Semi-Automatic Depth Map Generation in Unconstrained Images and Video Sequences for 2D to Stereoscopic 3D Conversion

by

Raymond Phan

Bachelor of Engineering, Computer Engineering, Ryerson, 2006

Master of Applied Science, Electrical & Computer Engineering, Ryerson, 2008

A dissertation

presented to Ryerson University

in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

in the Program of

Electrical & Computer Engineering

Toronto, Ontario, Canada, 2013

Semi-Automatic Depth Map Generation in Unconstrained Images and Video Sequences for 2D to Stereoscopic 3D Conversion

Doctor of Philosophy 2013

Raymond Phan

Electrical & Computer Engineering

Ryerson University

Abstract

In this work, we describe a system for accurately estimating depth through synthetic depth maps in unconstrained conventional monocular images and video sequences, to semi-automatically convert these into their stereoscopic 3D counterparts. With current accepted industry efforts, this conversion process is performed automatically in a "black box" fashion, or manually converted using human operators to extract features and objects on a frame by frame basis, known as rotoscopers. Automatic conversion is the least labour intensive, but allows little to no user intervention, and error correction can be difficult. Manual is the most accurate, providing the most control, but very time consuming, and is prohibitive for use to all but the largest production studios. Noting the merits and disadvantages between these two methods, a semi-automatic method blends the two together, allowing for faster and accurate conversion, while decreasing time for releasing 3D content for user digest. Semi-automatic methods require the user to place user-defined strokes over the image, or over several keyframes in the case of video, corresponding to a rough estimate of the depths in the scene at these strokes. After, the rest of the depths are determined, creating depth maps to generate stereoscopic 3D content, and Depth Image Based Rendering is employed to generate the artificial views. Here, depth map estimation can be considered as a multi-label image segmentation problem: each class is a depth value. Additionally, for video, we allow the option of labeling only the first frame, and the strokes are propagated using one of two techniques: A modified computer vision object tracking algorithm, and edge-aware temporally consistent optical flow.

Fundamentally, this work combines the merits of two well-respected segmentation algorithms: Graph Cuts and Random Walks. The diffusion of depths, with smooth gradients from Random Walks, combined with the edge preserving properties from Graph Cuts can create the best possible result. To demonstrate that the proposed framework generates good quality stereoscopic content with minimal effort, we create results and compare to the current best known semi-automatic conversion framework. We also show that our results are more suitable for human perception in comparison to this framework.

Acknowledgements

It has been 11 long years spent at Ryerson University - many of these years have been spent studying, interacting with many people, gaining friends, teaching and hosting tutorial sessions, and researching. It is impossible to quantify how many people I have met throughout the years that have made a difference in my life, or whom I have had the pleasure of getting to know. However, out of all of these individuals, the one person who, to this day, still garners the greatest respect I have ever had in a person, have had the pleasure to know and work with is my supervisor, my friend, my mentor, and distinguished professor, Dr. Dimitri Androutsos, to whom I have served with and was his diligent Masters and Ph.D. student for the last 7 years. Without his guidance and knowledge, I would not be the individual, or graduate student, that I am today. He had the greatest impact during my academic career at Ryerson as a student, researcher, as an individual, as well as increasing my publication record by leaps and bounds to what it was several years ago. His character, personality and advice given to me both as a graduate student, and as a human being was both insightful, and downright humble. He has treated me unlike any professor I have met - as their equal. All of these wonderful things led me to the reason of pursuing my Ph.D., and to embark in a career of research and development. No amount of words can convey the respect and gratitude that I have towards him.

Most of the years I have spent at Ryerson were in the Multimedia & Distributed Computing Laboratory (MDC) (EPH 408), working on my thesis, engaging in projects with the industry, prepping for any teaching work, or socializing with fellow graduate students. Among these in my Ph.D. journey were Alexander Babalis, Mohammad Fawaz, and Richard Rzesuztek, fellow students of Dr. Androutsos of the MDC. Alexander, whom I learned the latest and greatest technologies in stereoscopic 3D, has been great to converse with, and is an effective and diligent research. Mohammad, who was a former student of mine in one of the courses I taught, is one of the brightest students I have ever known. His tenacity for completing tasks, his natural gift in problem solving, and for always learning something new whenever I speak to him is what makes him one of the best people to be friends with. Finally with Richard, I attribute a lot of my publication record and research career to him. He is the reason why most of this thesis came to fruition, as we exchanged ideas about the core fundamentals regarding how my thesis works. He was also integral in comments on my various publications, suggestions to make my work better, and kept me apprised in the latest state-of-the-art on technologies related to my thesis. With Richard, I am eternally grateful. I would also like to thank the following graduate students (in no particular order), who have been around for as long as I have and have made my time at Ryerson more enjoyable: Barry Vuong, Thomas Behan, Adrian Bulzacki, Victor Dumitriu, Hamed Rasouli, Joseph Santarcangelo, Ning Zhang and Ronnie Wong. I would also like to thank Danoush Hosseinzadeh and April Khademi, two wonderful people, who were former graduate students, that kept me in perspective whenever I had doubts about my graduate studies - especially April, as she is the reason why I received government funding.

In addition to the wonderful people I have met at Ryerson, I would like to thank the Natural Sciences and Engineering Research Council (NSERC) for awarding me with the Vanier Canada Graduate Scholarship, the most prestigious award for Ph.D. study in Canada, to which my research (and this thesis) would not be possible without their support. In terms of my thesis, I would like to thank Nicole Brosch from the Technical University of Vienna who provided valuable insight and details on how to implement the conversion framework I am comparing to, as the details described in their paper were quite ambiguous. There were many undergraduate students I have instructed throughout the years, both serving as a Teaching Assistant, and a Part-Time Instructor. I had the chance to know most of them on a personal level, and with each one, they have made a difference in terms of who I am now as an individual. Though there are many to thank, to which I do not have the luxury of naming them here, I am eternally grateful for all you have done.

Finally, I would like to thank my parents, my in-law family, and my wife - Rebecca Sauro. As I get older (and hopefully wiser), I realize with each and every day that they all have made sacrifices to allow me to pursue my dreams, and how hard (and how patient!) they were to allow me to reach this point. They have instilled a strong sense of family values, as well as cultural values, integrity and honour. When I move past this point in my life, they will be the ones that I will always remember, and finishing my studies would not be possible without them.

Dedication

To my dearest niece, Ava Jade Sauro, who has been a constant source of happiness, with the first moment of her being born as I began my Ph.D. studies. When the situation seemed grim, or when aspects about my research became frustrating, she has always managed to put a smile on my face. To my dearest wife, Rebecca Sauro, whom without her, I would not be where I am academically, professionally, and mentally. Through many late nights of research and implementation, she has always kept me objective, and has always been a constant source of companionship and love. Without her, the completion of this thesis would not be possible.

"Shooting The Hobbit in 3D is a dream come true.
If I had the ability to shoot the Lord of the Rings in 3D,
I certainly would have done it. The reality is, it's not
that difficult shooting in 3D. I love it when a film draws
you in and you become part of the experience,
and 3D helps immerse you in the film."

Peter Jackson, Film Director - *The Lord of the Rings, The Hobbit*

Contents

List of Tables

List of Figures

List of Appendices

Chapter 1

Introduction

2D to 3D conversion is the process of taking conventional images or videos captured by a single camera, and artificially transform them so that the illusion of depth is seen by human observers. 2D to 3D conversion has seen a recent surge in popularity over the last few years due to the recent proliferation of big budget 3D movies, or with some portions of the film being offered in 3D. Such examples are Avatar (2009), Tron Legacy (2010), Transformers: Dark of the Moon (2011), Harry Potter: The Deathly Hallows (2010 and 2011), The Dark Knight (2012) and The Hobbit (2013). This surge in popularity is due to the enhanced experience that the viewer is absorbed in when watching the film, as 3D movies attempt to mimic the sense of being immersed in the scene, as if the viewer was present in the environment. This previous movie list is hardly exhaustive, and quite often, if a film is of the action genre, adventure genre, or is animated, it will inevitably be released as a 3D film, alongside a standard "2D" release. The push for 3D in theatres is also being complimented by the increase in the number of 3D-capable or 3D-Ready televisions, with 3D-capable media players that embrace the Multiview Coding (MVC) standard, that is part of the H.264 / AVC standard [2]. It is within these two realms of advancement that allow for the efficient storage and transmission of stereoscopic content, as well as the capability of viewing and easily obtaining the necessary hardware to do so.

Despite its surge in popularity, some viewers are very skeptical with regards to the enhanced experience when they watch stereoscopic 3D films. In particular,Roger Ebert, a well-known film critic for the Chicago Sun Times, wrote an article describing his abhorrence for 3D, and why it should not be used as a means of entertainment. He lists various reasons, which include nausea and headaches, as well as visual discomfort [3]. Despite his opinion, which is shared by certain viewers, many others thoroughly enjoy the 3D movie experience, as evidenced by the detailed analysis of stereoscopic 3D movie sales done by Quantel [4], which is a company that manufactures high-end visual effects and editing systems for film and television. This analysis shows that certain movies that did not fare well in the standard 2D release, actually performed much better when it was released in 3D. The Polar Express (2004) is a prime example of this particular situation.

While much of the initial focus in stereoscopic 3D was on creating the hardware for display, and the infrastructure and standards necessary for the dissemination of this content, attention has now shifted to efficient methods of creating and editing stereoscopic 3D content. The current stage of technologies for 3D display and for user digest is well established, such as 3D enabled Digital Light Processing (DLP) TVs and projectors [5][6], IMAX 3D theatres [7], stereoscopic computer monitors for gaming [8], and so on. However, the current accepted methods to produce 3D content for human digest are either expensive, time consuming, or have a limited niche of interested individuals, and have thus prohibited widespread production.

1.1 Introduction to Stereoscopy

Before going into 2D to 3D conversion in more detail, due to the fact that the whole point of the process is to create the illusion of depth in the scene, it is required to establish the necessary vocabulary in stereoscopy before proceeding. Stereoscopy is a technique for creating, or enhancing the illusion of depth in an image when viewed with two eyes by a human observer. To properly perform stereoscopy, most methods present two offset images separately to the left and right eye of the viewer. Humans view the world using two views (also known as binocular vision), where the left view is one view of the scene, and the right view is a slightly horizontally (or sometimes with the additional of some vertical components) shifted version of the left view. These views are presented to each eye, they are processed by the visual cortex in our brain, and we thus perceive depth [9][10]. Stereoscopy is quite different from viewing *3D displays*, which displays an image in three full dimensions, allowing the observer to increase information intake about the three-dimensional objects being displayed by performing head and eye movements.

Human vision, including how we perceive depth, is a complex process which only begins with obtaining visual information through the eyes. A good majority of the processing is performed within the brain, as it attempts to make intelligent and meaningful sense of the information gained. One of the very important visual functions that occur within the brain as it interprets what the eyes see, is that of assessing the relative distances of various objects from the viewer, and the depth dimension of those same perceived objects. The brain makes use of a number of *depth cues* to determine relative distances and depth in a perceived scene. These include:

- Stereopsis: For human beings, the information obtained from the eyes is put to use by exploiting the different projection of objects onto each eye to judge depth. By using two images of the same scene obtained from slightly different angles, it is possible to triangulate the distance to an object with a high degree of accuracy. As mentioned previously, each eye views a slightly different angle of an object seen by the left and right eyes. This happens because of the *horizontal separation parallax* of the eyes, and is related to the *disparity* between the two views. Disparity in this sense refers to the difference in image location of an object by the left and right eyes. Essentially, objects seen in one view are in a slightly different position in the right view. This displacement is processed by the brain in order to perceive depth. Horizontal separation parallax demonstrates that if an object is far away, the disparity between the two views presented to both eyes will be small. If the object is close or near, the disparity will be large. It is stereopsis that creates the illusion of depth.

- Eye convergence: Due to stereopsis, the two eyes focus on the *same object.* In doing so, the eyes *converge* to the same object. The angle of convergence, or the angle subtended by the eyes of an observer at the point of focus, is smaller when the eye is fixating on far away objects.
- Eye accommodation: This is what happens when focusing on a near object, and then immediately looking at a distant object after. At first, the far object seems to be blurred, but is immediately put into focus after. This is automatically performed by our visual system, and many properties of the eye change during accommodation, such as the lens shape and pupil size. Accommodation also helps humans distinguish between near and far objects, and with perceiving depth as well.
- Overlapping one object by another: It is natural to see that when one object is in front of another, this object is perceived to be closer to the viewer than the object that is behind. This is related to the concept of *occlusions*, a well-established concept in computer vision.

All of these concepts must adequately be modelled when performing 2D to 3D conversion, as the video itself is a 2D projection of a 3D world. The material is presented on a flat surface, and so stereoscopy must be able to recreate the above on a surface which inherently has no depth. In addition, stereoscopy is directly related to *depth perception*, which is the visual ability to perceive the world in three dimensions, and the distance of an object to the viewer. In retrospect, 2D to 3D conversion attempts to take conventional images and videos, and mimic the way human beings perceive the world, thus mimicking the same depth perception as viewing a natural scene. This enhanced experience by exploiting depth perception is the main attraction for viewing 3D films. Unlike traditional films, 3D films are subject to a number of constraints that arise from properly generating distinct views for the eyes. For example, the size of the viewing screen can have an impact on the overall perceived depth for a scene. If there are any depth errors, they may not be so apparent on a screen the size of a mobile device, but may become more apparent when viewed on a cinema screen. There have been remapping techniques to ensure a consistent perception of depth, such as the method by Lang *et al.* [11]. They propose four simple operators for remapping, but they do not consider relative depth changes among neighbouring features, and also content preservation, such as planes, and left/right frame coherence. This is successfully addressed in Yan *et al.* [12], and was shown to outperform [11].

1.2 Relation Between Depth and Disparity

Depth and perceived disparity are quite related to each other. Before we continue with our discussion of 2D to 3D conversion, this concept must also be understood first. To sense an illusion of depth in an image, or in a video, this is created by simulating *stereo parallax*, where in human vision, objects in a scene that appear in one eye, appear in a different position in the other eye. For this case, depth and disparity are very much related in the context of *stereo correspondence*, which is the method for determining the correct displacement of these objects [13]. With this idea, because 2D to 3D conversion requires capturing content by cameras, a camera can essentially be modelled as an "eye" or view of the human visual system, and we thus turn to the use of cameras or sensors to capture and process images.

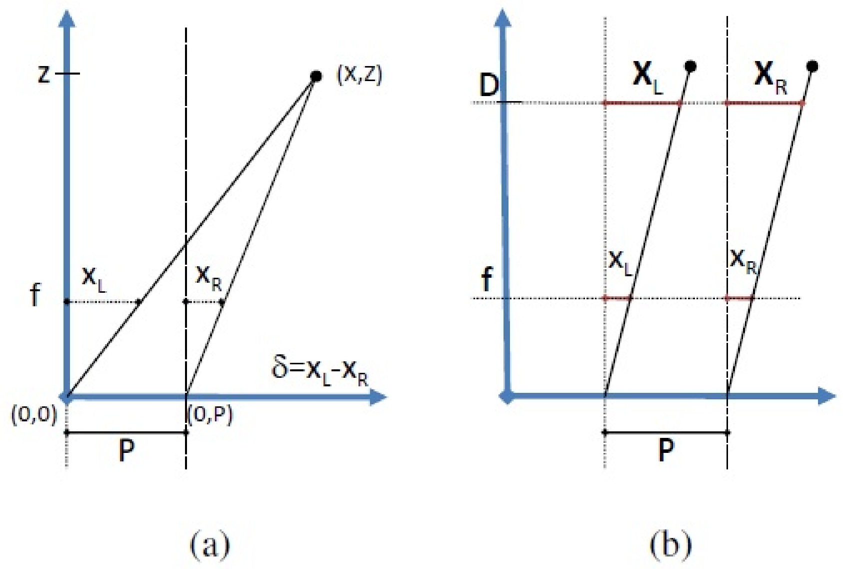


Figure 1.1: (a) "Bird's eye view" of a pair of cameras or sensors capturing a 3D scene using the pinhole camera model. (b) The same configuration as (a), but the scene is now a flat screen, ultimately mirroring how a human would be viewing a stereoscopic monitor or screen

Consider the camera or sensor configuration, when observed above the scene (i.e. a "bird's eye view"), shown in Fig. 1.1(a).

It should be noted that the scene is of the real world, with 3D spatial co-ordinates. This is a simplified model, with two aligned cameras or sensors, by using the pinhole camera model. These two sensors correspond to the positions of where the left and right eyes would be situated at. Essentially, the pinhole camera model is a camera that has an infinitesimally small aperture, allowing only one ray of light corresponding to one point in a 3D scene [9]. Modern Charged Coupled Device (CCD) cameras have lenses, where multiple rays of light correspond to a point in a 3D scene, allowing for better focus. However, under most conditions, the pinhole camera model is accurate enough formulation. These sensors are separated by a distance of f from the image plane, known as the *focal length.* In this aspect, the *image plane* is where the image is formed when observing the scene, and is captured for storage and processing for later. The *aperture* is a hole or an opening to allow light to travel. The aperture size is directly related to the focus of an object, and as stated previously, the aperture is infinitesimally small to allow one ray of light to focus on one point from the 3D scene. f is the distance between the aperture, and the image plane, and is also a measure of how strongly the camera or sensor converges or diverges light. As the sensors are aligned, and these are observed from above, a point in the scene is represented as a pair of (X, Z) co-ordinates: X representing the horizontal displacement of the point, and Z is the depth of the point. P represents the interpupillary distance, or the distance between the two cameras or sensors. Using the pinhole camera model, we can trace a ray to both sensors, and we can thus introduce two new quantities: x_L and x_R. These denote the horizontal distance from the left sensor to the ray

(x_L), and from the right sensor to the ray (x_R). The disparity in this model, δ, is defined as $x_L - x_R$. By using similar triangles in Fig. 1.1(a), we can determine the following relationships:

$$\frac{Z}{f} = \frac{X}{x_L} = \frac{X-P}{x_R}$$
$$\therefore \frac{Z}{f} = \frac{P}{\delta} \Rightarrow Z = \frac{fP}{\delta} \tag{1.1}$$

When viewing stereoscopic content, the perceived disparity is a result of projecting the scene to the two sensors. This projection results in showing two different images to each sensor from the scene. As we wish to view content stereoscopically, we will ultimately view this content on a *flat screen*. Assuming that the flat screen is parallel to the image plane of the two sensors, this new model is shown in Fig. 1.1(b). D is the distance from the image plane to the screen, and X_L/X_R is the point on the screen that is closest to the left/right sensor, and the screen point being projected to the left/right eye. x_L and x_R are the same as before. Again, using similar triangles, we thus have the following relationships:

$$\frac{D}{f} = \frac{X_L}{x_l} = \frac{X_R}{x_R} \tag{1.2}$$

Using Fig. 1.1(b), the on-screen disparity, or the distance between the two points on the screen, is given as $D_x = X_R + P - X_L$. Combining Eq. 1.1 and Eq. 1.2, we obtain that in order to provide a depth perception that descries an object at a distance of Z units from the sensors, it is necessary to have a screen disparity D_x that satisfies:

$$Z = \frac{PD}{P - D_x} \tag{1.3}$$

Eq. 1.3 shows a relation between the amount of disparity on the flat screen (ultimately the video), and the perceived depth. Specifically, the perceived depth depends on the distance of the screen from the viewer, the size of the screen, and the distance between the pupils of the viewer. Consequently, the larger the display, this requires a larger on-screen disparity to see the same effect. As such, a set of stereoscopic images for one set of viewing conditions may not be appropriate in another set of viewing conditions. In order to provide good viewing experience, one needs to be familiar with the conditions in which the stereoscopic video is to be viewed. It is not enough to enlarge the screen by the ratio of the distance-to-screen change, as P remains constant.

1.3 Overview of Generation Methods

With regards to films and television content, there are three methods that are the most well-known for creating stereoscopic 3D content [4]:

1. *Computer Generated Imagery (CGI) Films*: CGI films, such as those created by Pixar, Dreamworks SKG, etc. are amongst the most successful for generating a modest revenue within the 3D films

category, as evidenced by The Polar Express. The main reason why is due to the animated nature of the film. Because these films are mostly computer generated, generating novel views is very intuitive. The base model for most CGI is to move a virtual camera of the drawn scene in different positions. A novel view is generated with respect to that virtual camera. These films take the least amount of time to create in 3D, and is the least costly, as generating virtual views is quite intuitive and most of the parameters required to do so is given by the CGI base model.

