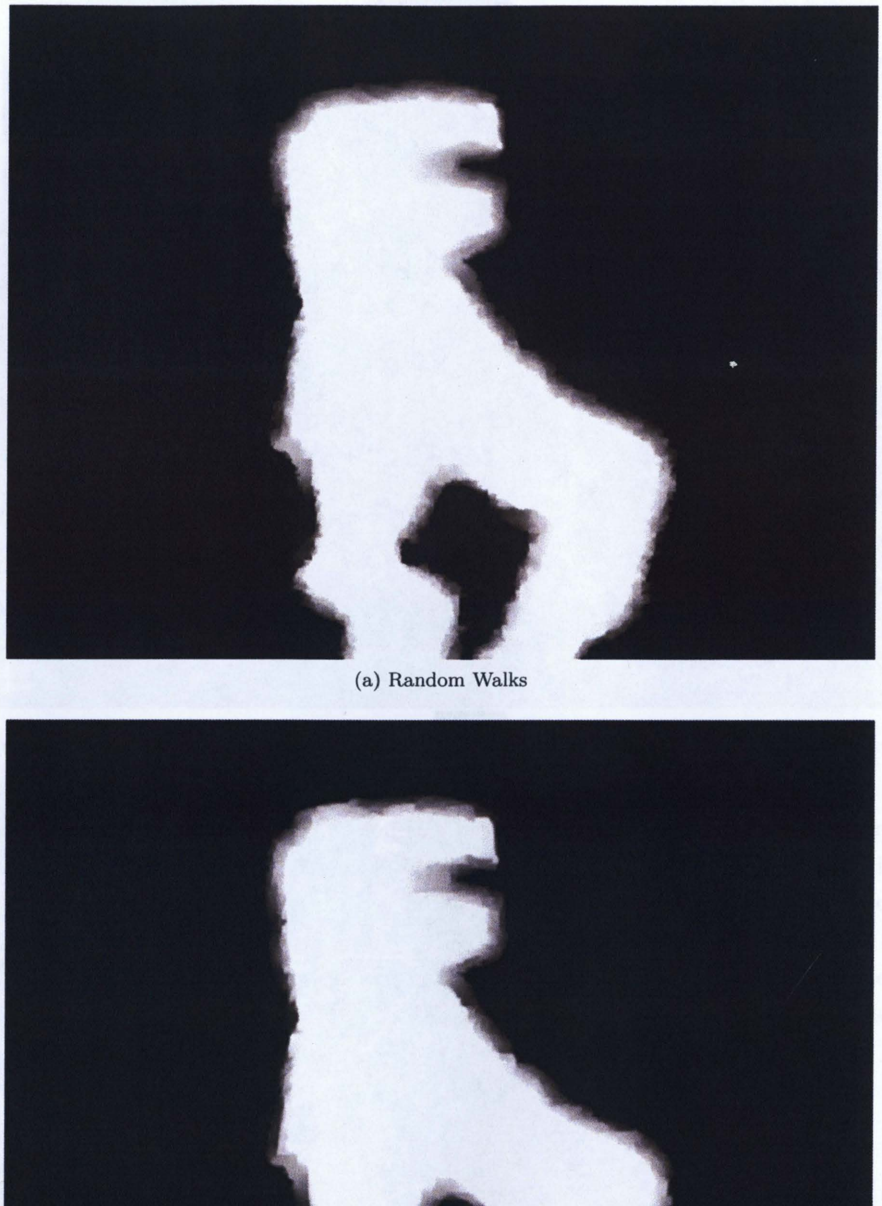
Overall, the SSRW does a pretty good job on this particular image. Because the algorithm is "locally-operating", it will pick the midpoint between the foreground and background as the segmentation boundary if the edges are weak. However, in some cases, Random Walks appears to follow the perceived object boundary *better* than the SSRW. This occurs because of the presence of a strong edge inside of the unknown region. Noise causes the random walker to ignore the edge but, when it is filtered through the scale-space, the edge becomes visible and it picks it. Generally, this can be fixed by simply adjusting the labelling in that particular region.



(a) Labelling

(b) Segmentation Boundary

Figure 4.31: The segmentation results for Figure 4.30. The Random Walks segmentation is shown as a red line while the SSRW segmentation is shown as a yellow line.



(a) Random Walks

(b) SSRW

Figure 4.32: The Random Walks and SSRW probabilities for the segmentation results shown in Figure 4.31.

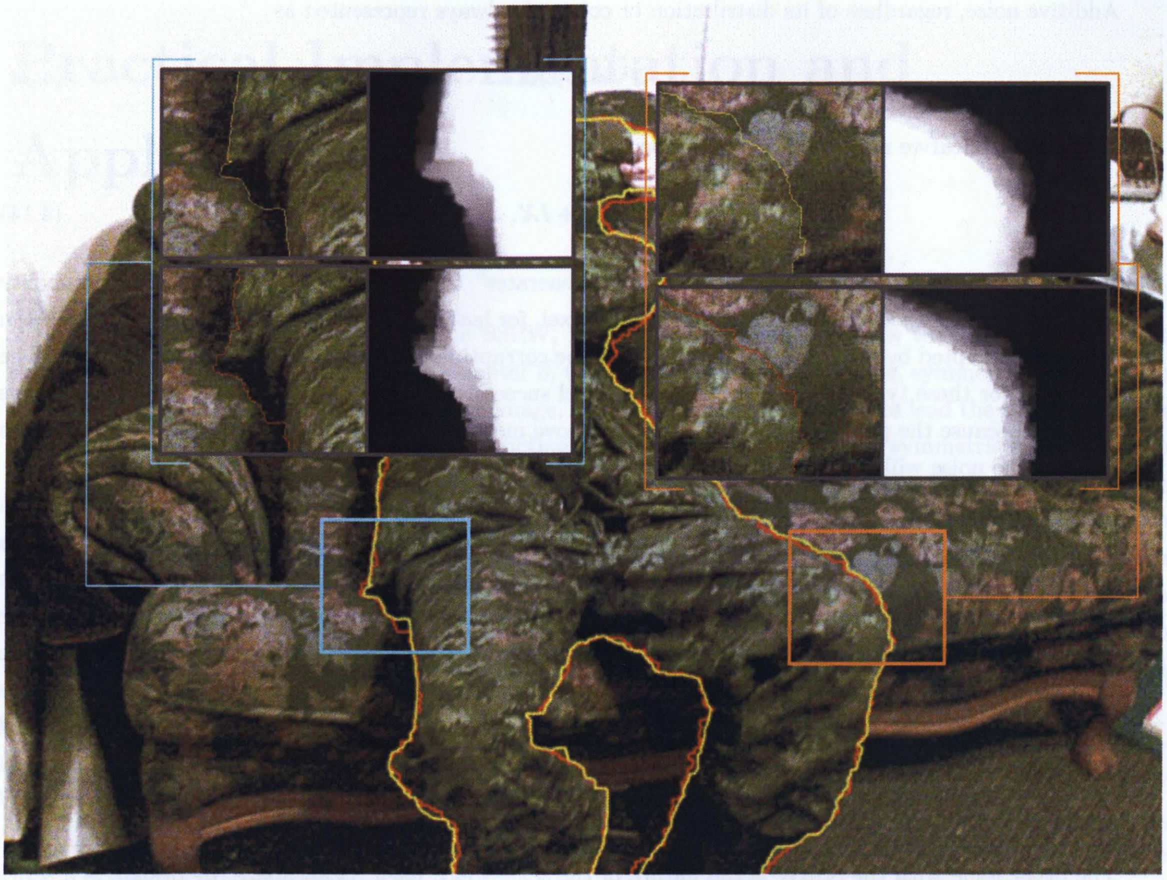


Figure 4.33: A close-up of two regions showing the results for Figure 4.31. Again, for the segmentation boundaries, Random Walks is represented by a red line and the SSRW is represented by a yellow line.

4.7 Issues with the SSRW

The SSRW provides an effective means of improving a segmentation in the presence of noise. However, how the SSRW accomplishes this leads to a couple of issues that can be observed in the examples presented. In the artificial images in 4.5.1 – 4.5.3, the SSRW clearly outperformed Random Walks, regardless of the type of noise. But, for the example in 4.5.4, the SSRW performs noticeably worse than Random Walks. This performance disparity can be understood by considering how impulsive noise is different from all of the other noise types.

Additive noise, regardless of its distribution or colour, is always represented as

$$I' = I + \mathcal{X}. \tag{4.13}$$

Similarly, multiplicative noise is always described as

$$I' = I + I\mathcal{X}. \tag{4.14}$$

As before, $\mathcal{X}$ represents the random variable that "generates" the noise in the image. The commonality between the noise types is that the noise affects *each* pixel, for lack of a better term, uniformly. Each pixel in the image is corrupted by the noise and the degree of the corruption is determined by noise distribution. The SSRW works for these types of noise because each level successively filters out the noise. The type of noise is irrelevant because the progressively larger filters remove more and more of the noise. At coarse scales, the majority of the noise will filtered out, leaving only the large-scale structures from the original image.

Impulsive, also known as "salt-and-pepper" noise is different. Impulsive noise randomly turns pixels on and off, giving the corrupted image the appearance of having been sprinkled with salt and pepper. Only a small percentage of the total pixels in the image are affected by the noise, the rest are still the same. The problem with simply blurring an image with impulsive noise is that the blurring spreads the effect of the impulse to the surrounding pixels. Considering the diffusion-like nature of scale-space, these impulses diffuse in the scale-space resulting in the noise having a more *pronounced* effect at higher scales. Random Walks works well under impulsive noise because it can, essentially, bypass the corrupted pixel. The corrupted pixel is so dissimilar to its neighbors that, for all intents and purposes, that pixel does not exist and there is a hole in the Random Walks lattice. However, because impulsive noise corrupts the scale-space, the SSRW does a poor job at dealing with it.

The other issue with the SSRW is the nature of the scale-space itself. Scale-space, by design, is a series of low-pass filtered signals. Theoretically, the scale-space should conserve all of the information from the original signal but, since the scale-space has been discretized, some information is lost. The end result is that, for a clean image, the SSRW will *never* be as accurate as Random Walks, as shown in Figure 4.11. Therefore, if the region being segmented has very detailed edges and the image is noise-free, it may not be the best choice to use the SSRW. Ultimately, the choice of whether or not to use the SSRW should be left to the user since they are the ones who can best determine where to apply it.

Chapter 5

Practical Implementation and Application

RANDOM Walks, and by extension, the SSRW, fall under a class of problems that have very efficient software implementations. As stated in Chapter 3, the system matrix, **L**, is sparse, symmetric and, because a grid (lattice) is being used to represent the image, banded. All of these properties lead the system to being efficiently stored and processed. A variety of methods exist for solving sparse and symmetric systems due to how often they occur in numerical analysis problems, specifically, the Finite Element Method (FEM). These methods are used to solve partial differential equations (PDE) on a lattice, normally for physical simulations such as the forces experienced by a bending beam and electromagnetic wave propagation. However, because Random Walks is essentially the solution to a PDE [21], these methods can be applied directly to find the image probabilities. Furthermore, because FEM is so commonplace, many of the methods are designed to run very quickly so that a simple simulation does not take days to complete. This makes it possible to run Random Walks, and the SSRW, at, or close to, realtime.

Two separate implementations of the SSRW were developed. The first is a basic MATLAB implementation used to develop and test out the SSRW concept. The benefit of MATLAB was that it allowed many of the more complicated mathematical operations, such as the linear equation solver and matrix operations, to be handled by MATLAB. This was at the expense of having control over how computing resources were allocated. Due to how MATLAB is designed, memory handling became an issue as certain operations had to be "vectorized", or expressed as matrix operations, rather than more memory efficient loops.

The second implementation was as a plugin to the Adobe After Effects [38] compositing application. The plugin utilized Nvidia Corporations CUDA platform [39] to leverage the power of an onboard graphics card, also referred to as a "Graphics Processing Unit" (GPU). The primary benefit to using CUDA is that a graphics card is designed to process many pieces of information in parallel. The vast majority of operations to produce a simulated 3D image are matrix operations on a very large number of vertices. These vertices can be treated independently, therefore making it possible to perform many matrix operations at once. As a result of this design choice, graphics cards are also well suited to problems such as image processing and solving large systems of linear equations. Unfortunately, and unlike MATLAB, this meant that all of the mathematical operations would have to be implemented from scratch but in exchange for more control over the available computing resources. This allowed the SSRW to run, in many case, at realtime speeds.

5.1 Scale-Space Generation

The SSRW poses a number of problems with respect to its implementation. Namely, the generation of the scale-space is the largest computational addition to the Random Walks algorithm. If done improperly, generating the scale-space will become quite time-consuming and it will defeat any advantages conferred by an efficient linear equation solver.

The Gaussian kernel has a number of features, from a signal processing perspective, that make it very simple to implement a fast blur filter. The first property is the separability of the Gaussian kernel. Separability refers to the property of a kernel such that

$$h[x,y] = h_x[x] * h_y[y], \tag{5.1}$$

where $h_x[x]$ and $h_y[y]$ are the x and y components of $h[x,y]$. Because the Gaussian kernel,

$$h[x,y|\sigma] = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right), \tag{5.2}$$

is separable, a filtered image, I_f, can be produced with the operation

$$I_f[x,y] = h_y[y|\sigma] * (h_x[x|\sigma] * I[x,y]), \tag{5.3}$$

where

$$h_x[x|\sigma] = \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{x^2}{2\sigma^2}\right) \text{ and } h_y[y|\sigma] = \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{y^2}{2\sigma^2}\right). \tag{5.4}$$

Using a separable filter is more desirable than a non-separable filter because the number of operations required to filter the image is less for a separable filter. The reason is due to the basic filtering operation being a convolution. Discrete, two-dimensional convolution is defined as the sum

$$f[x,y] * h[x,y] = \sum_{k=-\infty}^{\infty}\sum_{j=-\infty}^{\infty} f[x-j, y-k]h[j,k]. \tag{5.5}$$

For a NxN-pixel kernel, there are a total of N^2 floating-point multiplications *per pixel* (float-point multiplications are considered to be "slow" operations). However, for a separable filter, there are only a total of $2N$ multiplications per pixel, a considerable reduction in execution time when the kernel and/or image are large.

The other desirable property of the Gaussian kernel is that, as a Finite Impulse Response (FIR) filter, it has "zero phase". That is, given the kernel's discrete Fourier transform, $H(e^{j\omega})$, its phase response is zero, or $\angle H(e^{j\omega}) = 0$. An FIR filter is considered to be "zero phase" if its coefficients are symmetric about $k = 0$ so that the relation

$$h[k] = h[-k] \tag{5.6}$$

is true [40]. For a Gaussian, this relation holds because the Gaussian function is symmetric about its mean. The fact that the filter is zero-phase means that the number of multiplications *per sample* can be halved.