2. *Native 3D Camera Films*: These 3D films are natively shot using a 3D camera, where there are two lenses roughly separated by the ocular distance between two eyes. The way this camera is set up is so that what is captured through the camera is what is naturally perceived by human perception. Therefore, very little post-processing work is required. This is in a class of its own, and the methodology is different in comparison to animated films. CGI films generate novel views by shifting a virtual camera in various positions, whereas native 3D camera films automatically capture two distinct views. Though this should produce the best quality in terms of depth perception, this is the most expensive route to take, as it requires specialized equipment.

3. *Post-processed, or 2D-to-3D Conversions*: In this category, films are shot using a normal single-view camera, and are processed using post-production techniques to be *converted* to 3D, and are commonly known as 2D to 3D Conversion - the subject of this thesis. Generating 3D films in this category is roughly in the middle of the three in terms of cost. However, with referencing Roger Ebert, this area of 3D cinema is the most controversial, and a good number of viewers blame their dislike for 3D because of this category. This makes sense, because when this kind of conversion is done improperly, it can lead to a number of disconcerting effects, such as parts of foreground objects appearing as the background or foreground objects taking on the appearance of cardboard cutouts, i.e. appear to have no internal depth.

   Well done conversions appear more natural and this category gives the director much more control over the final product. Regardless of the quality, the current industry accepted method for this kind of conversion is a very labour intensive process. Specifically, for a particular frame, a novel view must be generated only using the information of that particular frame, or some frames before or after the current frame if required. Virtual camera models are not possible as seen in animated films. Many artists and animators are required to manually segment foreground and background objects and fill in the disoccluded regions, or holes, that appear when the scene objects are moved around.

   As such, this current accepted method for performing 2D to 3D conversion is strictly a manual process, and is known as *rotoscoping*. An animator extracts objects from an image or frame, and manually manipulates them to create the left and right eye images. While this produces very convincing results, it is a difficult and time consuming process, and will inevitably be quite expensive, due to the large requirement of manual operators. This is very prohibitive to all but the largest of studios, and thus makes conversion difficult for smaller studios, amateur film makers, and even consumers.

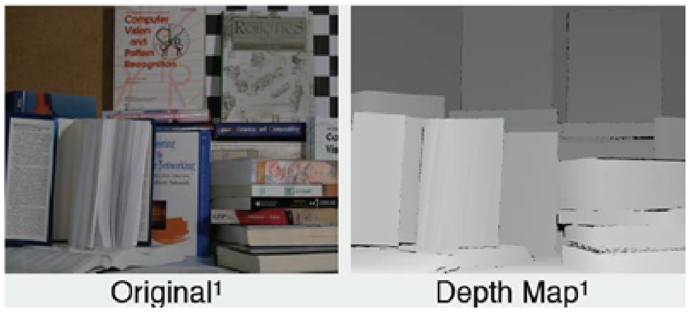


Figure 1.2: An example illustrating an image (left) with its associated depth map (right)

Despite the problems with 2D to 3D Conversion, this is a very important part of the stereoscopic post-processing process, and should not be dismissed. Stereoscopic filming can become difficult and expensive, and converting single-view footage into stereoscopic 3D can become useful in cases where filming directly in 3D is too costly, or difficult. One of the bigger appeals for using this technique is to convert older footage, such as legacy cartoons, or those well before the advent of 3D films, to be viewed with a 3D effect. Research into conversion techniques is on-going in order to minimize its labour-intensive process. The ultimate goal for conversion is to have a system that will automatically convert any 2D footage into 3D with minimal user input. Not only does this make the conversion more affordable, it allows for easier generation of 3D content.

Most research currently focuses on generating a *depth map*, which is a monochromatic image that shares the same dimensions as the image or frame to be converted to 3D. Each intensity, or brightness value, in the depth map represents the distance from the camera. Specifically, the greater the intensity, the closer the point in the image or frame is to the camera. Conversely, the darker the intensity, the farther the point is. Novel viewpoints can be generated from this depth map using Depth Image Based Rendering (DIBR) techniques, such as the method by Zhang *et al.* [14], or by Fehn *et al.* [15]. An example of what a depth map can look like is shown in Fig. 1.2 below [1].

1.4 Relevant Work

2D to 3D conversion algorithms have been the subject of many useful applications which are targeted for the industry, as well as the end user. Specifically, 2D to 3D conversion techniques are of primary consideration in the latest 3DTVs. This is due to the fact that 3D technology is now widely available for the home, and that the home owner can have the option of being able to experience their existing 2D content in 3D. There have been many paradigms for propagating 3D technology to be experienced at home. As an example, Redert *et al.* formed the Advanced Three-Dimensional Television System Technologies (ATTEST) group [16], which was the first ever attempt to introduce the 3DTV framework to the home. This included the paradigms for the transmission and reconstruction of the signals, the best way to capture stereoscopic content, and designing a compatible coding scheme for transmission.

[1]Taken from the Middlebury Stereo Database: `http://vision.middlebury.edu/stereo`

Similarly, Fehn *et al.* discuss the overall process in greater detail in [15]. However, the ATTEST group currently no longer exists, but there have been other efforts that envision the same goal to deliver 3D to the end user. Specifically, the Mobile3DTV initiative [17], which started in 2008, was created to deliver 3D content to mobile phones in Europe. It encompasses the same objectives as the ATTEST project, but for more specialized mobile devices. This group is still conducting research, with many European universities in collaboration with each other. However, though the method of delivering 3D content has a huge collaborative undertaking, this still does not solve how to deliver 2D content in 3D to the home user. There has, however, been research to develop a full stereoscopic signal processing pipeline, in order to post-process the 3D data received, such as the work done in [18], but we are thus brought to the same problem with the lack of 3D content available. There is a vast amount of 2D content available for user digest, which not only includes motion pictures and television shows, but standard computer vision and image processing applications can be augmented with a sense of depth, in order to give a more truer experience to the user.

In retrospect, the aim for this section is to discuss the most recent undertakings in 2D to 3D conversion to their stereoscopic counterparts, before addressing the current problems, and suggest methods to help improve current methods. These undertakings include the following, each concentrating on a particular feature or approach to solve the problem.

1.4.1 Using Motion

The first of the most predominant techniques that is used for 2D to 3D conversion is using motion estimation to determine the depth or disparity of the scene. The underlying mechanism is that for objects that are closer to the camera, they should move faster, whereas for objects that are far, the motion should be slower. Thus, motion estimation can be used to determine correspondence between two consecutive frames, and thus can be used to determine what the appropriate shifts of pixels are from the reference view (current frame), to the target view (next frame). However, the use of the actual motion vectors themselves to generate a stereoscopic video sequence varies between the methods, but the underlying mechanism is the same.

In Wang *et al.* [19], they perform some pre-processing first. Between pairs of frames, they perform a frame difference to eliminate static backgrounds. After, block-based motion estimation is used, where each pixel has a block centred around it, and a motion vector is determined in the area of where this block is closely matched with the most similar block in the frame. Some post-processing is also performed, such as morphological operators and so forth. Feng *et al.*[20] makes use of the compressed domain of MPEG-4. The MPEG-4 bitstream is analyzed to extract the motion vectors of each frame, which serves as an initial motion vector field. After, a refinement is performed by determining which motion vectors are noisy, or removing those vectors that are unreliable. With the removal of these vectors, some spatial smoothing is performed, as a last refinement step, to supplement the noise removal stage. With this final motion vector map, the depths can simply be described as the magnitude of the horizontal and vertical components. In Liu *et al.* [21], each shot is classified as either being a static scene, or one comprised of motion. If the scene is static, a geometric transformation is performed to shift pixels to create a stereoscopic image pair. For a motion scene, they extract motion vectors and use that as a preliminary measure of depth. After, they enhance the edges so that objects are well defined, dilate and region fill to fill in any holes, and perform smoothing to minimize noise. Finally, with Chen *et al.* [22], they fuse two depth maps together: one with motion estimation, and the other with using colour information. For motion estimation, they directly use the magnitude of the motion vectors, while for colour information, they use the Cr channel from the YCbCr colour space. This is loosely based on the work done by Tam *et al.* [23], who also use the Cr channel, and other pre- and post-processing operations to create depth maps. To merge both depths maps together, if the pixel experiences relatively small motion, the colour information is used to supplement, as this may be considered unreliable. If the motion at a particular pixel is relatively larger, then it is assumed that motion information for creating depth suffices.

1.4.2 Using Scene Features

Another popular method for converting 2D monocular video sequences into 3D is through analyzing the features of the scene of interest. Features such as shape, edges, or anything involving the direct use of features of any kind, other than motion are of interest here.

In Feng *et al.* [24], their work is comprised of three stages: 1) They use optical flow to determine an initial depth estimate, also using the flow to detect occlusions for better accuracy, 2) They perform object segmentation using improved region-growing from masks of determined depth layers and 3) a hybrid depth estimation scheme using content-based matching (inside a small library of true stereo image pairs). In Kuo *et al.*'s work [25], the objects within the scene are classified into one of four types: sky, ground, man-made and natural objects. An initial depth is assigned to each object by using vanishing points, a simplified camera projection model, and object classification algorithms. They finally refine the depth map by using a dark channel model: a model based on the light intensity reflected from the atmospheric haze commonly observed in outdoor images. In Wang *et al.*'s work [26], they use the concept of triangle meshes for estimate depth. This can only work if the scene has some sort of vanishing point. Essentially uniform triangles of various sizes are formed in the scene, using the vanishing point as the apex of all triangles. Within each triangle, with the use of the vanishing point, a depth gradient is determined to construct a smooth depth map over the entire frame. Next we consider the work by Han *et al.* [27], where they also use vanishing points and superpixels as geometric and texture cues respectively, and combine both cues to generate a depth map. Zhang *et al.* [28] develop a scheme modeling occlusion and visual attention. In this scheme, an initial depth model using the cue of relative height, a saliency map of visual attention and occlusion analysis are merged together to create a depth map. Yu *et al.* [29] first determine the depth of the static background scene using linear perspective cues (i.e. vanishing lines and points). After, moving objects are segmented separately, according to their positions in the scene. With the depth of the background and the moving objects, they are merged to create a single depth map. In Zhang *et al.* [30], the depth maps are first estimated in a multi-cue fusion manner by leveraging motion cues and photometric cues in video frames with a depth prior of spatial and temporal smoothness. After, the depth map is converted to a disparity map by considering both the display device size and the human stereoscopic visual perception constraints. The original 2D frames serve as the left views, and warp them to "virtual" right views according to the predicted disparity value. Finally, we look at the method presented by Schnyder *et al.* [31]. They create an automatic system for high quality stereoscopic video from monocular, or single view footage of field-based sports by exploiting context-specific priors, such as the ground plane, player size and known background. They create per-shot panoramas to ensure temporally consistent stereoscopic depth in video reconstructions. Players are rendered as billboards at correct depths on the ground plane. Their method uses additional sports priors to disambiguate segmentation artifacts and produce synthesized 3D shots that are in most cases, indistinguishable from stereoscopic ground truth footage.

1.4.3 Real-Time Methods

Another method that has seen a recent surge in interest is through the use of real-time algorithms implemented on specialized hardware, such as DSPs, LSI architectures or GPUs. Interestingly, there has been a literature survey on the requirements on what real-time 2D to 3D conversion requires in order to be successful in dissemination with the greatest amount of quality [32]. The general method is to first classify the frame into a particular category, as each scene's content can be different, and thus have different kinds of depth. The next step is depth analysis, where cues for depth perception are extracted from the input sequence. What follows is to use the depth cues to assign different depths to each element in the frame. Those elements can be pixels, blocks or connected regions. The resulting depth map must comply with requirements linked to the final 3D visual quality (i.e. properties of an edge map that guarantee good visual quality have not been documented). Perceptual depth inconsistencies such as breaking of continuous objects, or depth conflicts such as simultaneous uncovering and occlusion must be prevented at all cost. Next, depth control determines the offset and range of the depth map. The final step is the generation of a stereo pair from the depth map and the 2D input. This algorithm basically calculates the shift of the pixels on the left and right images, and then fills in the holes caused by uncovering and occlusions, using 2D spatial-temporal interpolation or image inpainting techniques, or is simply DIBR.

In Tsai *et al.* [33], their system performs real-time 1080p conversion by generating 3D depth information by fusing cues from edge feature-based global scene depth gradients and texture-based local depth refinement. By combining the global depth gradient and local depth refinement, generated 3D images have comfortable and vivid quality, and algorithm has very low computational complexity. They perform this on a system with a multi-core CPU and a GPU. To optimize performance, they use several techniques including unified streaming dataflow, multi-thread schedule synchronization, and GPU acceleration for depth image-based rendering (DIBR). Ramos-Diaz *et al.* [34] develop an algorithm programmed onto a DSP. The algorithm is based on a disparity map computation and an anaglyph synthesis. The disparity map is first estimated by employing the wavelet atomic functions technique at several decomposition levels in processing a 2D video sequence. After, anaglyph synthesis is used to apply the disparity map in a 3D video sequence reconstruction. Finally, we look at the algorithm by Lai *et al.* [35]. They propose a hybrid depth-perception algorithm consisting of three parts: moving object of border extraction, vanish line detection, and smoothing of the depth map. Furthermore, the proposed algorithm was designed to be placed on VLSI architecture.

1.4.4 Other 2D to 3D Conversion Methods

Other methods for converting 2D monocular video sequences that do not fall into any of the aforementioned categories are discussed here.

A good example of this is the work by Konrad *et al.* [36][37]. The rationale behind their approach is that automatic methods have not been widely accepted yet, and semi-automatic methods can be time consuming, even to the most skilled manual operator. They explore a radically different approach inspired by their work on saliency detection in images. Instead of relying on a deterministic scene model for the input 2D image, they attempt to "learn the model from a large dictionary of stereopairs, such as YouTube 3D. Essentially, the main observation they make is that among millions of stereopairs available on-line, there likely exist *many* stereopairs whose 3D content matches that of the 2D input (query). Using this reasoning, if two stereopairs whose left images are photometrically similar are likely to have similar disparity fields. This is essentially an image retrieval system, where the input query is a particular image they want to convert into 3D. The key difference is that the output is not a set of similar images, but they are a set of *depth maps* whose images that they were derived from are *photometrically* similar to the input query. Essentially, they find a number of on-line stereopairs whose left image is a close photometric match to the 2D query and then extract depth information from these stereopairs. Since the depths for the selected stereopairs differ due to differences in underlying image content, level of noise, distortions, and so on, they combine them by using the median.

The last method we discuss here is the work by Park *et al.* [38]. They present a system that detects "good" stereo frames from a 2D video, which was captured a priori without any constraints on camera motion or content. Essentially, the goal is to identify from a 2D video, the content that would present appreciable 3D effect to a human observer, *without* recovering the actual 3D depth. They use a trained classifier to detect pairs of video frames that are suitable for constructing stereo images. In particular, for a given frame I_t at time t, they determine if $\hat{t}$ exists such that $I_{t+\hat{t}}$ and I_t can form an acceptable stereo image. This is perhaps the first recent work that tries to create a stereo video sequence using frames separated by time offsets of a single 2D video sequence.

1.5 Method of Focus: Semi-Automatic

With the plethora of methods seen for 2D to stereoscopic 3D conversion, one method has recently seen a surge of interest within the last few years, and is ultimately the method we focus on in our approach. At the present time, most methods focus on the automatic viewpoint to extract depth information from an image or frame. Specifically, most conversion methods are implemented on dedicated hardware, built into 3D displays, and do conversion "on-the-fly", with variables and settings that are not tunable. As such, though these may work for some select footage, they perform at a substandard level in comparison to current accepted method of manual conversion by rotoscopers. By not allowing certain variables and settings to be available to the user, even when minimizing the amount of user intervention for faster conversion, these can become extremely difficult to control the results of the conversion. Also, any errors that occur in the process cannot be easily corrected. No provision is in place to correct objects that appear at the wrong depths while converting, and may require extensive pre-processing or post-processing to correct. Therefore, there are advantages to pursuing a user-guided, *semi-automatic* approach to 2D to 3D conversion. In this approach, for a single image or frame, the user simply marks objects and regions to what *they believe* is close or far from the camera, denoted by lighter and darker intensities respectively. One question that often arises is whether or not the depths that the user mark are indeed the right depths for perception in the scene. However, the depth labeling does not need to be accurate; it only has to be perceptually consistent, and more often than not, the labeling of the user will meet this criteria [1][39]. The end result is that the depths for the rest of the pixels in the image are estimated using this information. For the case of video, the user marks certain keyframes, each in the same fashion as the single image. The end result is that the rest of the depths over the entire video are estimated. By transforming the process into a semi-automatic one, this will allow the user to correct depth errors that surface during the evolution of the algorithm, should they arise. This is the ultimate reason as to why one would focus on semi-automatic methods: To allow for faster and more accurate 2D to 3D conversion, providing a more cost-effective and economical solution, and serving as a happy medium between complex methods, such as rotoscoping, to fully automatic ones. With respect to semi-automatic, some form of user input will inevitably allow the accuracy of conversion to increase, as human beings are the best "machines" in determining what is close and what is far from the camera, an issue that is one of the biggest unsolved, and open-ended problems in computer vision that exist today.

In addition, accurate 2D to 3D conversion should be performed on *any* image or video sequence supplied to the algorithm, and are hence *unconstrained*. Unconstrained images and videos are those that have been captured by an acquisition device in natural environments where there are factors beyond one's control. For these images and videos, the objects within the scene can be subject to many uncontrollable factors. High amounts of deformation, and large variations of colour depending on illumination conditions are inevitable. In addition, the objects can be of an arbitrary size, and can be in any location when the scene is captured. In other words, the content that can serve as input into the system can be *anything*, instead of focusing on a specific set of examples that match certain conditions and requirements. If a 2D to 3D conversion algorithm operates on unconstrained media, then it will be quite successful in the conversion of most media in an efficient and accurate manner.

1.6 Current Accepted Method: Guttmann *et al.*

There are some methods well known in this realm of conversion. In particular, the work by Guttmann *et al.* [1] is one that solves for the depths in a video sequence by marking strokes only on the first and last frames. Guttmann *et al.*'s work is arguably the first method that explores 2D to 3D conversion in this viewpoint, and is the method of focus for this thesis. In addition, this is the baseline algorithm for numerous semi-automated depth map generation methods, and so we concentrate on comparing performance and accuracy with respect to this work. In addition, for these recent methods, there are many key implementation details that are omitted, and so a faithful representation of their system and the results that are presented in their work would not be possible. In addition, an exact comparison between the proposed method and these recent methods would be inconsistent in illustrating the differences between them.

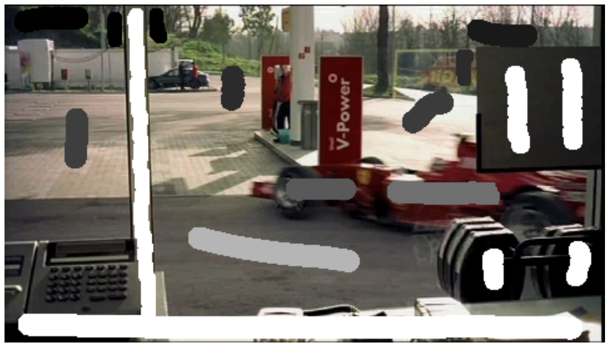
Essentially, to solve for the depths, Guttmann *et al.*, solve for the disparities first, and DIBR is used to render the views. Unfortunately, the framework suffers a myriad of problems, with each problem a further justification as to why the framework that is proposed in this thesis is more intuitive and accurate. In this chapter, we discuss each problem in detail, with their solutions as components in the overall conversion framework of this thesis. However, before we do so, we provide a brief overview of how Guttmann *et al.*'s framework functions so that the problems associated with the framework are better understood.