Consider the standard form of a N-element FIR filter:

$$g[x] = \sum_{i=-N/2}^{N/2} h[i + N/2]f[x - i]. \tag{5.7}$$

Assume that the number of elements in the filter (i.e. the value of N) is odd. If $h[x]$ is a zero-phase FIR filter then the filter coefficients are symmetric. Therefore, the following rearrangement can be made:

$$g[x] = h[0]f[x] + \sum_{i=1}^{N/2} h[i](f[x - i] + f[x + i]). \tag{5.8}$$

This filter form is beneficial for performance reasons because it requires half the number of multiplications and requires less filter coefficients to be computed.

Both the MATLAB and CUDA implementations rely on the filter being separable to speed up the processing. MATLAB's built-in image processing functionality was used to perform the actual blurring and scale-generation. The filtering functions provided by MATLAB are highly optimized and run very efficiently. For most images, the scale-space generation ran very quickly and was a perceptibly negligible portion of the processing time.

The CUDA implementation performed similarly to MATLAB, with the exception of running faster. Because this filter was running on a so-called "massively-parallel processor", each row and column could be processed, effectively, at the same time. The result is a noticeable speedup with respect to the MATLAB generation code. Table 5.1 lists the results of the scale-space being generated on a number of images. As before, a four-level scale-space of $\sigma = \{0, 1, 2, 4\}$ was used for the test.

Size	Matlab	CUDA	Speedup
425x571	0.329s	0.267s	1.23x
2048x872	2.247s	0.811s	2.77x
1920x1080	2.613s	0.948s	2.76x
2048x1503	3.868s	1.279s	3.02x

Table 5.1: Comparison of MATLAB and CUDA scale-space generation times for a number of various sample images. The figures are ordered by ascending pixel counts.

All of the tests were performed on the same computer and the actual execution times are unimportant since they merely reflect the nature of the hardware that the test was performed on. The "Speedup" column contains the most important information. It indicates how many times faster the CUDA implementation was than the MATLAB implementation.

5.2 System Storage and Generation

The generation of the system matrix, **L**, is the most difficult part of the Random Walks and SSRW algorithm. The difficultly stems from the size of the system that needs to be solved. For any image with a height of

H and a width of W, the resulting matrix will be HWxHW. For a scale-space with D levels, the size of the matrix balloons to HWDxHWD. Fortunately, as mentioned in Chapter 3, $\mathbf{L}$ is sparse, banded and symmetric, qualities that can be exploited to improve memory storage. This is particularly important for a CUDA implementation since, unlike a CPU that can swap data into and out of virtual memory, the GPU has a limited pool of available memory (video RAM). This pool is shared with the operating system so the amount available is typically less than the total amount.

The following discussion relates primarily to the CUDA implementation as MATLAB utilizes its own internal sparse matrix storage scheme. Furthermore, to be able to run the algorithm in any sort of reasonable length of time, for instance, less than ten minutes, the matrix indices have to be generated before the system can be constructed. The CUDA implementation demonstrates how the algorithm *should* be implemented (i.e. not requiring the generation of indices beforehand). With CUDA, all of the operations occur in parallel so every element in a vector or matrix is processed simultaneously.

The system matrix is not constructed directly. Rather, the adjacency matrix, $\mathbf{A}$, is constructed first and $\mathbf{L}$ is derived from that. The elements in the adjacency matrix are defined as

$$A_{i,j} = \begin{cases} G_{i,j} & i \text{ is adjacent to } j \\ 0 & \text{otherwise} \end{cases}. \tag{5.9}$$

This definition holds regardless of whether Random Walks or the SSRW is used. Because $\mathbf{A}$ is banded and symmetric, the matrix can be stored in a compressed representation, $\bar{\mathbf{A}}$. This is a dense, rectangular matrix composed of all of the bands in $\mathbf{A}$ such that

$$\bar{\mathbf{A}} = \left[\vec{A_0} \quad \vec{A_1} \quad \vec{A_2} \quad \dots \quad \vec{A}_{N-1}\right], \tag{5.10}$$

where $\vec{A_i}$ is each matrix band and N is the number of bands. This is typically known as "band storage". For a 4-connected (Random Walks) graph, there are two bands while for the 6-connected SSRW graph, there are three. Using a zero-based numbering scheme, the index of each vector, i, corresponds to the pixel offset; i.e. for pixel i, its horizontal neighbor is at $i+1$, its vertical neighbor is at $i+W$ and its scale neighbor is at $i+HW$. Therefore, the complete compressed SSRW system can be stored as

$$\bar{\mathbf{A}} = \left[\vec{0} \quad \vec{A}_1 \quad \vec{A}_W \quad \vec{A}_{HW}\right] \tag{5.11}$$

This storage is favorable to storing the uncompressed matrix because the amount of elements being stored will be $3HW$ and $4HWD$ for Random Walks and the SSRW, respectively.

The system matrix, $\bar{\mathbf{L}}$, is constructed *in-place* from $\bar{\mathbf{A}}$ in a two-step process. The basic definition of $\mathbf{L}$ is

$$\mathbf{L} = \deg(\mathbf{A}) - \mathbf{A}, \tag{5.12}$$

where $\deg(\mathbf{A})$ is the degree matrix of $\mathbf{A}$. The degree matrix is a matrix whose main diagonal contains the degree (sum of edge weights) of each node. All of the other elements in the matrix are zero.

The first step is to calculate $\deg(\mathbf{A})$. This value will be stored in the main diagonal of $\bar{\mathbf{L}}$ so that

$\vec{L}_0 = \deg(\mathbf{A})$. The i-th element of the vector, $L_0^{(i)}$, is defined as

$$L_0^{(i)} = \left(A_1^{(i)} + A_W^{(i)} + A_{HW}^{(i)}\right) + \left(A_1^{(i-1)} + A_W^{(i-W)} + A_{HW}^{(i-HW)}\right). \tag{5.13}$$

The next step is to solve $\deg(\mathbf{A}) - \mathbf{A}$, which is trivial. This simply requires negating each of the off-diagonal bands. The final result is that

$$\bar{\mathbf{L}} = \left[\vec{L}_0 \quad -\vec{A}_1 \quad -\vec{A}_W \quad -\vec{A}_{HW}\right]. \tag{5.14}$$

Up until this point, the foreground and background labelling has not been considered. Because these nodes have a known value, each element of $\vec{L}_0$ that is in the set of seeds, $\mathbb{S}$, is set to 1. The corresponding "rows" and "columns" in $\bar{\mathbf{L}}$ are set to 0.

The boundary vector, $\vec{b}$, is generated by the operation

$$\vec{b} = \mathbf{L}\vec{s}, \tag{5.15}$$

where $\vec{s}$ is the seed vector. All of the elements in $\vec{b}$ that are in $\mathbb{S}$ are set to the appropriate value (x_{BG} or x_{FG}) since they are already known.

5.3 System Solver

The method of solving the system $\mathbf{L}\vec{v} = \vec{b}$ depends mainly on the computational environment. Generally, system solvers that solve the system directly, such as LU Decomposition followed by Gaussian Elimination, are computationally quick because the system is solved immediately. The number of operations required to solve the system are directly related to the size of the matrix. However, these methods require a fair amount of memory because several matrices are produced by the decomposition process.

The other option is to use a slower iterative system solver. An iterative solver is really an optimization algorithm that tries to minimize an error term that is derived from the current "solution" and the actual solution. By iteratively minimizing the error, the solution to the system can be found. But, since this method relies on an error term, the solution is not "exact" and there is no way to know how long the algorithm will take to run. The benefit is that the method is more conservative in terms of memory consumption and requires much less than a direct solver.

A good choice for an equation solver is the Conjugate Gradient Method [41]. It is designed to work on sparse systems that are symmetric and positive-definite, both attributes that $\mathbf{L}$ has [21]. The algorithm also has very modest memory requirements and is guaranteed to converge within N iterations, where N is the number of equations in the system. A preconditioner, $\mathbf{P}$, was used with the CGM because it is advisable to use a preconditioner for large systems. It is shown in [42] that $\mathbf{P}^{-1}\mathbf{L}\vec{v} = \mathbf{P}^{-1}\vec{b}$ will be solved faster than $\mathbf{L}\vec{v} = \vec{b}$.

$\mathbf{P}$ is not applied to the system itself. Rather, it is applied to the residual vector, $\vec{r}$, a measure of how close the algorithm is to the solution. The Jacobi preconditioner was chosen since it was the simplest preconditioner and the most computationally efficient. The Jacobi preconditioner is defined as the diagonal matrix containing all of the diagonal entries in the system matrix. Therefore, $\mathbf{P}$ has a trivial inverse which is simply the inverse of every element on the diagonal. This allows the application of the preconditioner to a vector to be

$$\vec{r}_P = \mathbf{P}^{-1}\vec{r} \equiv r_P^{(i)} = \frac{r_i}{L_0^{(i)}}. \tag{5.16}$$

The implementation itself did not reside entirely on the GPU. Vector-vector operations (such as the dot product) and matrix-vector multiplication were implemented as CUDA functions but the algorithm itself was implemented as conventional CPU program. A GPU is very good at performing very simple operations (addition, multiplication, subtraction) very quickly and in parallel. An operations such as a vector dot product, which it composed of many additions and multiplications, is said to have very high *arithmetic intensity*. However, an operation that requires branching or iterating is said to have low arithmetic intensity because not many calculations are occurring at any given time. Therefore, it is natural to implement all of the basic mathematical operations on the GPU while allow the CPU to control the overall algorithm logic.

It should be noted that Grady has developed a method for speeding up the solution to Random Walks by pre-computing the eigenvectors of $\mathbf{L}$ [43]. This method essentially modifies the Normalized Cuts [7] algorithm that was mentioned in Chapter 2 to accept seeds at some later time. The algorithm uses the first K eigenvectors to approximate what the Random Walks solution *should* be. This method is much faster than doing a full solution to Random Walks but is less accurate. It can easily be utilized with the SSRW due to the nature of the SSRW. However, because the eigenvectors have to be pre-computed, this method is not particularly useful for an application such as plugin where pre-processing the image may not be an option. Therefore, applying this particular method to speeding up the SSRW has not yet been investigated, but may be at some future date where the application requires offline processing.

5.4 Applications

The nature of Random Walks, and by extension the SSRW, makes it very desirable for certain applications. For any application where the labelling between the foreground and background are well-defined *and* the regions must be connected and simple (i.e. there are no "holes"), then these two algorithms are natural candidates. The localized nature of the linear system ensures that the region won't have small gaps since the probability has to decrease monotonically along the shortest path from a source to a sink. One particular application that is well-suited to the SSRW is rotoscoping.

5.4.1 Rotoscoping

Rotoscoping is the act of cutting out objects from video sequences. Conceptually, this is the same as video segmentation but the difference is that video segmentation is, for the most part, an automated procedure while rotoscoping is completely supervised. Most rotoscoping artists use a variety of techniques to rotoscope a scene, ranging from using parametric curves to chromakey (aka "green-screen") matting.

While the SSRW cannot be used in cases such as chromakey matting, though segmentation techniques such as Bayesian matting can, the SSRW can be used as a form of online curve adjustment. Generally, parametric curves, such as Bézier curves, are used to describe object edges. These curves are described by their derivative and therefore will do a very good of following an edge if the edge is smooth. However, if the edge has many corners and is very rough then a parametric curve will have a difficult time describing the edge.

Generally speaking, the individual *control points* that the user places to control the curve topology are quite accurate. The user knows exactly where the points are supposed to lie, even over multiple frames. What is not known exactly is the curve between the points. Because it can be reasonably assumed that the points lie on the rotoscoped object's boundary, this problem can be viewed a segmentation problem where the curve is the segmentation boundary. Therefore, the problem is now reduced to determining the appropriate algorithm and labelling to use.

5.4.2 Interactive Rotoscoping

As mentioned, Random Walks and the SSRW are good algorithms for this type of task since they are locally operating and using the appropriate labelling will give very good results. The algorithm "operates locally" because the position of the seeds will affect the probabilities and, in turn, affect the segmentation. Therefore, it's possible to create a labelling that will respect the locations of the control points (Figure 5.1). To adjust the entire curve, the segments can be chained together as shown in Figure 5.2.

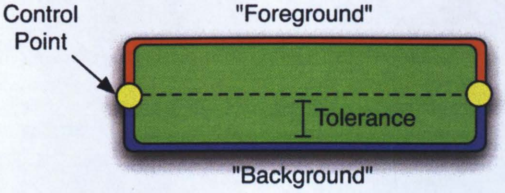


Figure 5.1: Labelling used for a curve segment in the rotoscoping process.

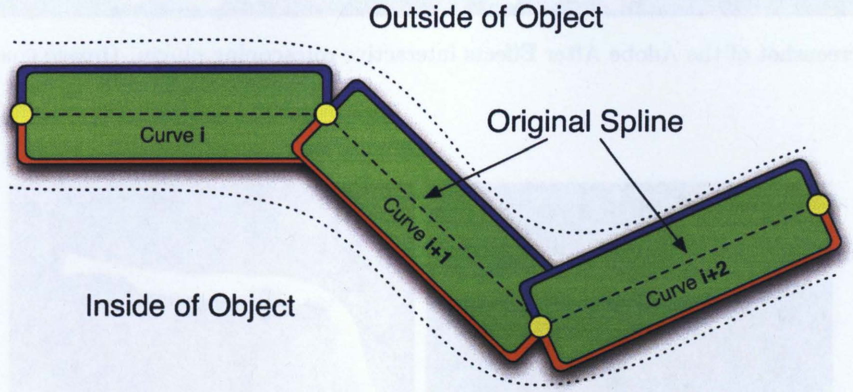


Figure 5.2: The labelling as shown for multiple curve segments.

The user interface for the rotoscoping implementation is shown in Figure 5.3. The plugin is an effect that operates on pre-existing Bézier curves that a user has already placed. The plugin iterates through each curve segment to produce the labelling for each segment. The segmentation algorithm runs on each segment to produce a segmentation boundary, shown as a white line. Depending on the situation, the user can choose to use either Random Walks or the SSRW. Again, this depends on the situation and is up to the user to decide if they need to use it. Notice that while the interpolated curve only loosely follows the object boundary, the segmentation boundary follows the object boundary very precisely. Figure 5.4 shows the generated labelling and probabilities. These allow the user to observe what the algorithm is doing and provide an idea of why the results are being returned the way they are.