The first step is to detect *anchor* points in all of the frames. As stated earlier, the framework only requires depth strokes to be placed on the first and last frames. This information is used to first create a rough estimate of the depth for the first and last frame is found. Using these rough estimates, Support Vector Machine (SVM) classifiers [40] are trained on the estimates of both frames separately. Each SVM is trained on classifying a unique depth seen in the rough estimate, using the Scale-Invariant Feature Transform (SIFT) [41], combined with the grayscale intensity at the particular point as the input space. To detect anchor points, SIFT points are detected throughout the entire video sequence, and are put through the SVM classifiers. For each SIFT point, a One-Vs-All scheme is employed, choosing the highest similarity out of all the classifiers. If this surpasses a high-confidence threshold, this point is an anchor point, and the depth that is assigned is the one belonging to the SVM classifier tuned for the particular depth. To solve for the rest of the depths in the video sequence, an energy function is minimized by least squares, where the function is a combination of spatial and temporal terms, edge terms, colour information, the anchor points and user strokes. Least squares minimization is performed by transforming the given information into a sparse linear system of equations, and directly solved. A similar method was by Wang *et al.* [42]. However, what is unique about [42] is that holes / occlusions are handled quite well, where they are simply sewn or snapped together with nearby non-occluding areas.

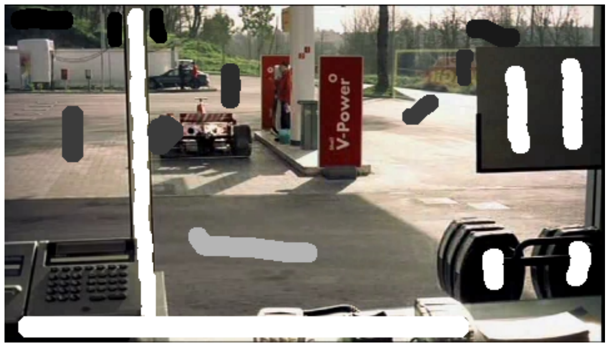
1.7 Problems with Guttmann *et al.*

1.7.1 SVM Training Step

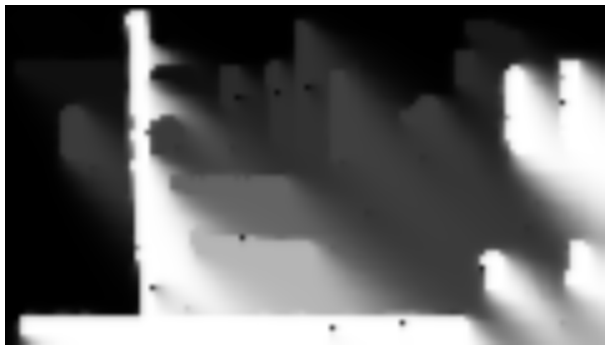
With all classification schemes, there is the inevitable property of misclassifications. As part of Guttmann *et al.*'s framework, anchor points are detected, which are based on training SVM classifiers, with SIFT points combined with grayscale intensities as the input space. It is inevitable that some points may have the same SIFT descriptor, but may lie on completely different areas of depth. Because of these misclassifications, the final depth maps will also be inaccurate. With the proposed framework, we do away with any classifier training and classification, and rely on the user to provide the right amount of partial information to create the best results possible. By introducing minimal user effort, and possibility of error through using classifiers, results that are more agreeable to human perception are produced. In



First Frame



Last Frame



Sample Depth Map

Figure 1.3: Illustration of errors by Guttmann *et al.*'s framework - SVM Training Step

addition, should the user not be satisfied with the results, small amendments can be added to improve them.

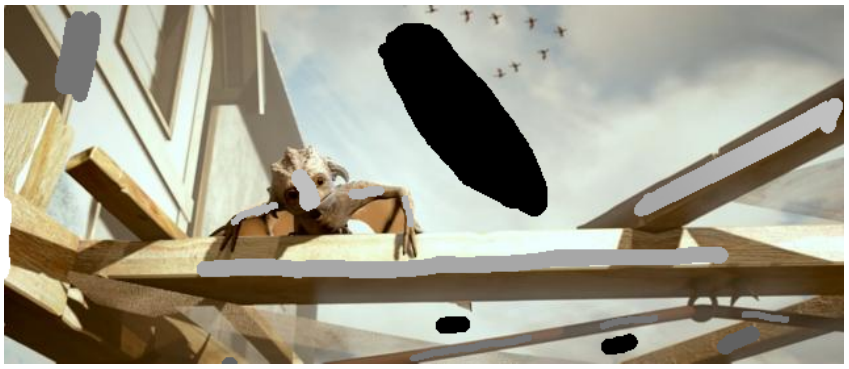
As an illustration of these errors, Fig. 1.3(a) and (b) shows an example of a video sequence where the first frames and last frames are shown, with their corresponding user-defined scribbles. Finally, Fig. 1.3(c) shows one of the depth maps that resulted from this sequence. With reference to Fig. 1.3, there are several "holes" in different areas of the depth map. These are likely due to the anchor points being detected, throughout the frame, which were classified as a different depth. As stated earlier, some points may have the same SIFT descriptor, but can lie on different areas of depth, and the "holes" that are seen throughout the frame are evidence of this fact. This could be rectified by region-filling methods, but the proposed thesis does not require such a processing step.

1.7.2 Improper Edge Definition

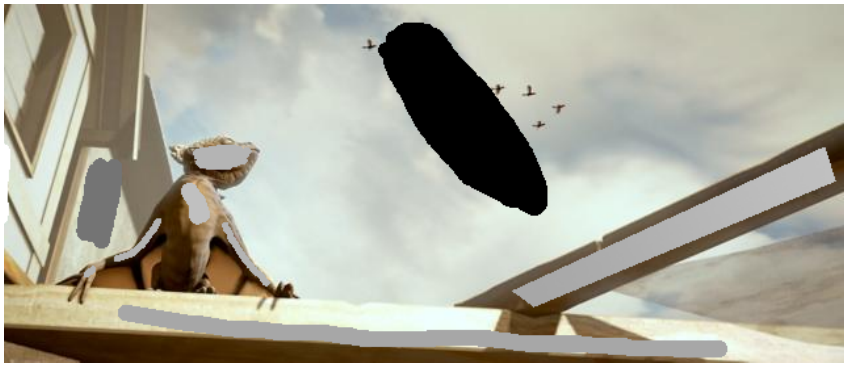
Guttmann *et al.* use edge strength as a component for their linear system of equations to solve for the depths. Edge strength will aid in ensuring that the depths of certain regions will not “leak” into other areas. Unfortunately, the edge definition that is used is not suitable for colour images. Usually, edges in images are defined as determining points where the image brightness changes sharply, or more formally, has discontinuities. These are usually detected by calculating discrete approximations to the horizontal and vertical gradients in spatial neighbourhoods surrounding a given pixel. Should these gradients surpass a threshold, this constitutes as an edge. This definition is what is used in Guttmann *et al.*’s work. However, as evidenced by [43] and [44], these do not adequately capture the edges of objects, and in addition, this definition is only defined for monochromatic (or grayscale) images. As such, for any colour images, these must be converted to a monochromatic version first before proceeding. By doing this, the correlation between the other colour channels is essentially ignored. When colours are produced in colour images, sensors to capture the scene that are geared towards dominant wavelengths may have contributed to other sensors that are geared for other wavelengths. Considering the colour image as separate planes loses this correlation, which is the reason why some of the edges that should appear in the final output. As such, the edge definitions that are used in this framework originate from *vector-order statistics*, where each colour pixel is treated as a three-dimensional vector. Neighbourhoods of pixels are gathered as a collection of vectors, and statistical calculations are employed to determine the gradient. In this approach, information from all colour channels are employed, and so the correlation between them are exploited.

With respect to Guttmann *et al.*, the horizontal and vertical gradient definition will inevitably fail, especially in areas of weak contrast. Should these areas surface, depth values of certain areas will inevitably leak into areas where the depth is completely different. This issue is resolved in the proposed framework, where the hard segmentation results of Graph Cuts are merged with the smoothing properties of Random Walks, ultimately mitigating any possibilities of depths leaking into other areas.

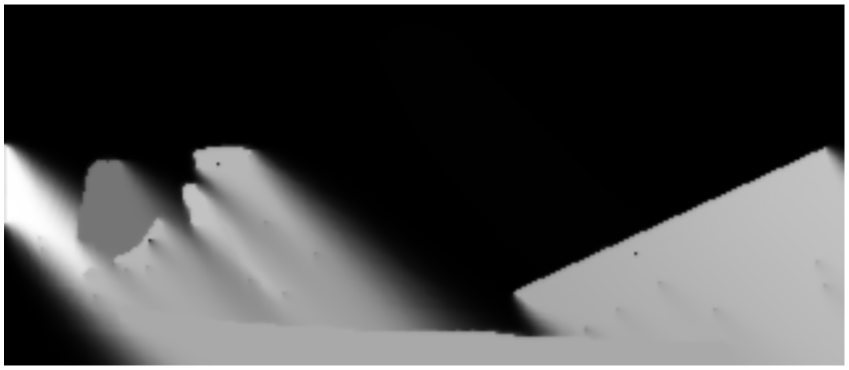
As an illustration of these errors, Fig. 1.4(a) and (b) shows an example of a video sequence where the first frames and last frames are shown, with their corresponding user-defined scribbles. Finally, Fig. 1.4(c) shows one of the depth maps that resulted from this sequence. With reference to Fig. 1.4, due to the improper definition of edges, though there are areas that clearly have a strong colour difference between the two, the discrete gradient does not properly discriminate between them, especially on the right of the frame. There is a wooden structure that has a different colour distribution than the background, but due to the edge definition, depths “leak” into other well defined areas. This does not occur in the proposed framework, as Graph Cuts ensures that edges are well-respected.



First Frame



Last Frame



Sample Depth Map

Figure 1.4: Illustration of errors by Guttmann *et al.*'s framework - Improper Edge Definition

1.7.3 Non-Temporally Consistent Optical Flow

The temporal terms in the framework rely on the use of optical flow. The temporal terms serve as a way of scaling the depth values, should the camera begin to zoom in or out. Specifically, the temporal terms are weighted so that areas that exhibit a high amount of motion have a higher weight in comparison to those that have little to no motion, and are assigned a lower weight. This should theoretically aid in dynamically adjusting the depth values for the final result, should the sequence exhibit motion perpendicular to the camera axis. Unfortunately, optical flow algorithms can be prone to errors, especially since most traditional algorithms only compute the flow between two consecutive frames. If the optical flow algorithm were to be temporally consistent, then this would ensure a smooth flow field throughout the entire sequence, due to the fact that information from neighbouring frames is used. For a video sequence to be *temporally consistent*, information from the other frames in the sequence should be incorporated, or employed in the current frame. This information can help resolve ambiguities, such as occlusions, disocclusions, or noisy areas in certain frames. As such, this problem can be solved by dynamically adjusting the labels through using a computer vision based object tracker that is designed for motion perpendicular to the camera axis, or by use of a temporally consistent optical flow algorithm. Results justifying that temporally consistent optical flow is the correct approach in optical flow estimation was demonstrated in Chapter 4.

1.7.4 Memory Requirements and First & Last Frame Labeling

To ultimately solve for the depths in this framework, a large sparse linear system of equations is solved. As the number of frames increases, so does the amount of memory required for solving the depths. In fact, as the number of frames grows, the computational time will grow, in addition to training the SVM classifiers, and detecting SIFT points. Guttmann *et al.* attempt to combat this by subsampling each frame by a factor of 4, but the SVM training and SIFT point detection do not escape the computational burden. Indirectly related to memory requirements, we have demonstrated in Chapter 2 that labeling only the first and last frame are subject to a number of problems. Also, there could be complex motion that exists in between these frames, and without the help of the user, or an algorithm that will help delineate these depths, the results will inevitably be erroneous. Specifically, there could be objects that are introduced in between the frames that require labeling. When such a situation occurs, more than one frame will inevitably require labeling to ensure high accuracy. In addition, labeling more than one frame would be ideal, so that the memory requirements for solving the linear system are minimized - the number of unknown variables are inevitably decreased. As such, the purpose of our label propagation algorithms are to minimize the amount of user interaction required to label all of the frames, as it can be quite tedious for the user, and simultaneously label all of the frames in the video sequence to reduce memory requirements. To further reduce memory requirements, we introduce a block-processing scheme, discussed in Chapter 2.6.2, that further provides a valid compromise between fine details in accuracy and memory consumption.

1.8 Thesis and Scientific Contributions

Though Guttmann *et al.*'s method attempts to bridge the gap between direct automatic computation of depth maps, and complete manual methods as done through rotoscopers, the issue is that the framework is quite computationally intensive. As will be discussed later, some portions of the framework are susceptible in giving false information, as well as requiring large amounts of memory. Using this method as a reference, and by examining work performed in semi-automatic segmentation, upon closer inspection, we can consider semi-automatic 2D to 3D conversion as a *multi-label image segmentation* problem. Each depth can be considered as a unique label, and to create a depth map, each pixel is classified to be long to one of these unique depths.

Inspired by Guttmann *et al.*, this thesis proposes a stereoscopic processing and conversion system that considers solving for depth maps, and ultimately performing 2D to 3D conversion, in this very same approach. However, the novelty in this approach, and thus the core of the system, is to combine the merits of two semi-automatic segmentation algorithms to produce high quality depth maps: Random Walks [45] and Graph Cuts [46]. In the segmentation viewpoint, automatic methods are not an option, as we wish to leverage the user in creating accurate results, and would also be contrary to our viewpoint on automatic 2D to 3D conversion. There are many semi-automatic algorithms in practice. Examples are Intelligent Scissors [47], or IT-SNAPS [48], where the user clicks on pixels, known as anchor points, that surround an object of interest to extract. In between the anchor points, the smallest cost path is determined that can delineate the foreground object from the background. However, this can only be performed on objects that have a clear separation from the background. In addition, this method could not be used in specifying depths on background areas, as the core of the algorithm relies on separation between the background and foreground by strong edges. Another example is the Simple Interactive Object Extraction (SIOX) framework [49], which operates in the same way as Graph Cuts and Random Walks with the use of foreground and background strokes, but unfortunately only works on single images. We ultimately chose Random Walks and Graph Cuts, not only because marking regions of similar depth is more intuitive, but these can ultimately be extended to video sequences.

However, for Random Walks and Graph Cuts, these were originally tailored for *binary* segmentation - decomposing an image into foreground and background regions. In order to adapt these for the multi-label case, each methodology must be modified for the purposes of creating depth maps. When determining depth maps for single images, the procedure to do so is very much the same as performing semi-automatic image segmentation, much like [45] and [46]. However, the case of video becomes complicated, as ambiguities in designing the system surface, such as what keyframes to mark, memory tradeoffs and computational complexity. With reference on determining what keyframes to mark, it is natural to assume that marking several keyframes over a video sequence would be time consuming.

In this aspect, not only do we allow the user to manually choose which keyframes to mark, we also investigate two novel methods for *automatically* propagating the user-defined strokes throughout the sequence. The first method is through exploiting the benefits of a well-established object tracking algorithm known as the Tracking-Learning-Detection (TLD) framework, and to tailor this towards propagating depths in a video sequence, while the second method is to use an optical flow algorithm that does not use global optimization, and create a edge-aware temporally consistent version of this. Without global optimization, the paradigm can be easily implemented on parallel architectures, which is attractive to produce results in real-time. However, each method has their merits and drawbacks, but they are to be used depending on the type of scene at hand. Nevertheless, these ambiguities will be discussed later. With Guttmann *et al.*, there are several problems with this approach, justifying the need to reformulate the problem all together, thus leading to the purpose of this thesis. Guttmann *et al.* omit several implementation details discussed in [1], there were some assumptions, as well as some design decisions that needed to be made in order for the implementation to succeed. These details regarding how Guttmann *et al.*'s framework was implemented for this thesis can be found in Appendix 1.

With reference to Section 1.5, in this thesis, disparities are estimated directly, without estimating the actual depth of the scene first. As can be seen in Eq. 1.3, perceived depth is connected to disparity only through a limited amount of parameters. Working directly with disparities has the numerical advantage of dealing with values that are no larger in magnitude than a few tens of pixels. However, for depth, this can range anywhere between zero and infinity, and is inversely proportional to disparity, as can be seen in Eq. 1.1. This quantifies the final observed change in both images.

In summary, the scientific contributions made by this thesis is as follows:

- Re-formulate the semi-automatic 2D to 3D conversion framework into one that takes advantage of the multi-label image segmentation framework. In this viewpoint, the goal is to classify each pixel in the image or frame to belong to one depth from a set of user-defined depths. The way this is performed is to use Random Walks and Graph Cuts, and modify the frameworks in such a way that is natural to suit this conversion. After, the merits of both are merged in a coherent way to combine the advantages of each method to produce the best result possible. This essentially generates dense depth maps that is representative of the scene, using the user as a powerful tool to guide the accuracy in generating the result.

- An extension of this algorithm is made to video sequences. Essentially, videos are just a sequence of images at a given frame rate. As such, a sequence of depth maps, one per frame in the video sequence, is generated by performing a modified version of a volumetric multi-label segmentation. Random Walks and Graph Cuts are performed on the video sequence, with their results merged like for single images. However, there are issues that surface for the case of video, such as what frames to mark in the video sequence, and memory constraints. These will be dealt with during the exposition of the proposed system.

- Referencing the previous point, more than one frame within the video sequence needs to be marked by the user so that the depths are estimated consistently throughout the sequence. Ideally, as many frames should be marked as possible to increase accuracy, but would be a significant undertaking on the user. As such, this thesis has the option of marking only the first frame, and these labels are propagated through the rest of the sequence, providing "user"-defined labels, but were labeled automatically by a label propagation algorithm.

- There are two methods to performing this label propagation algorithm. The first is through modifying a computer vision object tracking algorithm, which has the ability of dynamically adjusting the depths of regions as they come closer or farther from the camera. This method, however, can be computationally intensive (which will be shown later), which inspired the second method for this kind of propagation. This is by creating an edge-aware temporally consistent optical flow algorithm, should the scene exhibit only horizontal motion and can avoid some of the computational burden in comparison the first method, without the use of global optimization. The second method can become very attractive for implementation in dedicated parallel architectures, such as GPUs or FPGAs. Though the merging of Graph Cuts and Random Walks is novel, the most major contribution that this thesis is proposing is the two label propagation algorithms. As of the writing of this thesis, there has been no framework to automatically propagate depths throughout a video sequence, and in addition, the automatic adjustment of depths when required.

1.9 Outline

After covering the necessary preliminaries and recent state-of-the-art, the outline of this thesis is as follows: Chapter 2 discusses a general overview of how the proposed framework functions, including the semi-automatic segmentation algorithms used, the design choices that were made, and how the conversion framework functions for both image and video sequences. Chapters 3 and 4 discusses two methods to propagate user-defined strokes over the video sequence, as we mentioned previously, which are the computer vision object tracking based approach, and the edge-aware temporally consistent optical flow approach respectively. Because these methods are quite involved, these merit separate chapters to contain them. Results illustrating that these proposed methods function will be shown in their respective chapters. These include evidence on the proposed tracking algorithms, ensuring that they are functioning properly. As a compliment to the reader, as this thesis compares with Guttmann *et al.* - the most established work on 2D to 3D conversion, Appendix 1 gives implementation details for Guttmann *et al.*'s method, should the reader attempt to recreate the system themselves. Chapter 5 will show various results, including various depth map results, and the corresponding converted results illustrated through anaglyph images. These will be illustrated for both images and video sequences. This chapter also compares the results generated from the proposed framework with Guttmann *et al.*, as it is currently the most well-known accepted method for semi-automatic 2D to 3D conversion. As 3D content is observed by human beings, there is a need to verify that our results are pleasing to human perception, as well as tailoring towards their comfort. As such, results by user poll will be illustrated here, showing user opinions for the proposed work, as well as Guttmann *et al.*'s work. Finally, Chapter 6 presents a summary of the thesis, conclusions and future work in the area.

Chapter 2

Conversion Framework

As with Guttmann *et al.*, the proposed conversion framework in this thesis relies on the user providing the system with an initial estimate of the depth, where the user marks objects and regions as closer or farther from the camera. In our framework, not only does the user have the option of marking depths with monochromatic intensities, the user has the option of marking with different colours, so long as the colours vary from dark to bright. Specifically, brighter and darker colours perform the same task as marking with brighter and darker intensities. A flow chart that explains our framework can be seen in Fig. 2.1 shown below. Before we present our system, we provide an explanation of how the source media, whether it be an image, or a video sequence, is represented in our framework, and how the labeling of the user aids in producing good quality synthetic depth maps for the 2D to 3D conversion process.