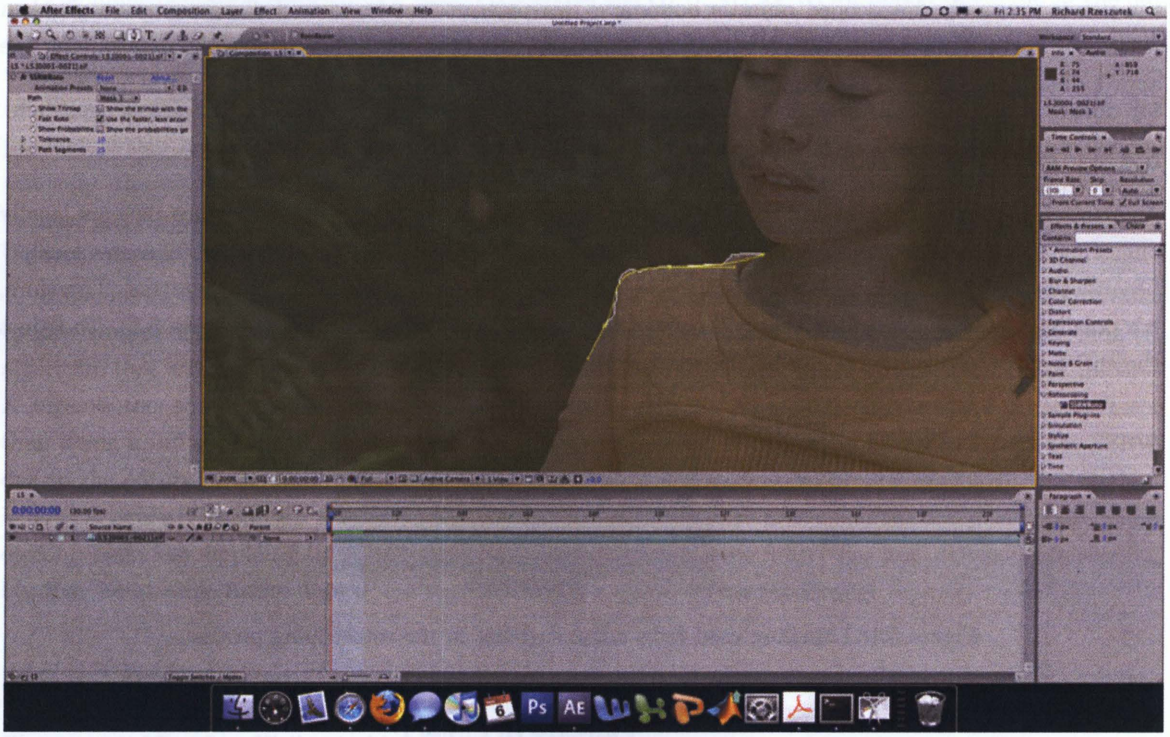
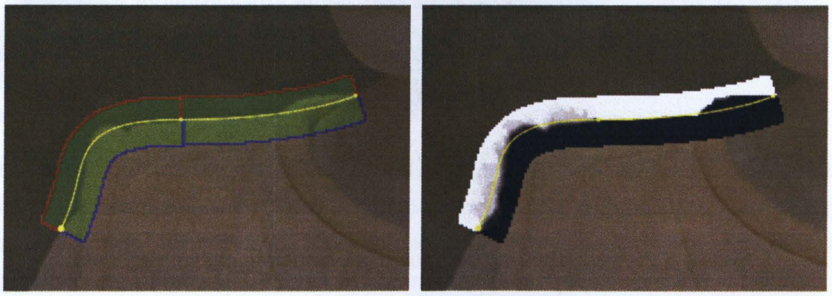

After Effects File Edit Composition Layer Effect Animation View Window Help

Figure 5.3: Screenshot of the Adobe After Effects interactive rotoscoping plugin. (Image courtesy of IMAX Corp.)



(a) Labelling

(b) Probabilities

Figure 5.4: Screenshots showing the plugin displaying the labelling used and the returned probabilities.

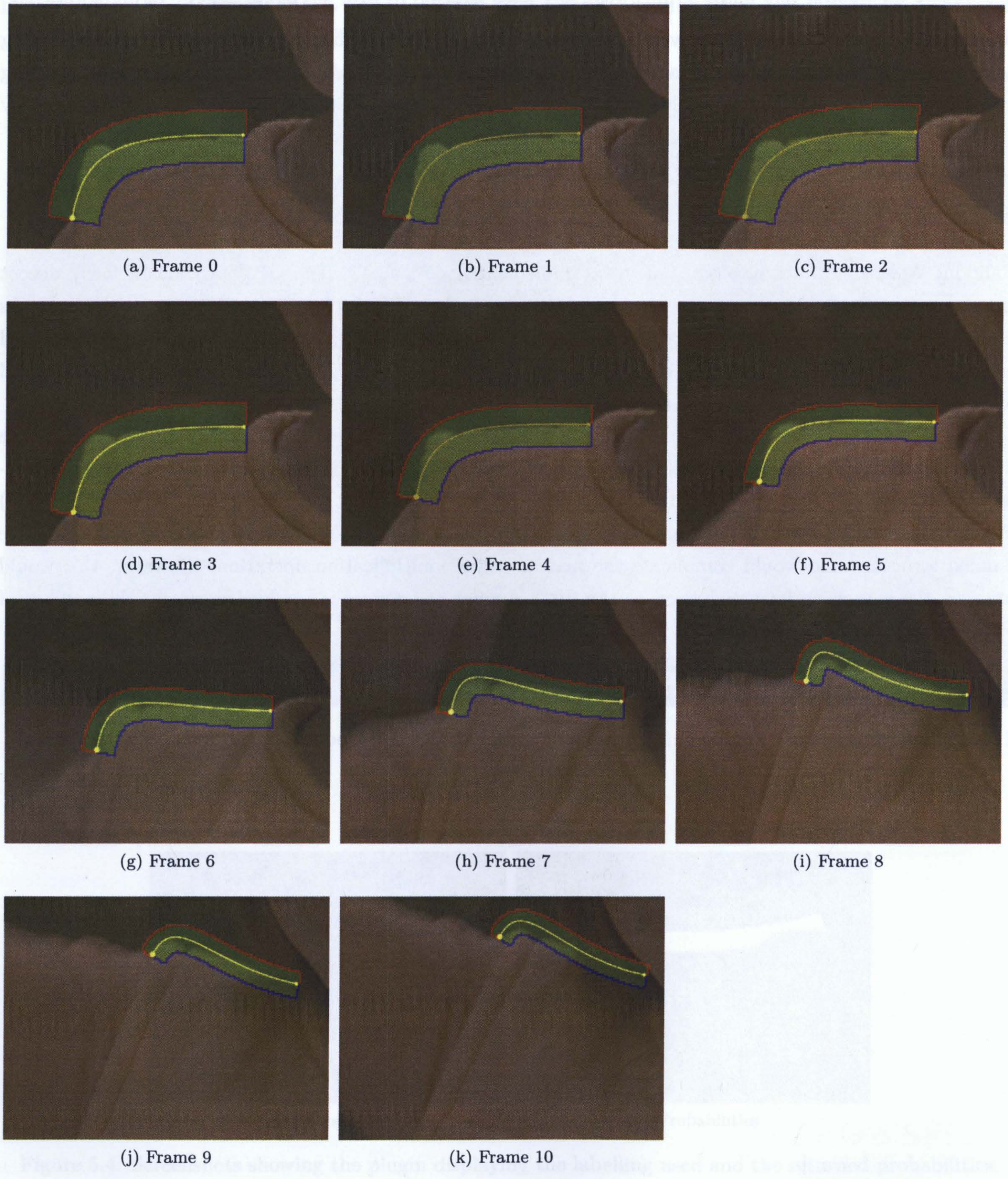
The algorithm was implemented in CUDA as described in the preceding sections. It was difficult to measure the exact execution times because they were heavily dependent on the number of nodes in the graph. Quite simply, the larger the curve segment, the longer it would take for the algorithm to execute.

However, on average, the algorithm typically found the segmentation boundary in 100 milliseconds. The times tended to vary between 10 milliseconds and 1 second, depending on the size of the image and size of the curve. In general, these times were fast enough that the user did not experience an unpleasantly long delay while waiting for the algorithm to finish processing. Figures 5.5, 5.6, 5.7 show the algorithm operating over several frames. For this particular frame sequence, After Effects reported a frame rate of approximately 10 frames per second while the calculation was occurring.