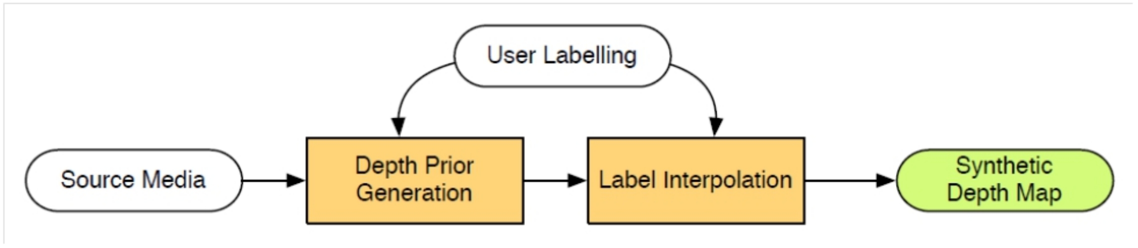


Figure 2.1: A flowchart representing the general conversion framework

A more detailed flow chart that explains our conversion framework is shown below in Fig. 2.2. Each stage of this framework will be explained in further detail as we proceed.

2.1 Graph Representation

To ensure the highest possible level of flexibility, the conversion framework treats both images and video the same. Images and video are represented in an N-connected graph. A graph in this viewpoint can

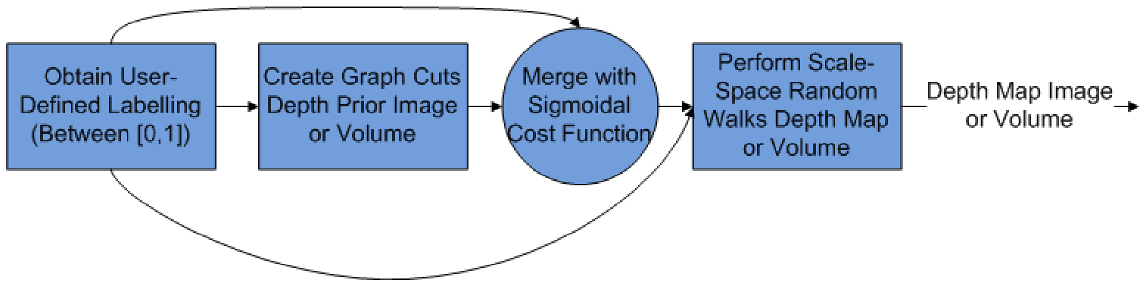

Obtain User-Defined Labelling (Between [0,1])
Create Graph Cuts Depth Prior Image or Volume
Merge with Sigmoidal Cost Function
Perform Scale-Space Random Walks Depth Map or Volume
Depth Map Image or Volume

Figure 2.2: A more detailed flowchart representing the general conversion framework

be considered as a lattice of nodes, where each pixel is one node. The pixels, or nodes, have edges connecting them together. The edges are assigned numerical values, representing how dissimilar the connected pixels are from each other. What measures the dissimilarity can be a variety of different ways, ranging from colour similarity, to textural differences. An N-connected graph means that there are N edges that come out of each node to connect to other nodes. Fig. 2.3 is an example of a 4 x 4 image labeled with three depths with different colours.

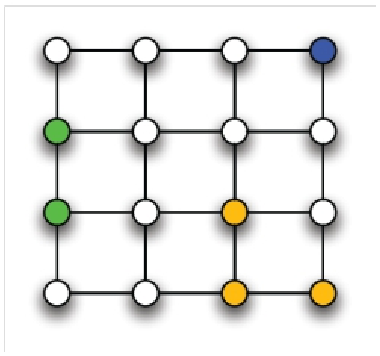


Figure 2.3: An example of a labeled graph. The shaded nodes represent pixels that have been labeled by the user

Here, each node is a pixel that connects to its vertical and horizontal neighbours, and it is thus a 4-connected graph. 4-connected graphs are not the only way to connect nodes together. 8-connected graphs are also possible, where each pixel would be connected by its vertical, horizontal, and diagonal neighbours. In the case of video, it is a three-dimensional graph, where each pixel in one frame not only connects with nodes within the same frame in a spatial manner, but also in a temporal manner. Such a possibility is for a 6-connected graph, where a frame is connected in a 4-way fashion as mentioned earlier, in addition to the node in the same spatial location in the previous frame, as well as the node in the next frame. Higher amounts of connectivity are possible, such as 26-connectedness, where the current frame is connected in an 8-connected fashion, as well as connecting to 9 pixels within a 3×3 neighbourhood in the previous frame, with the centre of the neighbourhood at the same spatial location as the current frame, as well as the 9 pixels in the next frame, maintaining the same spatial location there as well.

However, for the sake of memory constraints, 4-connectedness for images, and 6-connectedness for video are the schemes employed.

The aforementioned graph representation has been successfully used in the field of image segmentation, where the goal is to decompose, or split up, an image into coherent regions. Each region can either belong to a particular object, or have properties where the pixels within the region share a common feature. One of the most notable algorithms that abstract the image into graphs to perform image segmentation is with the Graph Cuts algorithm [46][50]. Along with Graph Cuts, Random Walks is another popular graph-based segmentation framework [45]. As mentioned previously, this thesis proposes a novel combination of Graph Cuts and Random Walks to be employed in finding an optimal labeling of the depths for the source images or videos, given an initial set of depth labels / strokes. To do this combination, Graph Cuts is used to determine an initial depth map, known as a *depth prior*. This depth prior serves as an additional channel of information into the Random Walks algorithm, and is weighted accordingly to control the contribution this information has to the final output. The reason why these two methods are combined will become evident much later, but first we discuss how the depth maps are generated for each framework separately, and the method to combine the information will follow later.

2.2 Random Walks for Images

Random Walks [45] is an optimization scheme that finds the likelihood of a random walker, starting at some label, visiting all of the unlabelled nodes in the graph. As a means of illustration, Fig. 2.4 illustrates a block diagram of the entire process when generating the depth map using Random Walks. It should be noted that the block diagram applies to both images and video sequences, and we will refer back to it when necessary.

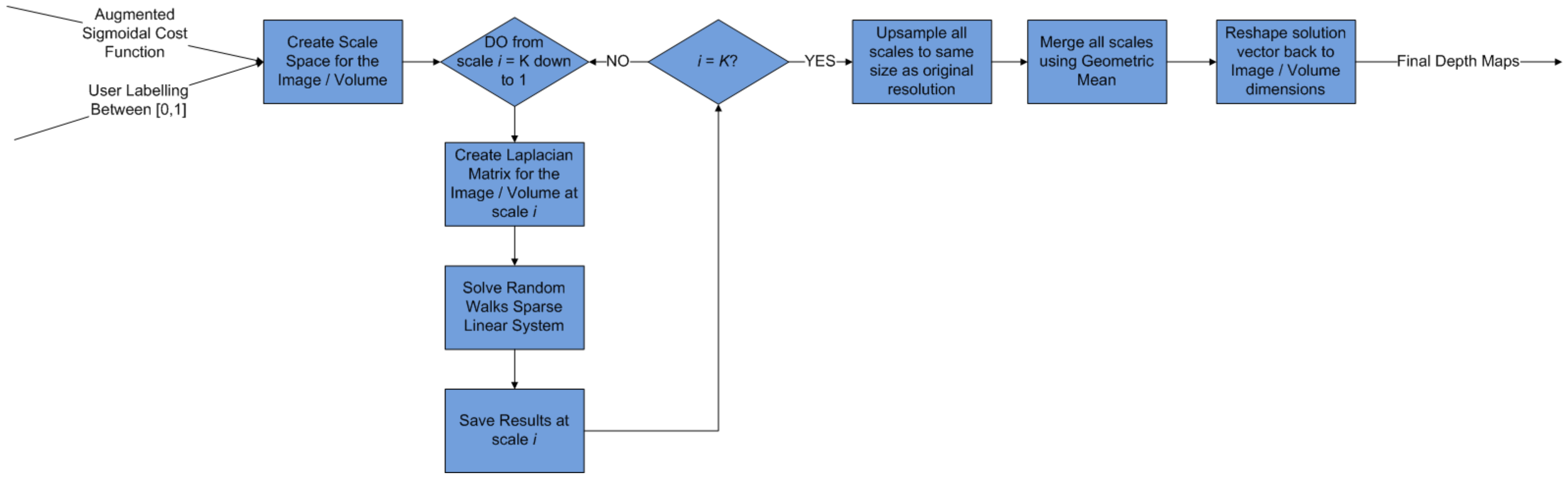

Augmented Sigmoidal Cost Function
User Labelling Between [0,1]
Create Scale Space for the Image / Volume
DO from scale *i* = K down to 1
Create Laplacian Matrix for the Image / Volume at scale *i*
Solve Random Walks Sparse Linear System
Save Results at scale *i*
i = *K*?
NO
YES
Upsample all scales to same size as original resolution
Merge all scales using Geometric Mean
Reshape solution vector back to Image / Volume dimensions
Final Depth Maps

Figure 2.4: A block diagram illustrating the steps taken to generate a depth map using Random Walks

The walker is biased by the edge weights so that it is more likely to visit similar nodes than dissimilar ones. This can be interpreted as a number of different processes, such as the steady-state solution to a diffusion problem or the node voltages in an electric circuit [45]. With reference to finding the probability of a random walker, with respect to image segmentation, the goal is to classify every pixel in an image to belong to one of K possible labels. Random Walks determines the probability of each pixel belonging to one of these labels, and the label that the pixel gets classified as, is the one with the highest probability. This is performed by solving a linear system of equations, that relies on using the Laplacian Matrix as the main tool for performing this labeling. If v_i represents the i^{th} pixel in the image, that is of size $R \times C$, then the Laplacian matrix, $\mathcal{L}$, of size $RC \times RC$ is defined in Eq. 2.1 as the following:

$$\mathcal{L} = \forall(i,j) \in \mathcal{P}, s.t. \begin{cases} \deg(v_i) & \text{if } i = j \\ -1 & \text{if } i \neq j \text{ and } v_i \text{ is connected to } v_j \\ 0 & \text{otherwise} \end{cases} . \tag{2.1}$$

(i,j) are pixels in the image $\mathcal{P}$. Coincidentally, i references the rows of $\mathcal{L}$, while j references the columns of $\mathcal{L}$. $\deg(v_i)$ is the degree of the pixel i, which is the sum of all of the edges leaving i. Each row of $\mathcal{L}$ is an indication of how each pixel i is connected to each other. Additionally, let us define the vector $\vec{x}$, of size $RC \times 1$, where the i^{th} row is the probability that the pixel i will be assigned a particular label, k. By letting $Q(v_i)$ represent the user-defined label of the i^{th} pixel, for a particular user-defined label k, vector $\vec{x}$ is defined as:

$$\vec{x} = \begin{bmatrix} x_1 & x_2 & x_3 & \cdots & x_{RC} \end{bmatrix}^T, \text{ s.t. } x_i = 1, \text{ if } Q(v_i) = k, \text{ and } x_i = 0 \text{ if } Q(v_i) \neq k, \forall i = 1, 2, \ldots, RC. \tag{2.2}$$

It should be noted that the above equation is only valid for user-defined pixels. As such, for those pixels not user-defined, or those that have no $Q(v_i)$ or were not assigned a $Q(v_i)$, those are the unknown probabilities that need to be solved, and are left as unknown. If we re-arrange the vector in Eq. 2.2 into two sets, such that those pixels that were marked by the user appear first, x_M, followed by those that were unknown after, x_U, we obtain the following decomposition:

$$\vec{x} = \begin{bmatrix} \vec{x}_M & \vec{x}_U \end{bmatrix}^T . \tag{2.3}$$

By keeping track of how we rearranged the vector in Eq. 2.3, by performing the same rearranging with the rows of the Laplacian Matrix, where each row represents the connectivity relationship for a particular pixel, we thus obtain the following decomposition for $\mathcal{L}$:

$$\mathcal{L} = \begin{bmatrix} \mathcal{L}_M & B^T \\ B & \mathcal{L}_U \end{bmatrix} \tag{2.4}$$

Finally, to solve for the unknown probabilities for the label k, we solve the following linear system

[45]:

$$L_U \vec{x}_U = -B^T \vec{x}_M \quad . \tag{2.5}$$

Therefore, using Eq. 2.5, we construct $\vec{x}$ for each label k, and solve for the unknown probabilities for the label k. Whatever label gives the maximum probability for a pixel in x_U, that is the label it is assigned. A number of different methods can be used to solve this linear system. LU Decomposition, the Conjugate Gradient technique [51], or Successive Overrelxation [52] are such examples. The Laplacian Matrix is known to be positive-definite, and so the Conjugate Gradient can be used here [51]. In our framework, we opt to use the Conjugate Gradient technique, due to the high sparsity of the Laplacian matrix.

However, with Random Walks, the original framework only performs *binary* segmentation, where pixels belong to either foreground or background. As mentioned previously, depth map generation can be considered as a case of multi-label image segmentation. Therefore, with Random Walks, for use in generating depth maps, the proposed framework modifies the methodology in the following fashion to achieve a multi-label image segmentation framework. The probabilities within Random Walks spans between the real set of $[0, 1]$. Therefore, the user-defined depths, and ultimately the solved depths for the rest of the image or frame, are allowed to be from this set as well. This allows for "manually" setting the probabilities of the marked pixels within the vector $\vec{x}$ as seen in Eq. 2.2. The ultimate goal here is to solve for only one label, where the label is the *depth* of the image or frame. As such, the user chooses values from the set of $[0, 1]$ to brush over the image or frame, where 0 represents a dark colour or intensity, and 1 represents a light colour or intensity. After, Eq. 2.4 can be used to solve for the rest of the depths. The resulting probabilities when solving Eq. 2.4 can directly be used as the depths for generating stereoscopic 3D content. As a means of increasing accuracy, and to combat noise, we employ Scale-Space Random Walks (SSRW) [53], which samples the image using a multi-resolution pyramid. Random Walks is applied to each scale within the pyramid, upsampled, and are all merged using the geometric mean. For the edge weights, the dissimilarity function we use is one that is frequently used in pattern classification research: the Sigmoidal function. Between two pixels, v_i and v_j, it is defined in Eq. 2.6 as:

$$N(v_i, v_j) = \gamma \left(\frac{2}{1 + \exp\left(\beta D(\vec{c}_i, \vec{c}_j)^{\alpha}\right)} \right) \quad . \tag{2.6}$$

$D(\vec{c}_i, \vec{c}_j)$ is the Euclidean distance, or L_2 norm, of the CIE L*a*b* components between v_i and v_j, and α, β, γ are parameters controlling how dissimilar two colours are. CIE L*a*b* is a colour space that is perceptually uniform, so that numerical colour distances in this space actually agree with perceived colour distances seen by humans [54], as opposed to the more common RGB colour space. This colour space is comprised of one achromatic, or lightness, component, and two chromatic, or colour, components: Lightness, L^*, redness-greenness, a^*, and yellowness-blueness, b^*. These components are calculated as follows:

$$L^* = 116 f\left(\frac{Y}{Y_o}\right) - 16 \tag{2.7}$$

$$a^* = 500\left[f\left(\frac{X}{X_o}\right) - f\left(\frac{Y}{Y_o}\right)\right] b^* = 200\left[f\left(\frac{Y}{Y_o}\right) - f\left(\frac{Z}{Z_o}\right)\right] \tag{2.8}$$

,

where

$$f(x) = \begin{cases} x^{\frac{1}{3}} & \text{if } x > 0.008856 \\ 7.7787x + \frac{16}{116} & \text{otherwise} \end{cases} \tag{2.9}$$

.

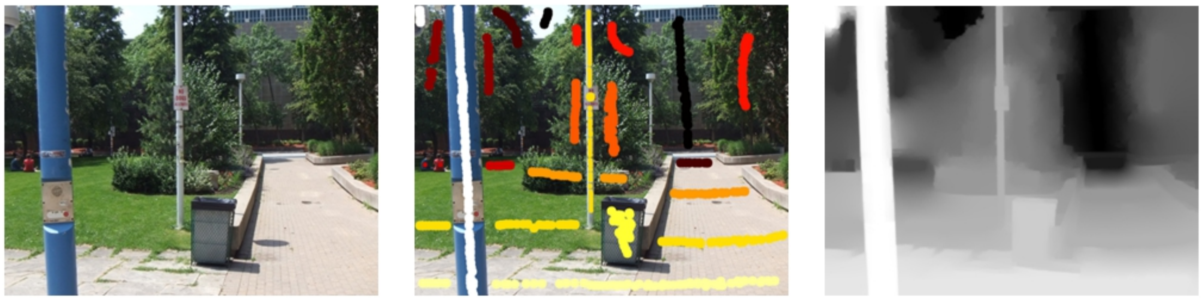
X, Y and Z are the CIE tristimulus values, and are calculated using RGB triplets, defined as:

$$X = 0.490R + 0.310G + 0.200B \tag{2.10}$$
$$Y = 0.177R + 0.812G + 0.011B \tag{2.11}$$
$$Z = 0.000R + 0.010G + 0.990B \tag{2.12}$$
$$\tag{2.13}$$

.

Here, R, G and B represent the red, green and blue components of a pixel in the image. X_o, Y_o and Z_o represent the *reference white* tristimulus values selected. These values define the colour 'white" in image capture or reproduction. Depending on the colour application, different reference white tristimulus values are needed to give acceptable results. Usually, the D65 reference white is chosen as the standard, which corresponds roughly to a mid-day sun in Western or Northern Europe, and was the colour space used in this thesis. This reference is also called a 'daylight" illuminant. The computational overhead for converting between RGB and CIE L*a*b* is very minimal, as it only requires a few multiplications and additions combined, and so this should not detract in performance for the proposed framework. For this thesis, and the results illustrated, the sigmoidal parameters were experimentally found, and were set to $\alpha = \gamma = 1$, and $\beta = 2$.

Unfortunately, with the use of Random Walks comes certain issues. A consequence with specifying depths in a continuous range through the pixel probabilities is that it is possible that depths will be produced that were not originally specified. This is certainly the desired effect, as this will allow for internal depth variation for objects, and that it would eliminate objects being perceived as *"cardboard cutouts"*—. What is meant by "cardboard cutouts", is the entire object being perceived as only one depth, where different regions of the object should appear at different depths, depending on which parts are closer or farther from the camera. However, in terms of weak contrast, this effect results in a consequence of “bleeding”, where regions at a closer depth can “bleed”, or merge, into regions of farther depth, even if these regions have a well defined border. An example of this is shown in Fig. 2.5, which is a snapshot of an area on the Ryerson University campus. This example was generating using the proposed framework for images. The image on the left shows the original image, the one in the middle shows user-defined scribbles, and the image on the right is the depth map produced by SSRW. As seen in the figure, the user only needs to mark a subset of the image, and a reasonable depth map is produced as a result. Though there is good depth variation on the objects and the background, there is evidence of bleeding around well-defined object boundaries. This is unlike Graph Cuts, and as we will see later, each pixel was labeled using only the labels provided by the user. In effect, Graph Cuts provides a hard segmentation, where the depth map that is generated only consists of the labels provided by the user. How to generate a depth map from the Graph Cuts point of view will be discussed next.



(a) (b) (c)

Figure 2.5: (a) Example image, (b) Example image with labels superimposed, (c) Depth map using SSRW. Depth Label Legend: Darker colours (red) - Far points, Brighter colours (orange and yellow) - Closer points, White pixels - Very close points

2.3 Graph Cuts for Images

Graph Cuts is based on solving the Maximum-A-Posteriori Markov Random Field (MAP-MRF) labeling problem with user-defined or hard constraints [50]. As a means of illustration, Fig. 2.6 illustrates a block diagram of the entire process when generating the depth map using Random Walks. It should be noted that the block diagram applies to both images and video sequences, and we will refer back to it when necessary.