5.5 Discussion

Random Walks is a natural algorithm to be implemented on a GPU. The GPU provides a fairly decent speedup with respect to, for instance, MATLAB-based code. The rotoscoping method from the previous section was originally implemented as a stand-alone MATLAB GUI application. However, the code would execute between two to three times slower than on the GPU. This is comparing the CGM solver implemented on the GPU to the linear system solver built into MATLAB.

At the moment, the code is not completely optimized and could run even faster. $\bar{\mathbf{L}}$ currently contains the information for *all* of the nodes, rather then just the nodes in the unknown region, $\mathbb{U}$. The current implementation does not store the element indices and relies on the fact that, as a complete (i.e. uncompressed) matrix, $\mathbf{L}$ is banded. Removing the rows and columns of $\mathbf{L}$ that correspond to $L_{i,i} = 1$ would destroy the banded structure and would complicate the matrix-vector multiplication operation. However, this would improve the overall performance because the total number of operations would decrease as the number of elements in $\mathbb{U}$ is *always* less than $\mathbb{S}$.


(a) Frame 0

(b) Frame 1

(c) Frame 2

(d) Frame 3

(e) Frame 4

(f) Frame 5

(g) Frame 6

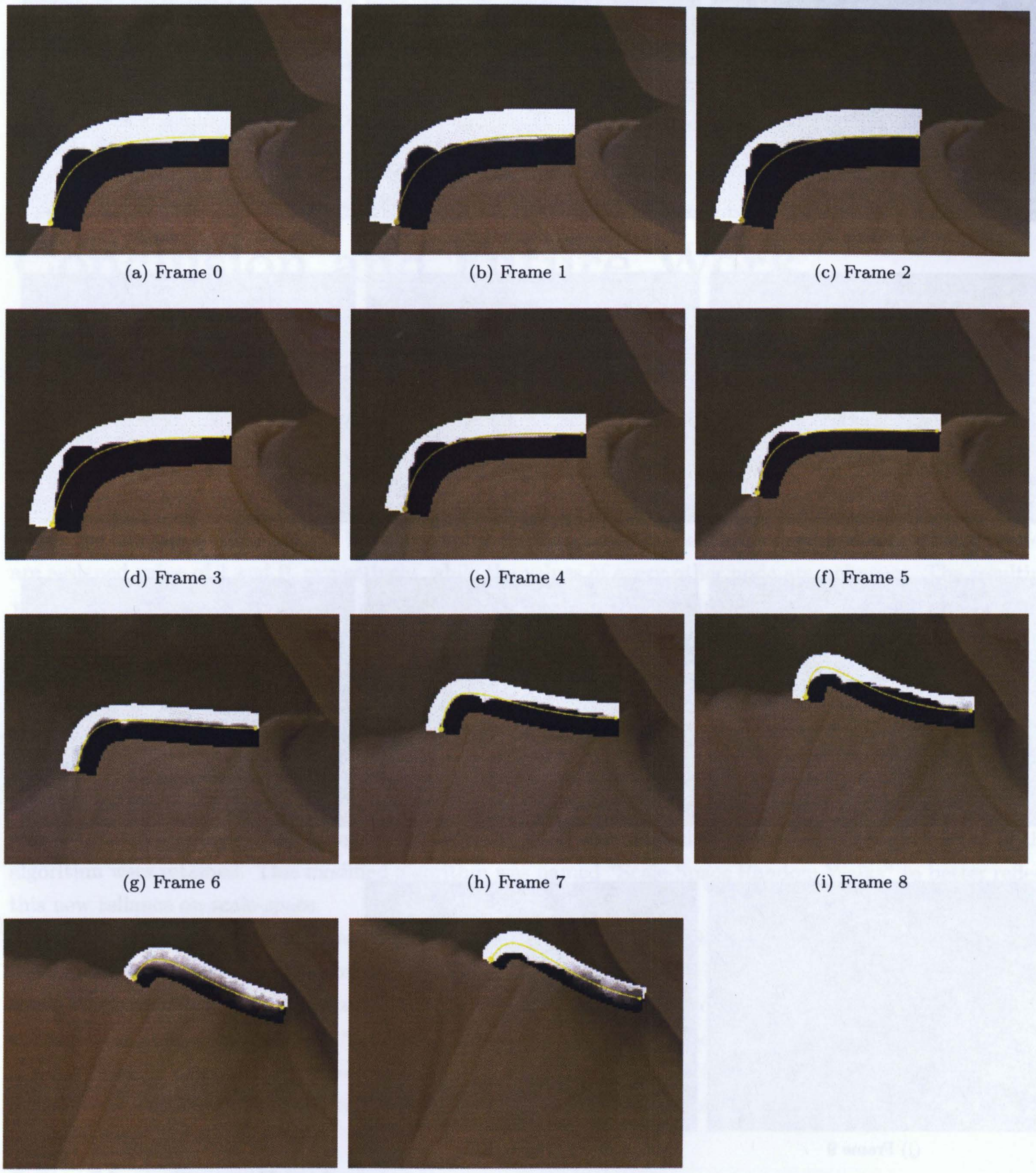
(h) Frame 7

(i) Frame 8

(j) Frame 9

(k) Frame 10

Figure 5.5: Frame sequence showing the evolution of the trimap over the course of the sequence. As the spline becomes closer to the actual boundary, the tolerance is reduced by the user to ensure that only the edge of interest is in the unknown region.



(a) Frame 0

(b) Frame 1

(c) Frame 2

(d) Frame 3

(e) Frame 4

(f) Frame 5

(g) Frame 6

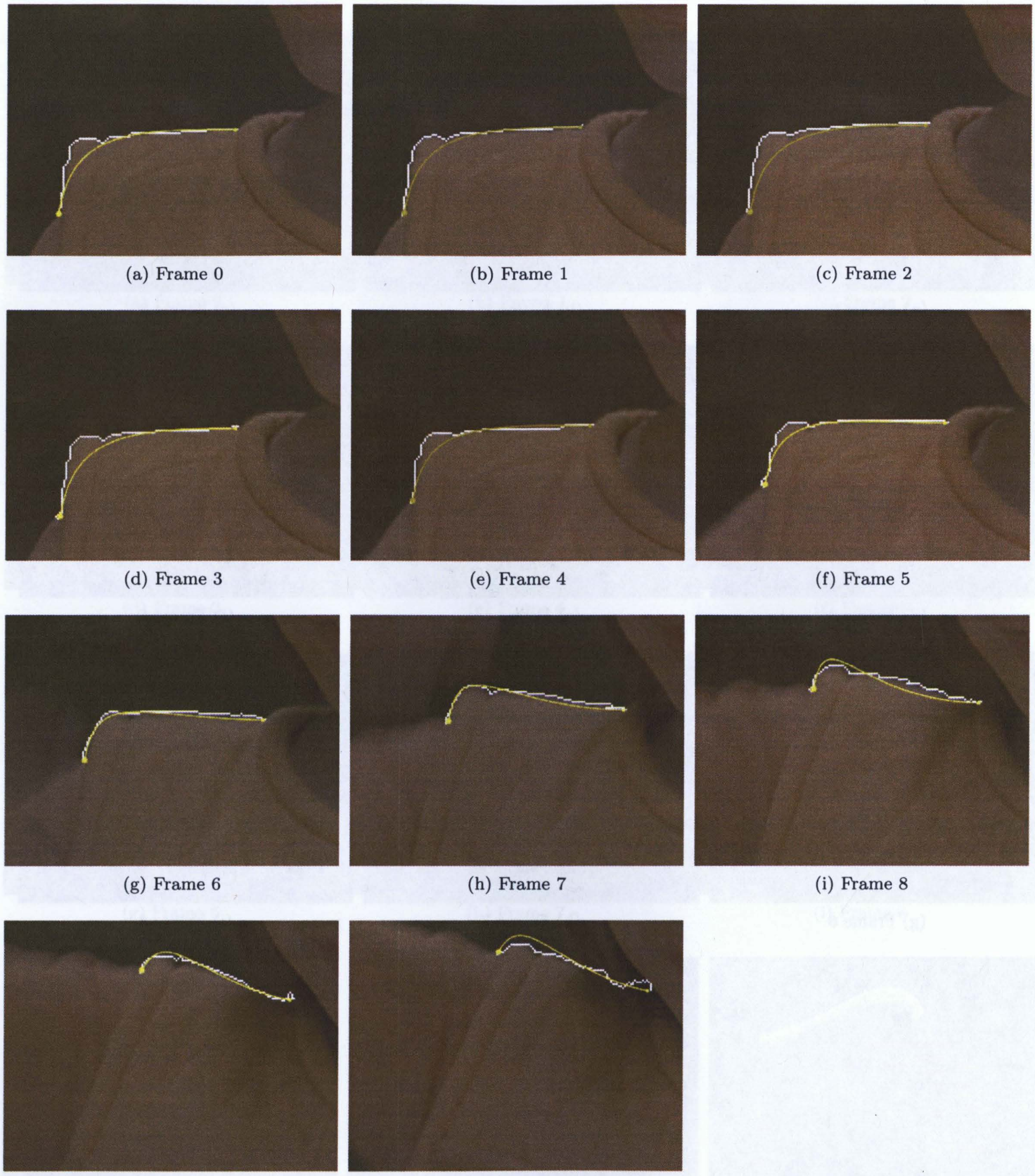
(h) Frame 7

(i) Frame 8

(j) Frame 9

(k) Frame 10

Figure 5.6: This frame sequence displays the probability map for each of the frames. Each frame is processed independently and the result in one frame has no affect in any other frame. The foreground/background labels are assigned arbitrarily so, in this instance the "background" is assigned a potential of 1.



(a) Frame 0

(b) Frame 1

(c) Frame 2

(d) Frame 3

(e) Frame 4

(f) Frame 5

(g) Frame 6

(h) Frame 7

(i) Frame 8

(j) Frame 9

(k) Frame 10

Figure 5.7: This frame sequence shows shows the evolution of the segmentation boundary over the course of the sequence. Note that the boundary accurately locks onto the edge regardless of the original curves accuracy. In this sequence, the edge is well-defined but, towards the last couple of frames, the right-most side of the curve is longer on any edge. The nature of the Random Walks algorithm ensures that when there is no strong edge, the segmentation will always be roughly in the centre of the unknown region. As such, the boundary defaults to the approximate position of the original curve (it is not *exactly* on the curve due to the fact the area is not completely uniform).

Chapter 6

Conclusion and Future Work

THIS thesis has presented a novel segmentation method based on the Random Walks frameworks. Random Walks is the solution to the linear system

$$\mathbf{L}\vec{v} = \vec{b}, \tag{6.1}$$

where the image is treated as large lattice (graph). Nodes (pixels) marked as "foreground" and "background" are assigned value of 1 and 0, respectively, while the values of every other node are unknown. The resulting vector, $\vec{v}$, is a probability map that assigns likelihoods to each pixel being visited by a random walker. Any pixel with a probability greater than 0.5 (50%) was considered to be part of the foreground.

A problem identified with Random Walks was its inability to handle noise well. Often, noise that still left many of the image features intact would result in a "garbage" segmentation when using Random Walks. However, by augmenting Random Walks so that it operated on a scale-space it was possible to make Random Walks return good segmentations in situations were it might not have otherwise. This modification to Random Walks was made by adjusting the graph structure so that it was a three dimensional scale-space, rather than a two dimensional image. The algorithm itself was untouched so that all of the properties of the algorithm were retained. This modified algorithm was named "Scale-Space Random Walks" to better reflect this new reliance on scale-space.

Finally, an example implementation and application of Random Walks and the SSRW were presented. An efficient implementation of the algorithm was presented on modern GPU hardware to show how the algorithm could be constructed to best utilize the power of a computer's onboard GPU. The original formulation of the algorithm was presented in a "parallel-friendly" manner to show how the operations used to build the system could be easily implemented on a GPU. An application of the SSRW inside of a novel rotoscoping method was also shown as a way to present the SSRW as a practical solution to an existing problem.

6.1 Key Contributions

A number of contributions have been made in this thesis. The main contributions are:

- **A Sigmoidal Weighting Technique:** The weighting calculation utilized a sigmoidal, rather than exponential function. This weighting scheme provides a degree of greater degree of control for the weighting than the exponential can provide on its own.

- **Use of a Scale-Space:** The primary contribution of the thesis. The use of the scale-space allows the segmentation algorithm to ignore a good percentage of the noise present in a corrupted image. While the scale-space did not work in the presence of impulsive noise, this is not unexpected as impulsive noise cannot be removed with a linear filter.
- **Implementation in CUDA:** The algorithm was successfully implemented using the CUDA parallel processing platform. This allowed the algorithm to directly access the computational resources of the onboard graphics card.
- **Development of a Rotoscoping Plugin:** The plugin demonstrated how the SSRW could be used to solve a practical problem such as improving the quality of a rotoscoper's rotoscope.

The examples presented can conclusively show that the SSRW improves the segmentation quality in the presence of noise. It is important to note, however, that the use of the scale-space was only made possible by the nature of the Random Walks algorithm. The would not have been possible with other algorithms, such as Graph Cuts, because it would be much more difficult to incorporate the information provided by the scale-space. For instance, Graph Cuts would have performed a volumetric binary segmentation through the scale-space and it would have been difficult to try to recombine the segmentations at each scale into one final segmentation, as was done with the SSRW.

As mentioned in the introduction, this thesis did not set out to solve the problem of image segmentation. In fact, this problem is so vague that, short of the development of strong artificial intelligence, there is unlikely to be any solution. However, new segmentation methods will always be developed to try and solve some specific segmentation problem. SSRW improves on an already well-performing algorithm, Random Walks, and improves its performance in the presence of noise. This makes it much more robust and more useful in situations where it may not have worked as well.

6.2 Future Work

Work on the SSRW algorithm itself is complete. The algorithm has met all of its development goals, mainly to improve the performance of Random Walks in the presence of noise. Where research can be directed is into *applications* of the SSRW. Currently, the SSRW is mainly used for segmentation problems but future work may include investigating how it may be extended into application such as stereo-matching. Already, several stereo-matching algorithms exist based on Graph Cuts and, because of some of the similarities that the algorithm shares with Graph Cuts, it may be possible to use the SSRW in a similar fashion. There are other, segmentation-type applications that can also be explored such as automated detail mask improvement. Detail masks are the result of rotoscoping and often there are small, but noticeable mistakes in the masks. It is desirable to develop an automated, offline improvement method to fix these small errors.

Other future projects includes improving the current CUDA implementation of the SSRW. It was stated in Chapter 5 that further performance gains could be obtained by optimizing the algorithm. The CUDA implementation is still a "work-in-progress" and there are a number of things that could be done to speed up the execution. These range from improving memory handling to modifying the matrix storage so that nodes unnecessary to the calculation (those whose values are known) are not used when solving the linear system. It is also desirable to investigate whether or not direct solvers, such as LU decomposition, are feasible to implement on a GPU.

It would also be feasible to investigate if the method proposed in [43] is a good candidate for improving the execution of the SSRW when processing very large, high definition images. Particularly, it would be interesting to see how effective a GPU would be at performing both the eigenvector calculations and the system solution approximation. From a theoretical standpoint, these are good candidates for a GPU implementation since they require many operations occurring in parallel. If successful, the eigenvector computation could be performed quickly enough as a pre-processing stage such that it would not negatively affect the user.