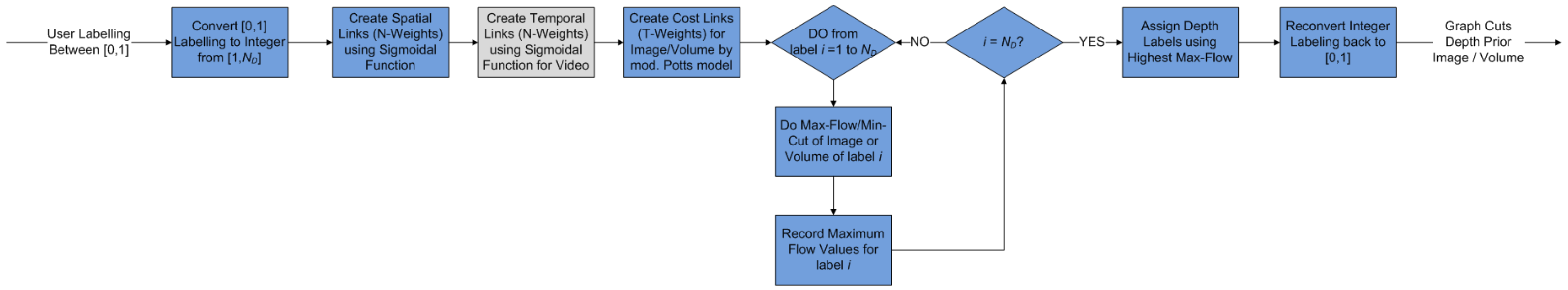

User Labelling Between [0,1]
Convert [0,1] Labelling to Integer from [1,N_D]
Create Spatial Links (N-Weights) using Sigmoidal Function
Create Temporal Links (N-Weights) using Sigmoidal Function for Video
Create Cost Links (T-Weights) for Image/Volume by mod. Potts model
DO from label i =1 to N_D
Do Max-Flow/Min-Cut of Image or Volume of label i
Record Maximum Flow Values for label i
NO
i = N_D?
YES
Assign Depth Labels using Highest Max-Flow
Reconvert Integer Labeling back to [0,1]
Graph Cuts Depth Prior Image / Volume

Figure 2.6: A block diagram illustrating the steps taken to generate a depth map using Graph Cuts

The solution to the MAP-MRF labeling problem is to find the most likely labeling for all pixels from the provided hard constraints. It has been shown in [50] that the solution can be found by minimizing the following energy function:

$$E(\mathcal{P}) = \sum_{p \in \mathcal{P}} D_p(f_p) + \sum_{\{p,q\} \in \mathcal{N}} V_{p,q}(f_p, f_q) \quad . \tag{2.14}$$

Here, $\mathcal{P}$ is the set of pixels that make up an image, and p is a pixel within the image. $E(\mathcal{P})$ is the "energy" of the image. $D_p(f_p)$ represents the data cost, or the cost incurred when a pixel p is assigned to the label f_p. $V_{p,q}(f_p, f_q)$ is the smoothness cost, or the cost incurred when two different labels are assigned to two different pixels within a spatial neighbourhood $\mathcal{N}$. In this framework, $\mathcal{N}$ is a 4-connected graph, as defined earlier, and we consider an image to be an N-connected graph. Alluding to graph theory, it has been shown that the solution that minimizes $E(\mathcal{P})$ is the solution that finds the max-flow/min-cut of a graph [50]. In other words, whichever configuration of pixels provides the *least amount of energy* generated by $E(\mathcal{P})$ should be the configuration used. Efficient algorithms and software has been created to perform this minimization, and is available by referring to [50]. It is this software that is used in this thesis, in conjunction with the methods that are proposed for using Graph Cuts, to create depth maps under this framework.

With the observations made previously, depth map generation in a semi-automatic approach can be considered as a multi-label classification problem. However, the formulation above for Graph Cuts also focuses solely on the binary segmentation problem. Coincidentally for Graph Cuts, each pixel will thus have associated foreground and background costs. Therefore, this thesis modifies the Graph Cuts methodology, and adapts it for the multi-label case as follows. Each unique user-defined depth value is assigned an integer label from the set $B \in [1, N_D]$, where N_D represents the total number of unique depths present in the user-defined labeling. A binary segmentation is performed for each label separately for each label $b \in B$. The user-defined labels having the label of b are assigned as foreground, while the other user-defined labels serve as background. The rest of the pixels are those we wish to label. Graph Cuts is run for a total of N_D times, once for each label, and the maximum flow values for the graph are recorded for each label in B. If a pixel was only assigned to one label, then that is the label it is assigned. However, if a pixel was assigned multiple labels, we assign the label with the highest maximum flow, corresponding to the least amount of energy required to classify the pixel. In some cases, even this will not always result in every pixel being classified, but region filling methods can be used to correct this.

Examining this in a graph point of view, one may question the spatial connectivity of a point in the image, should it be assigned multiple labels. In other words, one may question whether or not the spatial relationships of pixels are ignored in the segmentation result when multiple labels are all good candidates to classify one pixel. However, utilizing the maximum flows for multi-label segmentation, as neighbouring pixels will most will most likely be similar, and will share the same label. As such, neighbouring pixels will also most likely share the same maximum flow required to classify the pixel, and so there should not be ambiguous regions where pixels in uniform areas with respect to a depth stroke get assigned different labels.

In the Graph Cuts framework of this thesis, we use the aforementioned Sigmoidal function (Eq. 2.6) to reflect the smoothness costs. For the data costs, we use a modified Potts model, rather than the negative log histogram probability as done in [46]. In [46], the Potts model is a binary weighting, where the smoothness cost between two labels f_p and f_q within the spatial neighbourhood $\mathcal{N}$ (or $V_{p,q}(f_p, f_q)$) as 0 when both of the labels are the same, and a high cost K when both labels are different. In other words, the Potts model tries to encourage smooth labellings throughout the image, and penalizes when this is not possible. As this is a multi-label segmentation framework, the Potts model must be modified to suit this kind of task. The reason why the Potts model was chosen was due to the nature of the colour space to represent images. The negative log histogram probability is computed in the RGB colour space, which is inherently discrete. Naturally, histograms decompose the observed data into discrete bins, and so the RGB colour space is well suited here. However, we wish to employ the CIE L*a*b* colour space, as it naturally captures the human perception of colours better than the RGB colour space. In addition, CIE L*a*b* has floating point values for each of their components, and so it is not intuitive to determine how many bins are required when creating a histogram. It is with this reason that we consider a different data cost all together, and we thus turn to the modified Potts model. Table 2.1 shows how the data costs are computed when the user-defined label $b \in B$ is the one chosen as foreground.

Type of Link	Weights		
	$m = b, b \in B$	$m \neq b, b \in B$	$m \neq b, b \notin B$
Source Link	K	0	K
Sink Link	0	K	K

Table 2.1: Data costs used in the Graph Cuts segmentation for a label b, when the chosen foreground label is m.

Let us denote this chosen foreground label during a particular Graph Cuts minimization as m. Also, with referencing the table, the high cost K is defined as the following, using the same convention as in Eq. 2.14 [46].

$$K = 1 + \max_{p \in \mathcal{P}} \sum_{q:\{v_p, v_q\}} N(v_p, v_q) \quad . \tag{2.15}$$

Eq. 2.15 states that for each pixel in the image, sum up all of the weights of those coming out from it, and to determine what the maximum value is over all pixels, incrementing this value by one. To summarize the above table, we iterate through each possible label and perform a Graph Cuts segmentation. For the user-defined label that is currently being processed in the algorithm, or m, should another user-defined label be of the same label, the foreground/background costs are assigned $K/0$, as we should penalize the algorithm for assigning these pixels to "background", when the user clearly wanted them to be the label m. If the pixel is a different user-defined label, then the foreground/background data costs should be assigned $0/K$, as we should penalize the algorithm if it should decide to assign this pixel the current user-defined label m, when the user clearly did not do so. All other pixels are "unknown", and those are the pixels we want to solve the depth for. The foreground and background data costs are both assigned costs of K, as this will ensure that the algorithm is making the right choice when classifying an unknown pixel. For each segmentation result of each label, we determine the maximum flows that result, and classify those pixels accordingly, as was mentioned previously.

Fig. 2.7 below illustrates a depth map example in the same style as Fig. 2.5, also using the proposed framework in the image viewpoint. It should be noted that the same labels in Fig. 2.5 are used in order to be consistent. As noted with Random Walks, only certain portions of the image are labeled, and a reasonably consistent depth map was still generated successfully. It should be noted that there are some portions of the image that failed to be assigned any labels, such as the area around the right of the garbage can. However, this can certainly be rectified by using region-filling methods, like Criminisi *et al.*'s method in [55]. Though the result generated a depth map that respects object boundaries very well, the internal depths of each of the objects, as well as some regions in the background do not have any internal depth variation, or any depth gradients. As such, if this were to be used in visualizing stereoscopically, the objects would appear as “cardboard cutouts”. Noting the merits of Random Walks, where the depth variation within objects is desired, and combining this with the hard segmentation results of Graph Cuts, Random Walks should allow for depth variation to make objects more realistic, while Graph Cuts can eliminate the bleeding effect and respect hard boundaries. The way we combine these two depth maps together will be described next.

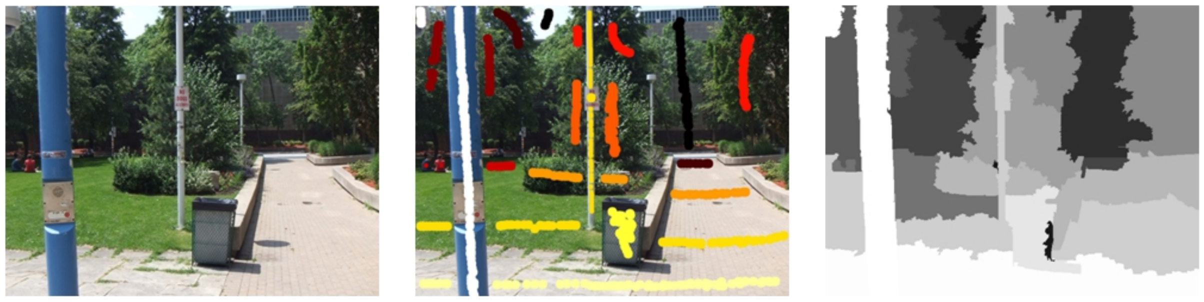

(a) (b) (c)

Figure 2.7: (a) Example image, (b) Example image with labels superimposed, (c) Depth map using Graph Cuts only. Depth Label Legend: Darker colours (red) - Far points, Brighter colours (orange and yellow) - Closer points, White pixels - Very close points

2.4 Merging the two together

To merge the two depth maps together, a depth prior is created, serving as an initial depth estimate, and provides a rough sketch of the depth. This information is fed directly into the Random Walks algorithm. The depth prior is essentially the Graph Cuts depth map, and should help in maintaining the strong boundaries in the Random Walks depth map. Before we can merge the two together, we must modify the depth prior. The depth map of Random Walks is in the continuous range of $[0, 1]$, while the depth map for Graph Cuts is in the integer range of $[1, N_D]$. Both depth maps correspond to each other, but one needs to be transformed into the other so that they both can be compatible. As such, when the Graph Cuts depth map is generated, it is then processed through a lookup table $T[k]$, where k in this case is an integer label in the Graph Cuts depth map. The goal of the lookup table is to transform the integer labeling into a compatible labeling for Random Walks. This can simply be done by first performing a labeling within the range of $[0, 1]$ for Random Walks, and sorting this list into a unique one in ascending order. Each depth in this list is then assigned an integer label from $[1, 2, \ldots, N_D]$, keeping track of which continuous label corresponds to each integer label. When Graph Cuts has completed, this correspondence can be used to map back to the continuous range.

To finally merge the two depth maps together, this depth prior is thus fed into the Random Walks algorithm, and this is done by modifying the edge weights directly. The depth prior information is incorporated by performing a Random Walks depth map generation, but the edge weight equation is modified (Eq. 2.6), where the distance function is appended to include information from the depth prior. If we let d_i represent the depth from the depth prior at pixel i, after being processed through the lookup table, the edge weight equation is modified by modifying the Euclidean distance, which is defined as:

$$D(\vec{c}_i, \vec{c}_j, d_i, d_j|\alpha) = \sqrt{[D(\vec{c}_i, \vec{c}_j)]^2 + \alpha(d_i - d_j)^2} \quad . \tag{2.16}$$

$\vec{c}_i, \vec{c}_j$ and $D(\vec{c}_i, \vec{c}_j)$ are defined in section 2.2. α is a positive real constant, which is set to 0.5 through repeated experimentation, and was shown to produce the best results. α serves as a scaling factor determining how much contribution the depth prior has to the output. d_p denotes the depth at pixel p from the depth prior. However, the dynamic range of the depth values from the depth prior is within $[0, 1]$, where as the components of the CIE L*a*b* colour space, as seen in Eq. 2.16, are significantly larger. As such, these terms will overpower the terms seen in the depth prior. With this, prior to depth map generation, the entire image / frame is converted into the CIE L*a*b* colour space first. The channels of every pixels are normalized individually, so that all components for each pixel is in the range of $[0, 1]$. These normalized components are the ones used in Eq. 2.16.

To briefly summarize the method, once each channel has been normalized in the CIE L*a*b* colour space [54], a depth prior is first generated. After, this information is fed into Random Walks to generate a final depth map, which is the one ultimately used for 2D to 3D conversion. Fig. 2.8 shows an example of the merged results, where we set $\alpha = 0.5$, using the proposed framework for images again. The left image shows the depth map generated by SSRW, the middle shows the depth map created by Graph Cuts (before region filling). Finally, the right shows the merged depth map by using Eq. 2.16. When compared to the left and middle images, the merged result seen on the right contains the most desirable aspects between the two. The depth prior has consistent and noticeable borders for the objects in the scene, while the Random Walks depth map alone contains subtle texture and gradients to those objects. In particular, the trees and shrubbery in the original image are now much more differentiated from the background and neighbouring objects than before. In addition, the areas that were left unclassified in the depth prior, or the "holes", have been filled in after the final depth map was produced. The reason is because the edge weights from Random Walks ultimately considered the depth prior as only one component of information out of the overall feature vector that is used, and can still rely on the colour image data to produce the final depth map.

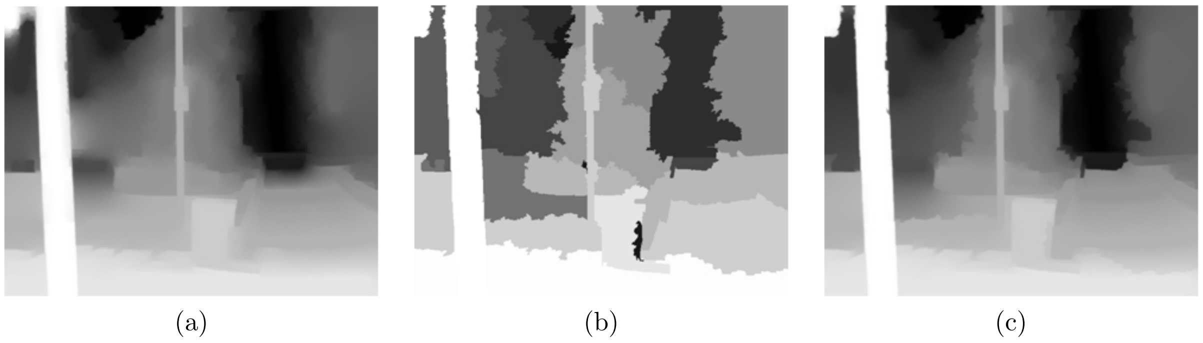


Figure 2.8: Depth map generation example by merging the two frameworks together (a) Depth Map via SSRW, (b) Depth Map via Graph Cuts, (c) Combined result using depth prior / Graph Cuts. $\alpha = 0.5$

2.5 Correcting the Depth Map

With the current framework, the result is heavily dependent on the labelling provided by the user. As such, should the user improperly label the scene, or omit labels, the user can simply append or remove labels and re-run the algorithm. This cycle can be repeated until the user is satisfied with the results. As an illustration, Fig. 2.9 illustrates the first attempt at creating a depth map. The image desired to be converted with its initial labellings is shown in Fig. 2.9(a). Fig. 2.9(b) shows what the depth map is using the combined framework discussed previously. As can be seen in the figure, the ground as it leads from the camera to the tower has not been captured properly. Specifically, there is a huge depth discontinuity, where the change should be gradual, following a sort of gradient change. Figs. 2.9(c) and (d) illustrate appending more labels on the ground between the initial ground labellings in white to the tower, with the resulting depth map. As can be seen, the result more accurately captures how the depths are perceived when observing the scene. Even though the result is heavily user-dependent, some simple corrections can be made to achieve the desired results.

2.6 Extension to Video

In the proposed framework, video data is considered to be a special case when looking at images. Essentially, a single image is a video sequence that consists of one frame. For all video sequences to be compatible with this framework, they are read in, and re-saved as an image sequence, with each frame re-encoded in RGB format. Several formats, such as MPEG, AVI and H.264 have different methods in storing the frame data, and in order to ensure compatibility over all video sequences, they are all converted into RGB format. Many open-source libraries, such as FFmpeg [56], have the capability of reading in video formats of all kinds and extracting the raw frame data, and re-saving as an image sequence. In terms of the computational overhead, this is again very minimal, as the libraries are optimized to operate on several thousands of frames, and take advantage of the instruction set available on many current and state-of-the-art processors. In this framework, FFmpeg is used for producing a raw sequence of frames in RGB format, for compatibility in the proposed thesis. In addition, for video data,

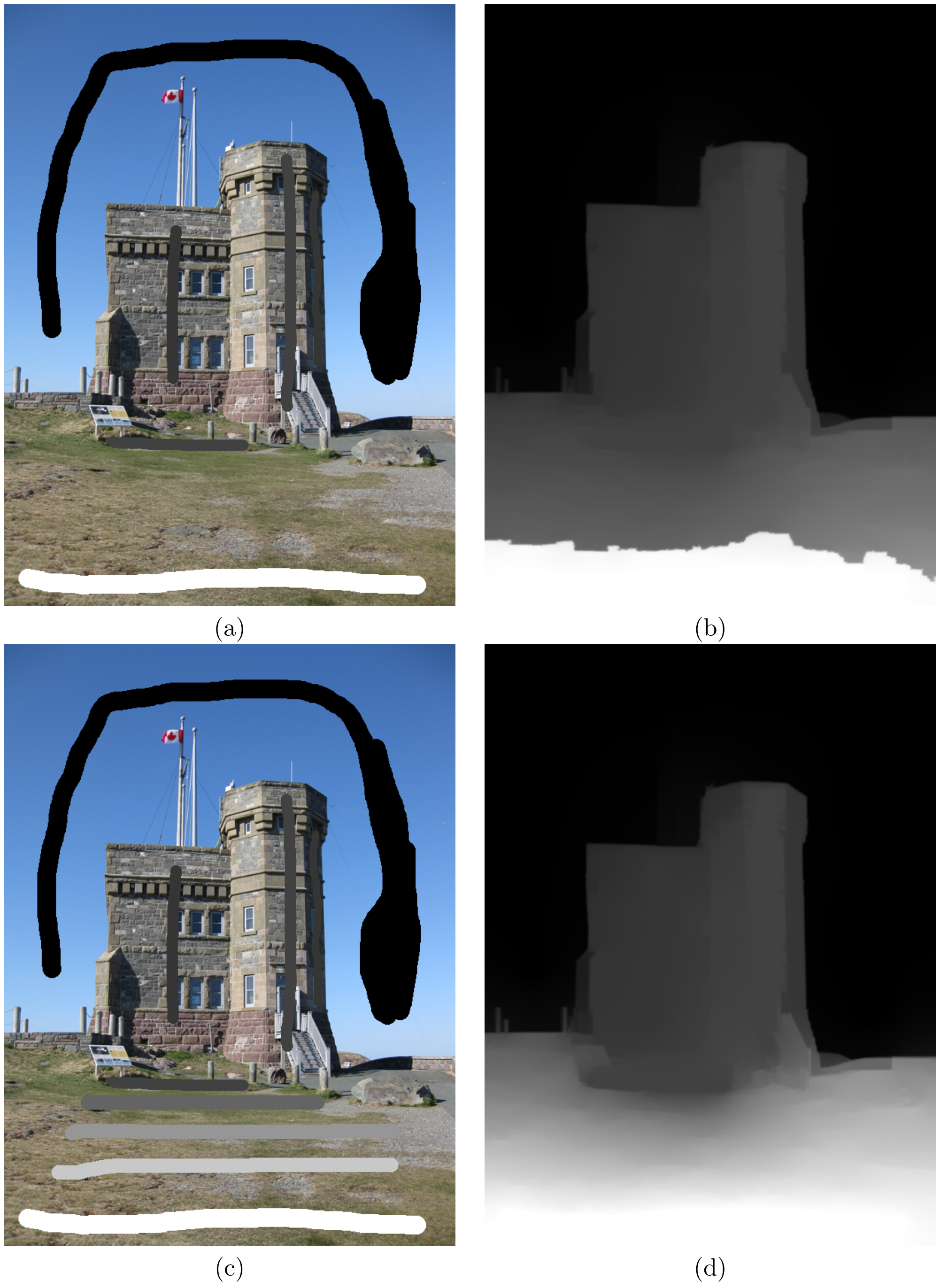


Figure 2.9: Depth map generation example with improper labellings and corrected labellings

it is essentially a sequence of images at a specified frame rate. When generating depth maps for the video case, our framework performs in the same fashion, except that there are two fundamental areas that need to be modified:

1. Instead of labeling only one frame, several keyframes need to be labeled. The number of keyframes that need to be labeled is an issue that we will discuss later. However, for the case of video, more than one frame must be labeled.