Bibliography

[1] N.R. Pal and S.K. Pal, "A review on image segmentation techniques," *Pattern recognition*, vol. 26, no. 9, pp. 1277–1294, 1993.

[2] J.M. Marin, K. Mengersen, and C.P. Robert, "Bayesian modelling and inference on mixtures of distributions," *Handbook of Statistics*, vol. 25, pp. 459–507, 2005.

[3] R.O. Duda, P.E. Hart, and D.G. Stork, *Pattern classification*, Wiley New York, 2001.

[4] R.C. Gonzalez and R.E. Woods, *Digital image processing*, Prentice Hall, 2007.

[5] N. Otsu et al., "A threshold selection method from gray-level histograms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 9, no. 1, pp. 62–66, 1979.

[6] F. Meyer, "Topographic distance and watershed lines," *Signal Processing*, vol. 38, no. 1, pp. 113–125, 1994.

[7] J. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 22, no. 8, pp. 888–905, 2000.

[8] D. Anoraganingrum, "Cell segmentation with median filter and mathematical morphologyoperation," in *Image Analysis and Processing, 1999. Proceedings. International Conference on*, 1999, pp. 1043–1046.

[9] S. Raman, CA Maxwell, MH Barcellos-Hoff, and B. Parvin, "Geometric approach to segmentation and protein localization in cell culture assays," *Journal of Microscopy*, vol. 225, no. 1, pp. 22, 2007.

[10] Yung-Yu Chuang, Brian Curless, David H. Salesin, and Richard Szeliski, "A bayesian approach to digital matting," in *Proceedings of IEEE CVPR 2001.* December 2001, vol. 2, pp. 264–271, IEEE Computer Society.

[11] P. Hillman, J. Hannah, and D. Renshaw, "Alpha Channel Estimation in High Resolution Images and Image Sequences," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition.* IEEE Computer Society; 1999, 2001, vol. 1.

[12] Mark A. Ruzon and Carlo Tomasi., "Alpha estimation in natural images," in *IEEE Conference on Computer Vision and Pattern Recognition, 2000. Proceedings.*, 2000, vol. 1.

[13] J. Sun, J. Jia, C.K. Tang, and H.Y. Shum, "Poisson matting," *ACM Transactions on Graphics (TOG)*, vol. 23, no. 3, pp. 315–321, 2004.

[14] J. Wang and M. Cohen, "Optimized color sampling for robust matting," in *Proceedings of IEEE CVPR*, 2007.

[15] J. Sun, Y. Li, S.B. Kang, and H.Y. Shum, "Flash matting," in *International Conference on Computer Graphics and Interactive Techniques.* ACM Press New York, NY, USA, 2006, pp. 772–778.

[16] Y.Y. Chuang, A. Agarwala, B. Curless, D.H. Salesin, and R. Szeliski, "Video matting of complex scenes," in *Proceedings of the 29th annual conference on Computer graphics and interactive techniques.* ACM Press New York, NY, USA, 2002, pp. 243–248.

[17] E.N. Mortensen and W.A. Barrett, "Intelligent scissors for image composition," in *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques.* ACM New York, NY, USA, 1995, pp. 191–198.

[18] E.N. Mortensen and W.A. Barrett, "Interactive Segmentation with Intelligent Scissors," *Graphical Models and Image Processing*, vol. 60, no. 5, pp. 349–384, 1998.

[19] J. Wang and MF Cohen, "An Iterative Optimization Approach for Unified Image Segmentation and Matting," in *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, 2005, vol. 2.

[20] L. Grady, T. Schiwietz, S. Aharon, and R. Westermann, "Random walks for interactive alpha-matting," *Proc. VIIP05*, 2005.

[21] L. Grady, "Random Walks for Image Segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1768–1783, 2006.

[22] Dheeraj Singaraju, Leo Grady, and René Vidal, "Interactive image segmentation of quadratic energies on directed graphs," in *Proc. of CVPR 2008*. IEEE Computer Society, June 2008, IEEE.

[23] Y. Boykov, O. Veksler, and R. Zabih, "Markov random fields with efficient approximations," in *Computer Vision and Pattern Recognition, 1998. Proceedings. 1998 IEEE Computer Society Conference on*, 1998, pp. 648–655.

[24] Y. Boykov, O. Veksler, and R. Zabih, "Fast Approximate Energy Minimization via Graph Cuts," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1222–1239, 2001.

[25] Y. Boykov and M.P. Jolly, "Interactive graph cuts for optimal boundary and region segmentation of objects in ndimages," in *International Conference on Computer Vision.* Vancouver, BC, Canada, 2001, vol. 1, pp. 105–112.

[26] A. Agarwala, M. Dontcheva, M. Agrawala, S. Drucker, A. Colburn, B. Curless, D. Salesin, and M. Cohen, "Interactive digital photomontage," *ACM Transactions on Graphics (TOG)*, vol. 23, no. 3, pp. 294–302, 2004.

[27] V. Kolmogorov and R. Zabih, "Computing visual correspondence with occlusions using graph cuts," in *International Conference on Computer Vision*, 2001, vol. 2, pp. 508–515.

[28] L.R. Ford and D.R. Fulkerson, "Maximal flow through a network," *Canadian Journal of Mathematics*, vol. 8, no. 3, pp. 399–404, 1956.

[29] Y. Li, J. Sun, C.K. Tang, and H.Y. Shum, "Lazy snapping," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 303–308, 2004.

[30] C. Rother, V. Kolmogorov, and A. Blake, ""GrabCut": interactive foreground extraction using iterated graph cuts," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 309–314, 2004.

[31] K. McLaren, "The development of the CIE 1976 (L* a* b*) uniform colour-space and colour-difference formula," *Journal of the Society of Dyers and Colourists*, vol. 92, pp. 338–341, 1976.

[32] Y. Boykov, O. Veksler, and R. Zabih, "Efficient approximate energy minimization via graph cuts," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 12, pp. 1222–1239, 2001.

[33] V. Kolmogorov and R. Zabin, "What energy functions can be minimized via graph cuts?," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 2, pp. 147–159, 2004.

[34] Y. Boykov and V. Kolmogorov, "An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 9, pp. 1124–1137, 2004.

[35] A.P. Witkin, "Scale-Space Filtering," *Readings in Computer Vision: Issues, Problems, Principles, and Paradigms*, 1987.

[36] J. Sporring, M. Nielsen, L. Florack, and P. Johansen, *Gaussian Scale-Space Theory*, Kluwer Academic Publishers, 1997.

[37] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in *Computer Vision, 1998. Sixth International Conference on*, 1998, pp. 839–846.

[38] Adobe Systems Inc., "`http://www.adobe.com/products/aftereffects/`," .

[39] NVIDIA Corporation, "`http://www.nvida.com`,"

[40] Sanjit K. Mitra, *Digital Signal Processing: A Computer-Based Approach*, chapter 7, pp. 361–363, McGraw-Hill, 3rd edition, 2006.

[41] M.R. Hestenes and E. Stiefel, "Methods of conjugate gradients for solving linear systems," *J. Res. Nat. Bur. Stand*, vol. 49, no. 6, pp. 409–436, 1952.

[42] J.R. Shewchuk, "An introduction to the conjugate gradient method without the agonizing pain," *Computer Science Tech. Report*, pp. 94–125, 1994.

[43] Leo Grady and Ali Kemal Sinop, "Fast approximate random walker segmentation using eigenvector precomputation," in *Proc. of CVPR 2008*. IEEE Computer Society, June 2008, IEEE.