2. The graph representation needs to be modified from a two-dimensional one, to a three-dimensional one. Instead of having a 4-connected graph, we now have a 6-connected graph, that connects the frames together in a temporal fashion. These connections ensure that the depth map in one frame is consistent with the depth map in the subsequent frame.

Even with the above modifications, there are a number of caveats that one must look out for when dealing with video. Before we go through the exact mechanics of how video conversion is performed, and with these caveats, we are assuming that there are no abrupt changes in the video being processed. In other words, the video consists of a single camera shot. If a video has any camera changes, then it must be split up into smaller portions where each portion does not have a change of shot. This can be done either manually, or using an automated shot detection system. A skeleton block diagram of the conversion framework extended to video can be seen in Fig. 2.10. As previously, each component will be explained as we proceed. The label propagation algorithms, as previously mentioned, these alleviate user effort and increase the accuracy in the depth map estimation, and will be explained in the proceeding chapters. These merit separate chapters to explain how they function.

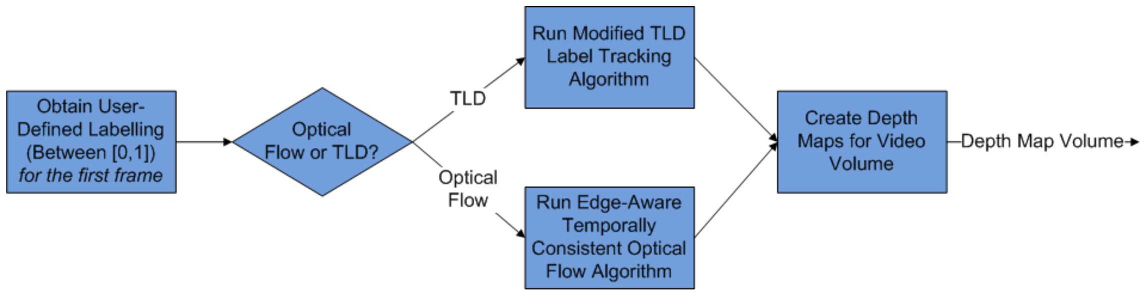

Obtain User-Defined Labelling (Between [0,1]) *for the first frame*
Optical Flow or TLD?
TLD
Optical Flow
Run Modified TLD Label Tracking Algorithm
Run Edge-Aware Temporally Consistent Optical Flow Algorithm
Create Depth Maps for Video Volume
Depth Map Volume

Figure 2.10: A flowchart illustrating the steps taken to generate depth maps for a video sequence

2.6.1 Keyframes to label

The first caveat is the problem of applying labels to keyframes. Specifically, one question that a user may ask is how many frames is required for labeling in order for the depth map to be the most accurate? For the video segmentation realm, at least the first and last frames need to be labeled, in order for the segmentation to be consistent across the sequence. This is the approach taken in Guttmann *et al.*'s work. However, while it is possible to label only certain keyframes, as we will demonstrate with their work later, this will result in a number of different depth artifacts. Fig. 2.11 illustrates a sample of frames, with their labels overlaid. This comes from the Sintel sequence[1], and this particular shot is 38

[1]`http://www.sintel.org`

frames. Fig. 2.11 is comprised of the 1st, 19th and 38th frames, with their labels superimposed. Fig. 2.12 illustrates some depth maps within the sequence using our approach to video (to be discussed later), showing what happens when only frames 1, 19 and 38 are labelled. Finally, Fig. 2.13 illustrates what happens when all frames are labeled in the same style as Fig. 2.12, again using our framework to video. In both Figs. 2.12 and 2.13, the first row illustrates frames 1 and 13, while the second row illustrates the frame 25 and 38. For the sake of brevity, we only show the labels for frames 1, 19 and 38. The colour scheme for the labels is the same as in Figs. 2.5, 2.7 and 2.8.

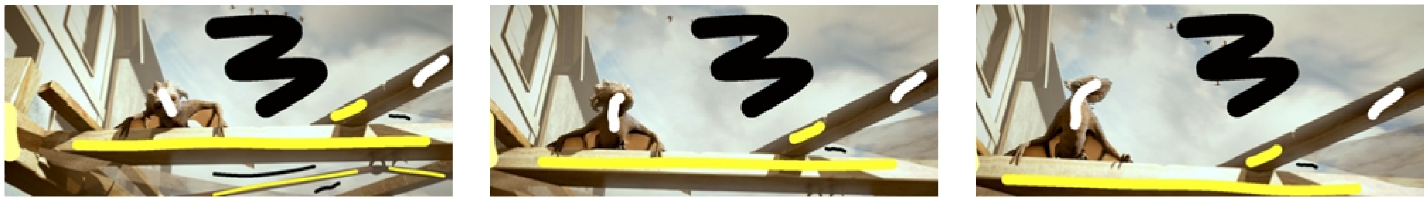

(a) (b) (c)

Figure 2.11: Labeling used for the Frames 1 (a), 19 (b) and 38 (c) frames in the Sintel sequence

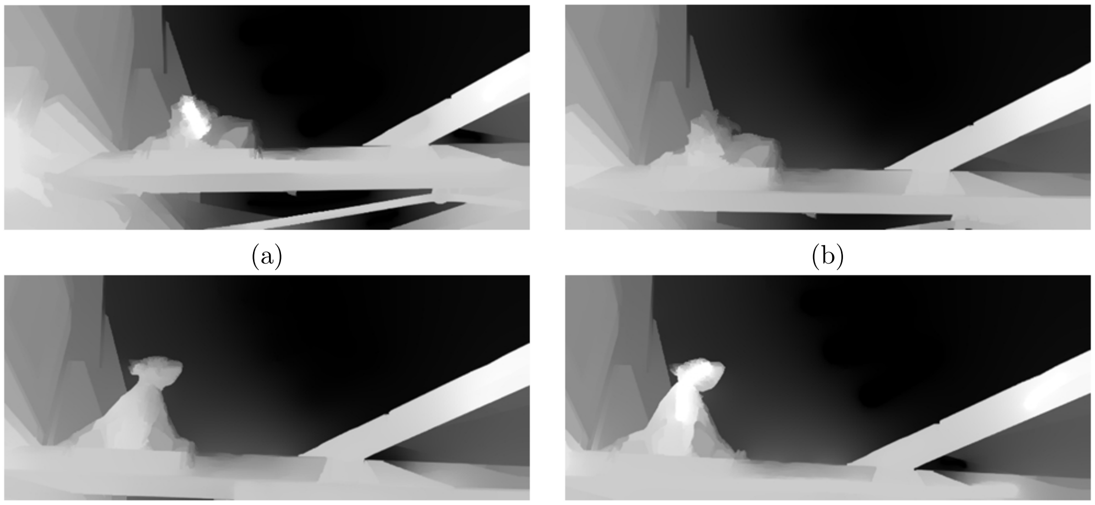

(a) (b)
(c) (d)

Figure 2.12: Depth maps for Sintel sequence with only three frames labeled. (a) Frame 1, (b) Frame 13, (c) Frame 25, (d) Frame 38

As can be seen in the figures, for the frames that had no labels, the rapidly moving parts of those frames quickly "fade" in depth or appear to move away from the camera. This is clearly not the case, since all objects in the scene move laterally, and not farther away. If all of the frames are labelled appropriately, then the depth remains consistent for all frames. While labeling every frame produces the best results, it is not the easiest task to perform. For a modest sequence, such as the Sintel sequence performed previously having 38 frames, labeling each frame manually is quite a tedious task, and would take a considerable amount of time. However, if the labeling can be performed in a more automated

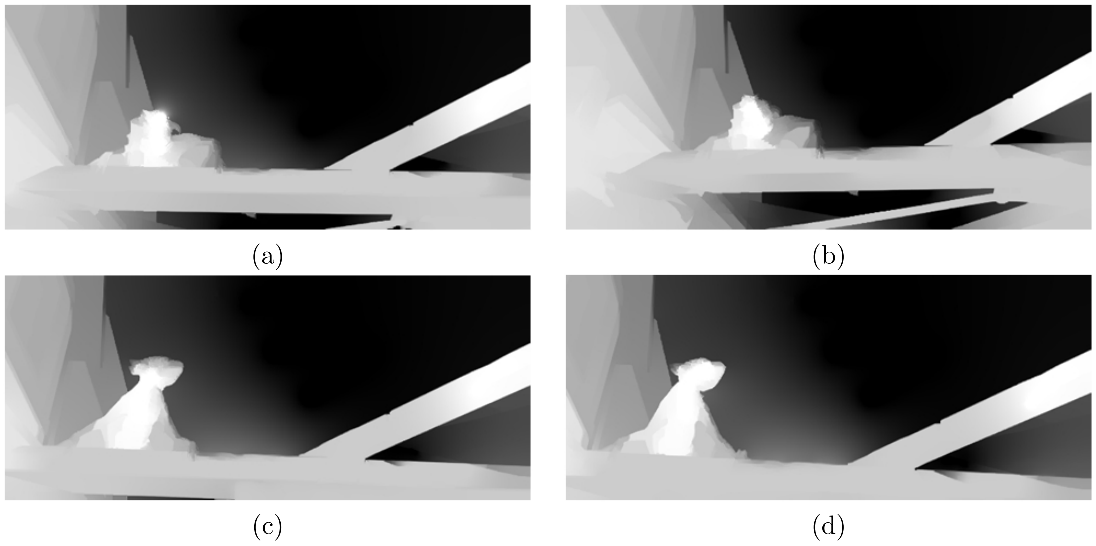


Figure 2.13: Depth maps for Sintel sequence with all frames labeled. (a) Frame 1, (b) Frame 13, (c) Frame 25, (d) Frame 38

fashion, this would certainly simplify the task of creating a more detailed labeling, and thus increase the accuracy of the depth maps. The way we perform this is through the use of a computer vision tracking algorithm for tracking objects in unconstrained video, since unconstrained is the type that our framework will most likely encounter. The idea is to only label the first frame with the user-defined depth labels. After, these labels are tracked throughout the entire video sequence. As can be seen in Figs. 2.5, 2.7 and 2.8, there will be areas in the video where the depth will change from what the user defined those areas to be. As such, there has to be a way of dynamically adjusting those depths when those situations arise. The way we perform this automatic labeling is through using one of two computer vision based algorithms: (a) An algorithm for tracking objects in unconstrained video, since unconstrained is the type that our framework will most likely encounter, and (b) An edge-aware temporally consistent optical flow based algorithm that is employed, should the scene exhibit only horizontal motion and can avoid some of the computational burden exhibited in (a). What is attractive about (b) is that it determines optical flow without the use of global optimization. This becomes very attractive for implementation in dedicated parallel architectures, such as GPUs or FPGAs. However, we hold on the explanation of these two algorithms for the next chapters (Chapters 3 and 4), as they are quite extensive, and merit a separate chapter.

To quickly summarize, in this framework, the user can perform keyframe labeling in two ways:

1. Manually selecting certain keyframes in the video sequence to label, and the depths are generated for the rest of those pixels (and frames) that were not labeled. As previously mentioned, only marking a certain subset of frames leads to depth artifacts.

2. Marking only the first frame, and allowing a computer vision tracking algorithm or an edge-aware temporally consistent optical flow algorithm to propagate those labels throughout the rest of the

frame. For the first algorithm, it is used when depth of certain points changes due to movement perpendicular to the camera axis (i.e. moving farther or away from the camera), whereas the second method is used only when the scene exhibits horizontal motion.

2.6.2 Memory Constraints - Video

The other main caveat with processing video is the amount of memory required. Regardless of the computer being used, it is very easy to quickly consume all of the available memory on the machine attempting to process the entire video at once. As the number of frames increases, so does the size of the graph representing the video.

There are a number of different approaches to processing video, each with their own benefits and trade-offs. The most intuitive option is to process each frame independently, essentially treating them as regular images. The benefit is that it requires little memory and the frames can be processed in parallel. Unfortunately, this breaks the temporal relationship between frames and the result is no longer temporally coherent. This manifests as a flickering effect where subtle changes between frames due to, for example, noise, small amounts of camera motion and the position of the labelling, can result in completely different depths in regions of the frame that have not actually changed. The most ideal situation would be to process the entire volume in a single volumetric fashion. Unfortunately, this does not bode well with memory, which is the subject of this section, and thus is not feasible for realizing.

To preserve temporal coherency, we propose a block processing scheme, as illustrated in Fig. 2.14. In this scheme, frames are processed in overlapping blocks. The size of the block is set to be large enough for that the depth map can be generated for that volume without completely exhausting all available memory. The overlapping frames are left unused since each block is treated as being independent of one another. While this breaks the temporal relationship between blocks, the overlap minimizes the change between blocks.

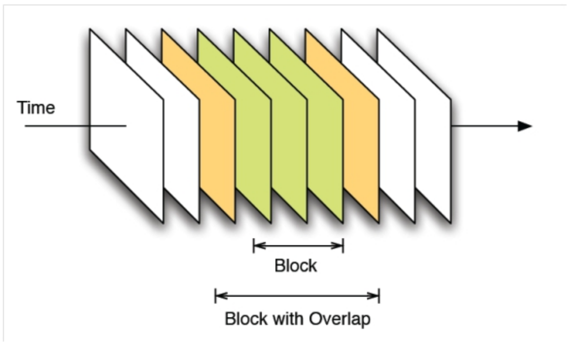


Figure 2.14: An example of block-based processing. The overlapping frames, which are the first and last of the shaded frames, act like a sliding window and help keep the results between blocks coherent

An advantage of the block-based processing scheme is that it provides a good balance between memory usage, temporal coherency and processing time. At the moment, the results for the overlapping blocks are discarded but the results can be used to ensure that changes between blocks are not too dramatic.

One potential use of these blocks is to be part of the initial conditions when the depth interpolation is performed. The amount of blocks used in each overlapping block is dependent in the particular scene at hand. Specifically, if the video sequence is of low resolution, and if the computer has an ample amount of memory, then the sequence should be able to comfortably fit in memory, and the entire volume can be processed. However, if the sequence is of higher resolution, then block processing is a necessity. For the results reported in this thesis, should block processing be employed, the default block size is 5. This means that for a given frame to convert, 2 frames before it, and 2 frames after are used as a small volume, and depth map estimation is performed on that volume. After completion, the depth map of the given frame is used, while the other depth maps are discarded.

Chapter 3

Tracking Labels: Object Tracking

IN this chapter, we discuss the first of the two approaches employed in this thesis for propagating user-defined strokes over a video sequence. This is performed using an algorithm for tracking objects in unconstrained video. However, before we discuss the object tracking framework used in this thesis, we provide the necessary background in object tracking.

3.1 Literature Survey

Object tracking is the task of the estimation of the motion of an object. Object trackers typically assume that the object to track is visible throughout the sequence. In order to track the object successfully, the object is usually transformed, or represented in another fashion that is robust and insensitive to noise and illumination changes. Examples of these different representations are through articulated models, where the problem is constrained to human body tracking [57][58][59], or through contours where a spline is wrapped around an object of interest, and its points tracked between frames [60][61][62][63]. However, in this thesis, because we wish to convert unconstrained video into stereoscopic 3D, we instead focus on methods that represent objects by geometric shapes, and their motion is estimated between consecutive frames, which is also known as frame-to-frame tracking. One of the most earliest works for tracking in this viewpoint was the work by Lucas and Kanade [64], where template matching was used. In fact, this is the most straight forward method to use, where the object is described by a target template. This could be an image patch, a colour histogram, or another representation. The motion is defined as the best transformation that minimizes the mismatch between the template, and a region of the image that potentially contains the object to track. Template tracking can be decomposed into either static tracking, where the object to track does not change in shape and only the displacement is changed, or adaptive which does not fall under this category. A good example of static tracking is through Mean Shift [65], and an example with adaptive tracking the Kanade-Lucas-Tomasi (KLT) tracker [66], based on [64]. Methods that combine static and adaptive template tracking have been proposed, such as the methods in [67], [68] and [69], as well as methods that recognize "reliable" parts of the template, like in

[70] and [71].

Unfortunately, most methods for template tracking only incorporate a single appearance of an object, and have a very limited ability to represent multiple appearances. Should the object deform in any way, template tracking will inevitably fail. As such, template methods can be amended so that more variations of the appearance of the object are modeled, known as *generative models*. These models are either built offline [72], or in real-time [73][74]. Trackers using generative models only model the appearance of the object and as such often fail in cluttered background. In order to circumvent this problem, recent trackers also model the environment where the objects moves as well. Two approaches to environment modeling are often used. The first method is by searching the environment for supporting objects, and the motion of these objects are correlated with each template of the object of interest to track. These supporting objects help in tracking when the object of interest disappears from the camera view, and undergoes a difficult transformation. Such examples are in [75] and [76]. The second method is that the environment is considered as a *negative class* against which the tracker should discriminate, known as *discriminative trackers*. These trackers usually build a binary classifier, representing the decision boundary between the object and its background. Static discriminative trackers, such as the approach by Avidan [77], train an object classifier before tracking, which limits their application to known objects. Adaptive discriminative trackers, such as those in [78], [79], [80] and [81], build a classifier during tracking. The essential phase of adaptive discriminative trackers is the *update*: the close neighbourhood of the current location of the object is used to sample positive training examples, while distant patches from the current location are used to sample negative training examples, and these are ultimately used to update the classifier in every frame. It has been demonstrated that this updating strategy handles significant appearance changes, short-term occlusions, and cluttered background. However, these methods also suffer from draft and fail if the object leavers the scene for longer than expected. To address these problems, the update of the tracking classifier has been constrained by auxiliary classifier trained in the first frame [82], or by training a pair of independent classifiers [83][84].

3.2 Focus: TLD

Building on what was discussed with object tracking, we can extend this to the realm of tracking the depth labels over unconstrained video. We can essentially take a stroke and extend a region around it to construct a template, and this template would be tracked over the rest of the video, ultimately automating the labeling process over the video sequence. In order to accurately track labels, a *long term tracker* should be employed, which can properly handle unconstrained cases over video, such as the sequences being possibly infinite in length, containing frame-cuts, fast camera movements, and objects disappearing. The techniques that we have previously discussed do not handle these situations very well. Therefore, the proposed tracking algorithm for this thesis is a modification to the one created by Kalal *et al.* [85, 86, 87], known as the *Tracking-Learning-Detection* framework, in which a long term tracker was created to handle the aforementioned issues. TLD closely integrates adaptive tracking with online learning of the appearances that are specific to the object desired to be tracked. The user simply has to draw a bounding box around the object of interest in the first frame, and its trajectory is determined over the entire video sequence using only this information, and the online learning. Specifically, the tracker learns a set of *positive* and *negative* patches, runs through a two-stage classification algorithm (which will be discussed later), and consults these patches to determine whether the object to track is in the frame, and its corresponding location. The positive patches correspond to those belonging to the object to track, while negative patches are those that do not contain the object, such as anything belonging to the background. Only those positive and negative patches that pass a threshold during the learning stage are what are used in the final stage of tracking the object. What is adaptive about this framework is that it is able to relearn positive and negative patches as the video sequence proceeds, as the object will most likely change in appearance, in addition to the background. Essentially, for each frame in the video sequence, the positive and negative patches are relearned on the fly, given the initial positive and negative patches from the previous frame, thus allowing the retraining to consume little computation time.

However, the tracker is optimally designed to track objects. When the depth strokes coincide to the background, these are handled differently, and will be discussed in Section 3.3.4. Marking only the first frame further reduces user effort, and is different from Guttmann *et al.*, where two frames must be marked for the system to function. If more than one frame is marked, particularly for the case of multiple shots in the sequence, the tracker is restarted, using the strokes in each user-marked frame as the "first" frame. The online detector incorporates multiple appearances of the object, including size and illumination changes. The detector examines boxes of multiple scales in each frame and determines whether the object is in any of these boxes. To increase computational efficiency, only bounding boxes of all possible scales that have a 60% overlap to the tracked bounding box of the previous frame are used.

3.3 TLD Tracking Algorithm

3.3.1 Preliminaries

The main input required is at least one sparsely marked user-defined initial depth map for a frame i in a video sequence, denoted as ${D_{\text{est}}}^{i}$. With this, the entire set of initial depth maps is denoted as $\mathcal{D}$, where $\mathcal{D} = \{{D_{\text{est}}}^{F(i)}\}$, and $F(i)$ is an indicator function, classifying whether the initial depth map of frame i is included in the set if marked by the user, and not included otherwise. It is not required to mark all frames in the video sequence; however, as seen in the previous chapter (Chapter 2), accuracy is inevitably increased if all frames are accurately marked, leading us to this purpose. It is also assumed that the video sequence is decomposed into shots, and for each new shot, a new user-defined labelling is provided for the first frame of each new shot. To demonstrate that the proposed tracking system is functioning properly, results must be shown in a purely tracking-based point of view, and will be shown in this chapter.

As user-defined depth strokes will inevitably vary in shape and length, it would be computationally intensive to track *all* possible points assigned an initial depth, thus hindering the system's real-time property. As such, we consider a single stroke $\mathcal{S}$, having an associated user-defined depth d_u, to be an ordered set of N points. This is defined as $\mathcal{S} = \{\vec{s}_0, \vec{s}_1, \ldots, \vec{s}_{N-1}\}$, where $\vec{s}_i$ is one of the N points on the stroke marked by the user. If the stroke is uniformly subsampled to contain roughly five to ten points, cubic spline interpolation can be used to faithfully represent the stroke with a high degree of fidelity. Therefore, for each user-defined stroke on a particular frame, a binary morphological thinning is used to reduce the stroke to a minimally connected one. After, it is uniformly subsampled and is finally represented by these points. In the case of where the user draws overlapping strokes at the same depth, a clustering is performed to merge these to a single stroke; morphological thinning is performed on this augmented stroke instead, and the points are collected in the same fashion. Once all points for all strokes are collected, they are individually tracked using a computer vision tracking algorithm, and their depths are automatically adjusted when subject to non-simple motion. When the tracking of the points is completed, cubic spline interpolation is used to faithfully represent what the final strokes are, constrained to the locations of the tracked points. In order to determine the width of the stroke, the smallest distance between centre of mass of the thinned stroke, with the perimeter of the original stroke before thinning is performed. This overall process inevitably creates "user"-defined strokes across *all* frames, except that a tracking algorithm was used to mark the frames. This ultimately reduces the amount of interaction involved, while maintaining the same amount of accuracy (if not better) for marking frames as would be done by the user. In the proposed work, five points are used to represent each user-defined stroke, and cubic spline interpolation with a natural spline is used [88].

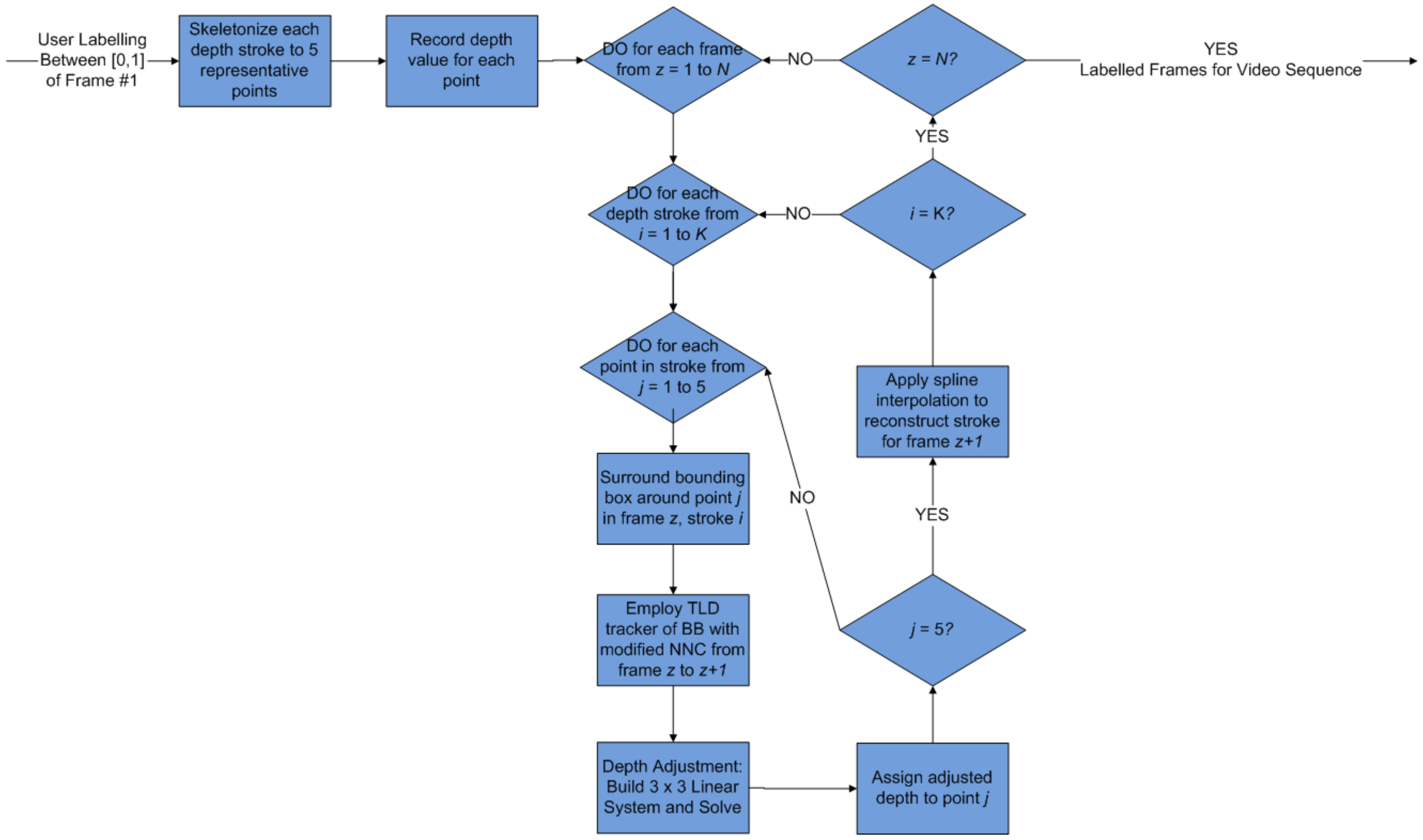

User Labelling Between [0,1] of Frame #1
Skeletonize each depth stroke to 5 representative points
Record depth value for each point
DO for each frame from z = 1 to N
NO
z = N?
YES
Labelled Frames for Video Sequence
YES
DO for each depth stroke from i = 1 to K
NO
i = K?
DO for each point in stroke from j = 1 to 5
Apply spline interpolation to reconstruct stroke for frame z+1
Surround bounding box around point j in frame z, stroke i
NO
YES
Employ TLD tracker of BB with modified NNC from frame z to z+1
j = 5?
Depth Adjustment: Build 3 x 3 Linear System and Solve
Assign adjusted depth to point j

Figure 3.1: A block diagram of the TLD tracking process for depth label tracking

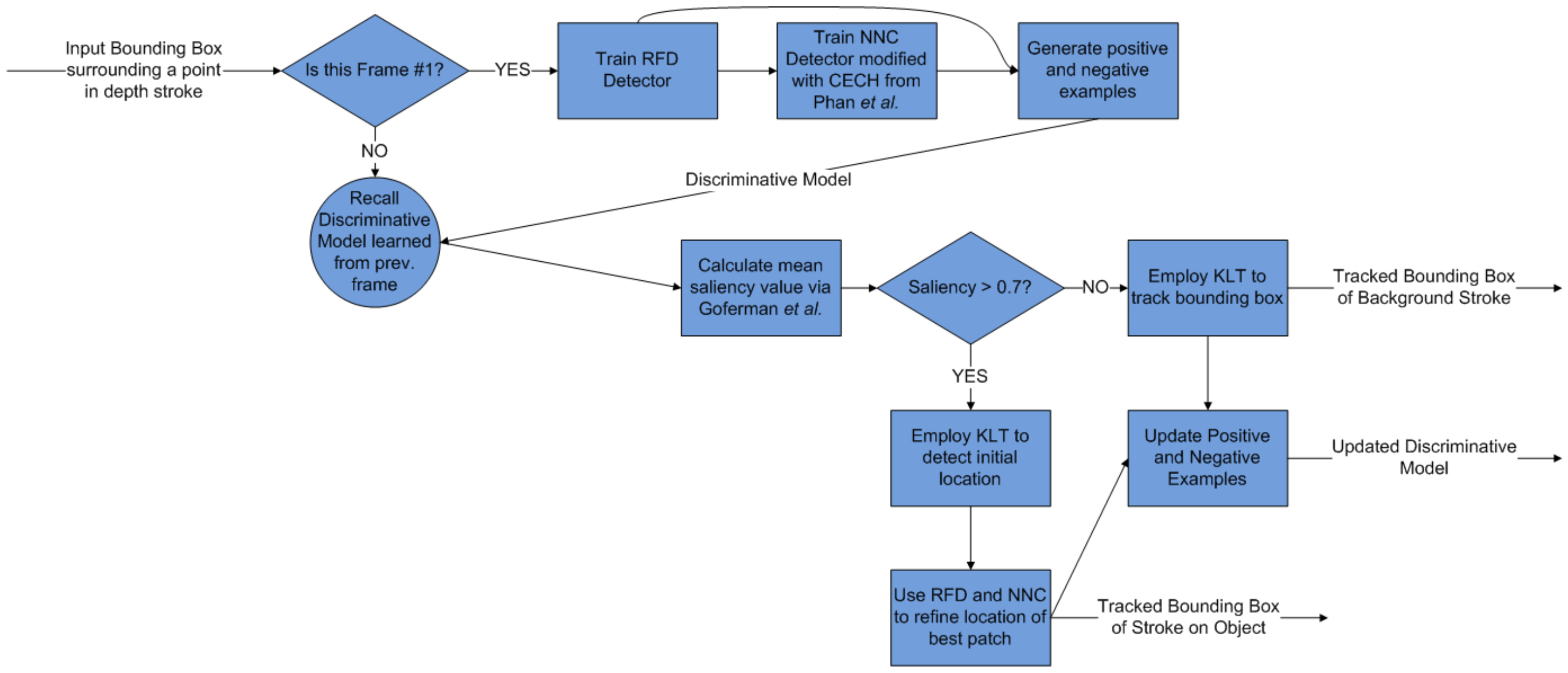

Input Bounding Box surrounding a point in depth stroke
Is this Frame #1?
YES
Train RFD Detector
Train NNC Detector modified with CECH from Phan *et al.*
Generate positive and negative examples
NO
Discriminative Model
Recall Discriminative Model learned from prev. frame
Calculate mean saliency value via Goferman *et al.*
Saliency > 0.7?
NO
Employ KLT to track bounding box
Tracked Bounding Box of Background Stroke
YES
Employ KLT to detect initial location
Update Positive and Negative Examples
Updated Discriminative Model
Use RFD and NNC to refine location of best patch
Tracked Bounding Box of Stroke on Object

Figure 3.2: A block diagram of the TLD tracking process for a single point along a depth stroke

3.3.2 Method

As a means of illustration, Fig. 3.1 illustrates a block diagram of the entire process employed by TLD tracker, and modified to perform depth label tracking. As previously, each component of the tracking framework will be explained as we proceed. In addition, as each point, relevant to a thinned and skeletonized depth stroke, is individually tracked, Fig. 3.2 illustrates the process of tracking one of these points, which is also referenced in the block diagram of Fig. 3.1 when the modified TLD tracker is employed to track a single point along a depth stroke.

For Kalal *et al.*'s method, the initial location of the object is tracked using the KLT in a multi-scale pyramidal sense, and the location is further refined by using a two-stage detector. This is comprised of a randomized fern detector (RFD) [89] and an intensity-based nearest neighbour classifier (NNC), resizing the pixels within a bounding box to a 15 x 15 patch, with normalized cross-correlation as the similarity measure. The KLT, as well as the randomized fern detector and nearest neighbour classifier all operate on monochromatic frames, and are converted accordingly if the input is in colour. Patches that surround the initial bounding box of the object are considered positive examples, while bounding boxes that are farther away from the initial bounding box are considered negative. However, it has been noted in literature that colour images are naturally acquired in the RGB colour space, with significant correlation between each colour component. Information from other channels could have contributed to the acquisition of the target channel, and simply converting to grayscale ignores this correlation [44]. However, the KLT, as well as the RFD can achieve good results ignoring colour information. In retrospect, the NNC can most definitely be improved on, as this can utilize colour information, and can be transformed into a more perceptually uniform colour space so that quantitative differences in colours reflect what is perceived by human beings.

To this end, the NNC framework is transformed in such a fashion, while maintaining illumination invariance. The similarity between two colour patches is based on Phan *et al.* [43], where the main feature is a colour edge co-occurrence histogram (CECH), based on the colour co-occurrence histogram (CCH). CCHs are three-dimensional histograms, indexing the frequency of pairs of colours over different spatial distances; the first two dimensions are the colours, and the third is the spatial distance. It should be noted that some form of colour quantization should be performed to allow for memory efficient storage. Nevertheless, CECHs are different where co-occurrences on valid edge pixels are considered, as highly uniform areas would overwhelm valid similarity metrics [90]. CECHs have been used in unconstrained object detection, and have the ability to detect objects subject to a variety of factors, like object deformation, scale and illumination changes [90]. When examining the CECH of a particular bounding box over multiple scales, this simply results in a proportional change to each CECH entry, and can thus handle multiple scales. The main differences between [43] and [90] are the colour edge detection and colour quantization mechanisms. In [43], they demonstrated that the definition of edges in [90] is not entirely accurate, and the colour quantization mechanism is subject to problems when certain illumination conditions take place. In [43], the edge detection mechanism was designed to detect proper edges based on vector-order statistics (as mentioned in Chapter 1), and a new colour quantization algorithm was developed to mimic human colour perception and recall very well, and is illumination invariant. This will inevitably help in the design of the NNC when determining similar patches.

For the video sequence, each frame is colour quantized and edge detected using the method in [43]. Once morphological thinning and the extraction of points along each stroke are obtained, an initial bounding box surrounding each of these points serves as input into the TLD framework, and is tracked throughout the rest of the video sequence. In the proposed work, a 35 x 35 window surrounding these points are used. To adapt the CECH framework to the TLD, the 15 x 15 patch resizing is not performed, as the ultimate goal is to transform a bounding box into a CECH. Instead, the bounding boxes with significant overlap have their CECHs computed, and are used in the NNC. To compute the similarity two histograms, and ultimately the image patches, there are many methods to compare two histograms, such as the L_1 distance, or Histogram Intersection; however, these have been shown to be suitable [90]. As such, a similarity measure is proposed that overcomes these problems, by properly measuring if the same proportion of colours is present within two patches, and the edge information is incorporated into the measure. This is based on a least-squares approach, where a linear relationship is determined between two patches. Let C_t be a trained CECH stored in the NNC, and let C_q be a query CECH for classification. Both CECHs are transformed into vectorized 1D signals of length n before proceeding, in order to suit the task of linear regression. These CECHs are designated as $C_t(x)$ and $C_q(x)$ respectively, where x accesses a particular bin value in either CECH. By least squares, a linear relationship is found:

$$C_t(x) = mC_q(x) + b + \epsilon(x) \quad , \tag{3.1}$$

such that the residuals for each pair of bin values, $\epsilon(x)$, is minimized. It should be noted that there is a possibility for duplicate $(C_q(x), C_t(x))$ pairs when vectorizing, and this will obscure the least-squares process. Therefore, any duplicate pairs are eliminated from the vectorized CECHs before proceeding. As the bias b for linear regression is not useful in this context, this is zeroed by subtracting each CECH with its corresponding mean bin value. After, the slope, m, can thus be calculated by:

$$m = \frac{\sum_{i=1}^{n} C_t(i)C_q(i)}{\sum_{i=1}^{n}(C_q(i))^2} \quad . \tag{3.2}$$

The slope is an indication of how similar in scale the query patch is from any of the patches in the trained dataset for the NNC. If m is close to 1.0, the contents of the two patches are roughly the same size. Obviously, this reasoning is false should there be a false positive, and the query patch *does not* contain any similarities to any of the trained dataset. Therefore, the correlation coefficient between the two CECHs is employed, denoted as r. A high quality match indicates good linear relationship exists between the two histograms. The correlation coefficient is defined as:

$$r = \frac{\sum_{i=1}^{n} C_t(i)C_q(i)}{\sqrt{\sum_{i=1}^{n}(C_t(i))^2}\sqrt{\sum_{i=1}^{n}(C_q(i))^2}} \quad . \tag{3.3}$$

When r is low, this most likely corresponds to a poor match between the two patches. However, r incorporates scale invariance for determining their similarity. Specifically, if r is high, the corresponding bin values between the two CECHs are all roughly scaled by the same amount, and thus a good linear relationship between them exists. The similarity, S_{CECH}, between the two CECHs is a combination of the slope, m_{CECH}, and the correlation coefficient, r_{CECH}. This similarity is defined as:

$$S_{CECH} = k_1 |r_{CECH}| + \max(0, (1 - k_1)(1 - |\log_2(m_{CECH})|)) \quad , \tag{3.4}$$

where k_1 is a constant such that $k_1 \in (0, 1.0)$. The logarithm ensures that a match is penalized equally for being N times too large, as it is for being N times too small. Not only should the pairs of pixels at valid edges correspond, the colours between both patches should be evaluated to ensure that same quantities of colour are present. Therefore, the same similarity measure as seen in Eq. 3.4 should be applied to the colour content, and so the colour histograms of the quantized frame are used in Eq. 3.4, instead of the CECHs. By repeating the pre-processing steps on the CECHs, if the query and trained colour histograms are 1D vectorized, with m_{CH} and r_{CH} being their corresponding slope and correlation coefficient, the similarity measure, S_{CH}, is:

$$S_{CH} = k_2 |r_{CH}| + \max(0, (1 - k_2)(1 - |\log_2(m_{CH})|)) \quad , \tag{3.5}$$

where k_2 is also constant such that $k_2 \in (0, 1.0)$. Therefore, the final similarity, $\mathbf{S}$, between two image patches is thus defined as:

$$\mathbf{S} = k_3 S_{CECH} + (1 - k_3) S_{CH} \quad , \tag{3.6}$$

where k_3 is also constant such that $k_3 \in (0, 1.0)$. Two patches are considered similar if Eq. 3.6 surpasses a threshold.

3.3.3 Adjustment of Depths

Performing linear interpolation of depth labels can mislabel points when subject to non-simple motion. To handle this, the depth labels are automatically adjusted, when this exists. To realize this, the ability of the TLD framework to detect objects over multiple scales is the main feature used. Let $\mathcal{B} = \{b_0, b_1, \ldots, b_i, \ldots, b_{M-1}\}$ represent the scales of the initial bounding box of size $R \times C$ created in the first frame, such that for a scale b_i, the bounding box has dimensions $b_i R \times b_i C$. These scaled bounding boxes are based on the dimensions of the initial bounding box. When TLD tracking is performed, as the object moves closer to the camera, the size of the bounding box required for the object will inevitably increase, as its perceived size will increase. The inverse relationship can be said as the object moves farther away. It should also be noted that this kind of motion will most likely be non-linear in nature. Therefore, a mapping function is created, describing the relationship between the scale of the detected bounding box, with the labelled depth to be assigned for that point, and a simple parabolic perturbation is opted for use. If $D(x)$ represents the mapping function for a scale $x \in \mathcal{B}$, a parabolic relationship can be determined, where $D(x) = ax^2 + bx + c$. With respect to TLD, two details are already known. The smallest scale b_0 can be represented as the farthest depth, d_0, and the largest scale b_{M-1} can be represented as the closest depth, d_{max}. Finally, the initial bounding box has a scale of 1.0, with the depth assigned from the user, d_u, and the coefficients are now solved by the inverse of the following system:

$$\begin{bmatrix} (b_0)^2 & b_0 & 1 \\ 1 & 1 & 1 \\ (b_{M-1})^2 & b_{M-1} & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} d_0 \\ d_u \\ d_{max} \end{bmatrix} . \tag{3.7}$$

At each frame, the bounding box scale automatically adjusts the depth at this point along a stroke. This relationship between the scales and depths functions quite well; if the current bounding box is of the original scale, object will have only exhibited horizontal motion. As the object moves closer and farther from the camera, the depth is automatically adjusted within this bounding box. This will function well when different parts of the object appear at different depths (i.e. when the object is perceived at an angle).

3.3.4 Tracking of Background Strokes

The TLD framework is designed to track objects, and the RFD, as well as the CECHs, assume that edges exist within the initial bounding box. However, as the goal of the proposed work is to determine dense depth maps, there will inevitably be strokes placed on the background, and the TLD tracker will fail. However, as the TLD tracker is also comprised of the KLT, if the stroke lies on a uniform background, the online learning, RFD and NNC should be disabled, and the KLT should only function. There should be some mechanism to automatically determine the salient regions in a frame, corresponding to the most activity, and will thus contain a good amount of edges. We thus use the saliency detection algorithm by Goferman *et al.* [91]. With respect to computer vision, saliency detection is a framework that identifies regions of the scene that a human being would focus on immediately, once it is presented to them. A good example would be a prominent object appearing on some background the salient region would be the object itself. Regions that are salient, or *important*, with respect to a human being, is the purpose of saliency detection. However, most saliency detection algorithms focus on just the regions that a human being would focus on. Goferman *et al.* correct this, by not only detecting salient regions, but they also include regions of the background so that the *context*, or the story behind the captured regions is included. This would be the most appropriate framework to use for determining whether a stroke point lies on an object, or on the background. Background strokes most likely correspond to those that have the least amount of activity or saliency, and would lie on uniform areas, and inevitably the background. As such, each frame has its saliency computed using [91], and for each bounding box centered at a stroke point, the mean saliency value is computed. If it surpasses a normalized threshold, this is considered salient, and the full TLD tracker is used. Otherwise, only KLT is used for tracking. In this work, a threshold of 0.7 is used.

3.4 Object Tracking Results

To illustrate that the introduction of colour and the CECH into the NNC shows improvement to TLD, tracking results from two sample video sequences are shown in this section. The system parameters have been experimentally determined to be: 1) TLD: a) KLT: 7 levels of pyramidal decomposition, 15 x 15 block size, b) RFD: 10 trees, 13 features with a 50% threshold, c) NNC (Original): 15 x 15 block size with a 0.7 threshold, d) Scales: 21 scales, such that $\mathcal{B} = \{1.2^x \mid x \in \mathbf{Z}, -10 \leq x \leq 10\}$. 2) CECH for Object Detection Framework: $k_1 = 0.1, k_2 = 0.1, k_3 = 0.5$, with a 0.7 threshold. 3) Saliency Detection: The same parameters in [91] are used. 4) Labelling Mechanisms: $d_0 = 0, d_{max} = 255$. It should be noted that there is an implementation of the TLD framework available online[1], and the aforementioned parameters are from this implementation.

Select frames of each result from both the original and the proposed improvements are shown. It is realized that a more thorough analysis to the merits of introducing colour should be performed. However, this is currently being investigated, but including these sample video sequences is enough to demonstrate that there is improvement. The sequences come from the 100 frame motorcycle sequence included in the online implementation of the TLD, and a 97 frame shot taken from the Sintel animated short film[2], denoted as *Market*. The dimensions of each sequence are 470 x 310, and 1280 x 545, and the Sintel sequence was downsampled by 50% to promote computational efficiency. Fig. 3.3 shows tracking results over various frames for the motorcycle sequence, where the top row shows the original TLD framework, and the bottom row shows the proposed improvements. The top is shown in grayscale, as the original framework operates this way. The desired object to be tracked is the motorcycle and rider, as was defined in the initial bounding box. As can be seen, the bounding boxes in the original framework do not fully encapsulate the object, but the tracking of its position is - in a qualitative sense. For the proposed method, the tracking is fairly accurate, *and* the bounding box fully encapsulates the object for most of the frames - a promising result when considering depth adjustment. In each frame, the yellow box is the bounding box used to track the object, while the blue points within the box are the features detected by the randomized fern detector stage. The columns on the left and right of each frame denote the negative and positive examples learned in the tracker. Specifically, the image patches seen on the left column are those that the tracker has learned that does not have the object within them with high probability. The patches on the right denote patches that do. The caption on the bottom left denote the frame rate of the sequence, and the frame number shown to the user at the current time. As can be seen in the proposed tracking results, more negative patches are learned to ensure high accuracy. In addition, the positive patches learned encapsulate more of the entire object, which is the rider *and* the motorcycle, rather than the rider itself. Both the rider and the motorcycle appear at the same depth in the scene, and so they should naturally be classified the same way. By using the original TLD framework, it only classifies the rider, and would not allow the motorcycle to be classified as the same depth due to the mislabeling.

[1]`http://info.ee.surrey.ac.uk/Personal/Z.Kalal/tld.html`
[2]`http://www.sintel.org/download`

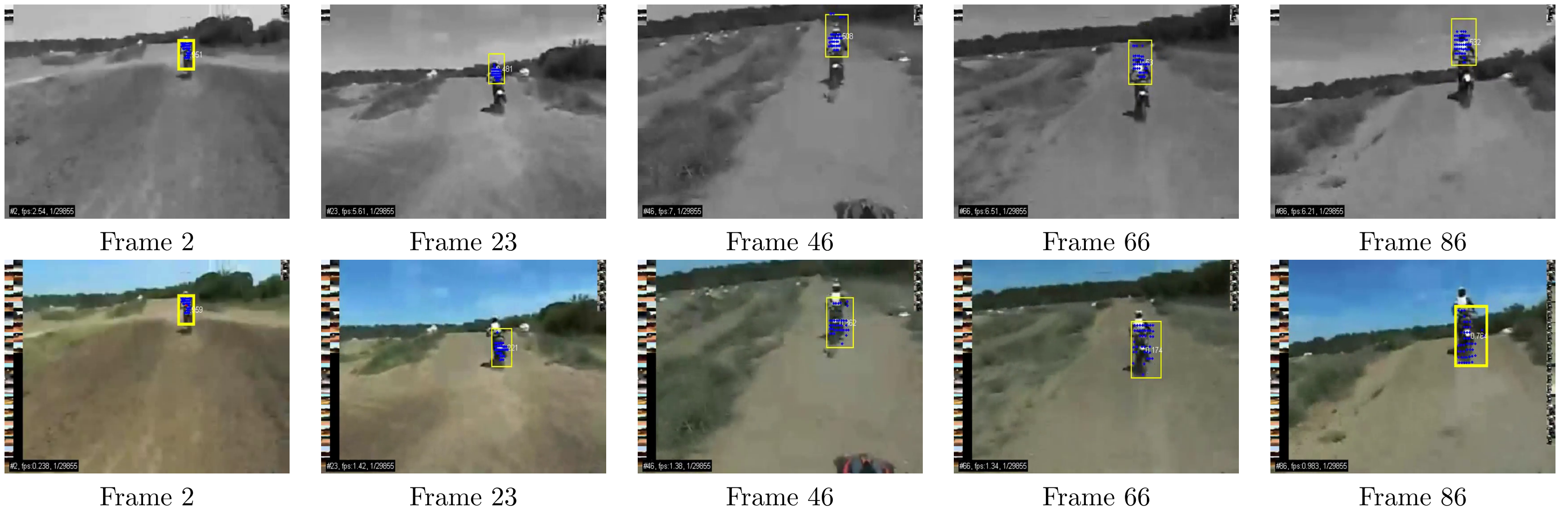


Figure 3.3: Motorcycle sequence - 1st row: Original TLD results. 2nd row: Results with proposed modifications to TLD.

In a similar style, Fig. 3.4 shows tracking results over various frames for the Sintel shot. The desired object is a village citizen, walking away from the camera. Interestingly, near the middle of the sequence, the original framework starts to track the hand of another villager, whereas in the proposed method, the original citizen is still being tracked. However, the proposed tracker begins to fail towards the end of the sequence, as other objects of a similar colour profile are entering the vicinity of the tracked bounding box. Nevertheless, it managed to track the villager for the majority of the sequence. What is also interesting in this sequence is that fewer negative and positive examples were required to be learned to faithfully track the sequence. This most likely is due to the colour information incorporated into the learning stage, as the modified CECH from [43] was able to differentiate between the object and the background.

There should also be some way to numerically assess the accuracy of the delineation of the bounding box for each frame in the sequence. As such, this thesis uses the framework by Phan *et al.* in [43], where each frame in the video sequence had a *ground truth* bounding box manually drawn, serving as reference frames. Bounding boxes are also generated using the original TLD framework, and the modified framework that is proposed. For each framework, an overlap calculation is determined between each of the bounding boxes with the ground truth bounding boxes. Should a bounding box surpass a threshold, this counts as a *proper* delineation, and a *missed* delineation is when it does not. For this thesis, this threshold is set to 0.6. The tracking method that obtains more proper delineations is declared the superior algorithm. Formally, a detection $\mathcal{D}$ for a particular frame in the sequence is defined as the following:

$$\mathcal{D} = \begin{cases} \text{Proper} & \text{if } \frac{Box_{GT} \cap Box_D}{Box_{GT} \cup Box_D} \geq 0.6 \\ \text{Missed} & \text{if } \frac{Box_{GT} \cap Box_D}{Box_{GT} \cup Box_D} < 0.6 \end{cases}, \tag{3.8}$$

where Box_{GT} is the area of the ground truth bounding box, while Box_D is the area of the detected or tracked bounding box from either of the two frameworks. In other words, a proper detection is the ratio of the intersection of the detected bounding box area and the ground truth bounding box area, over the union of the area of these two boxes when it is greater than or equal to 0.6. A missed detection is when the ratio is below 0.6. As such, Table 3.1 denotes how many bounding boxes were proper and missed between both frameworks for the Motorcycle and the Sintel Market sequence. As can be seen in the table, the proposed modified TLD tracker tracks the objects better than its original counterpart, as more valid delineations were produced by the aforementioned criteria.

Category	Motorcycle		Market	
	Original TLD	Proposed	Original TLD	Proposed
Proper	35	92	45	64
Missed	65	8	52	43

Table 3.1: Table illustrating proper and missed delineations of the object bounding boxes for the Motorcyle and Market sequences

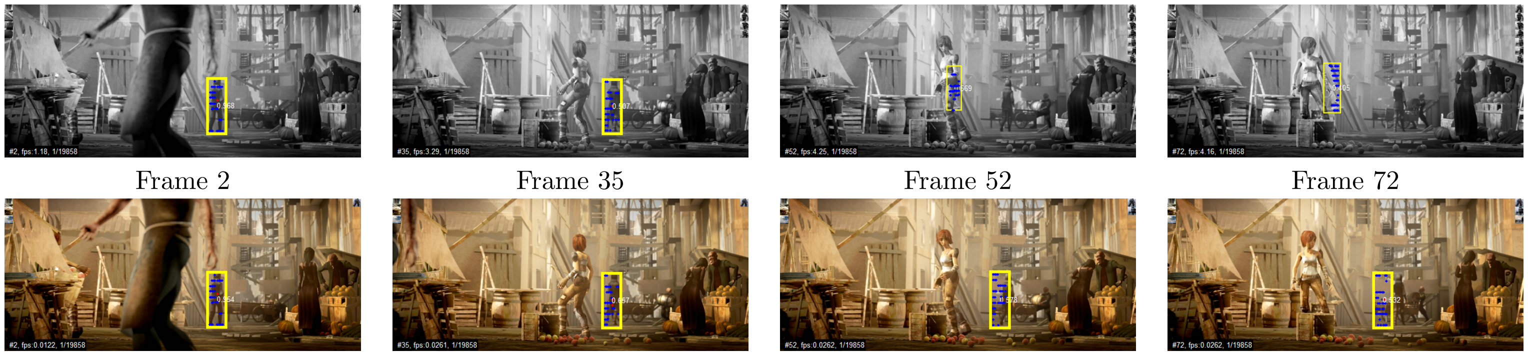


Figure 3.4: Sintel Market sequence - 1st row: Original TLD results. 2nd row: Results with proposed modifications to TLD.

3.5 Label Tracking Results

As a further means of illustrating the effectiveness of the tracker, results showing labelled frames from the modified TLD tracker will be shown. The user is only required to label the first frame, and the tracking algorithm tracks the labels throughout the rest of the frame, and reconstructs the strokes using spline interpolation when necessary. Two cases will be shown here: One where there exists only horizontal motion, and the other with motion perpendicular to the camera axis. The first case is necessary to illustrate that the tracker can be used when there is horizontal motion, and no depth adjustment is necessary, while the second case demonstrates what the modified TLD tracker was designed for. It should be noted that for any scenes that exhibit horizontal motion, it would be less computationally intensive to use the optical flow approach, which will be discussed later. Fig. 3.5 shows the label tracking results for another sequence originating from "Sintel", denoted as *Dragon*, and is comprised of 38 frames. This is a good example of a sequence exhibiting only horizontal motion, as the camera, as well as the dragon, are the only factors in the experience of motion in this sequence. Fig. 3.5(a) denotes the initial depth strokes placed by the user, while Figs. 3.5(b)-(f) illustrate the labels at frames 10, 20, 30, 35 and 38 respectively. These frames were automatically labelled by the modified TLD tracker and spline interpolation. As can be seen in the figure, the horizontal motion is experienced by the motion of the camera, and the label tracking algorithm is properly moving the labels in correspondence to this motion. This mostly coincides with the background and the stationary objects. What is also interesting is the labelling on the dragon itself, and they moving in synchronization with the moving of its body and head. Finally, Fig. 3.6 shows the label tracking results for the Shell-Ferrari-Partizan (SFP) sequence. This SFP segment was specifically chosen, as the race car moves away from the camera, and is a good test for depth adjustment. Fig. 3.6(a) denotes the initial depth strokes placed by the user, while Figs. 3.6(b)-(l) illustrate the labels at frames 6, 11, 16, 21, 26, 31, 36, 41, 46, 51 and 56 respectively. The race car is moving away from the camera, and so the front of the car should be classified as being farther away in comparison to the back of the car. This is properly performed by the depth adjustment, as evidenced by points along the race car that have been designated as closer to the camera, and the depth starts to grow farther as the car moves away. The actual depth maps and stereoscopic visualizations will be shown in the Results chapter in Chapter 5.

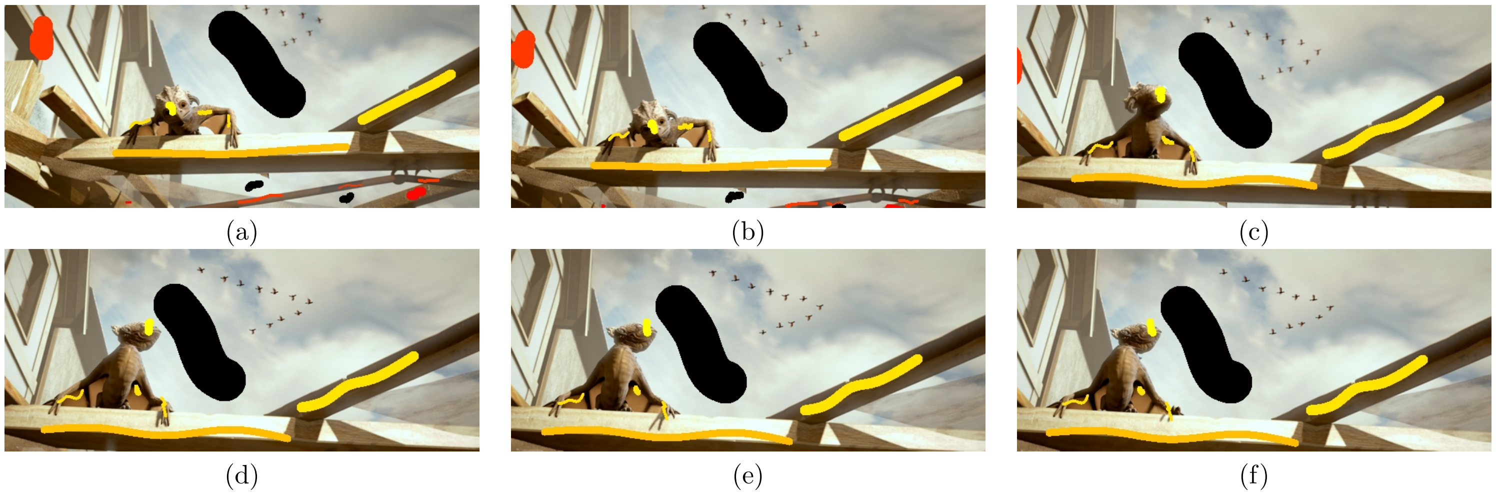


Figure 3.5: Sintel Dragon sequence - (a) User-defined labels for the first frame, (b) Frame 10, (c) Frame 20, (d) Frame 30, (e) Frame 35, (f) Frame 38

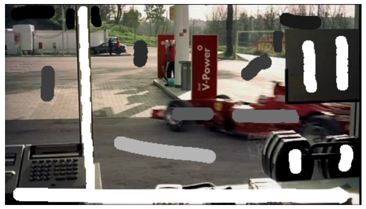
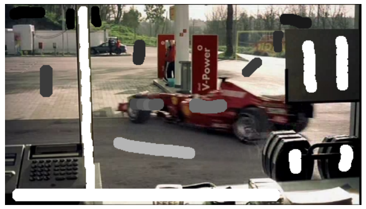
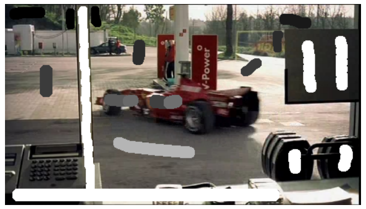
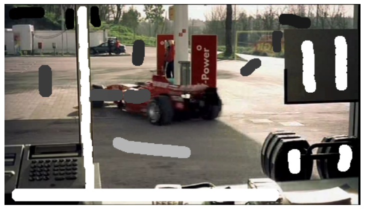
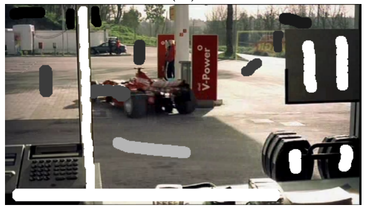
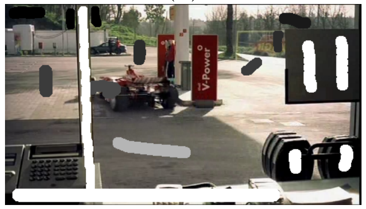
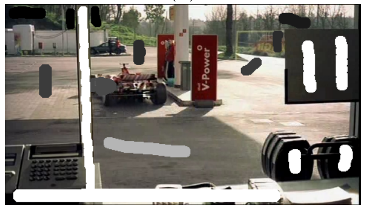
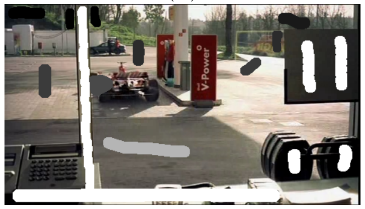
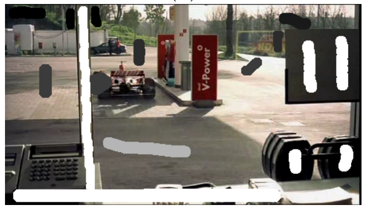
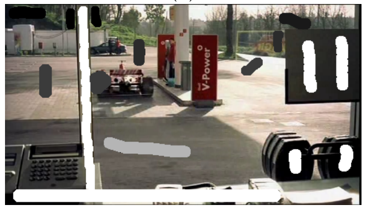
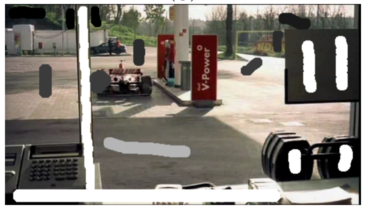
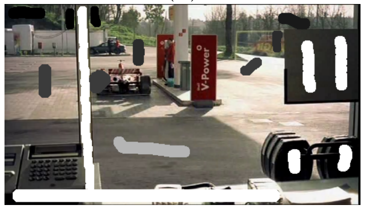

(a)

(b)

(c)

(d)

(e)

(f)

(g)

(h)

(i)

(j)

(k)

(l)

Figure 3.6: SFP sequence - (a) User-defined labels for the first frame, (b) Frame 6, (c) Frame 11, (d) Frame 16, (e) Frame 21, (f) Frame 26, (g) Frame 31, (h) Frame 36, (i) Frame 41, (j) Frame 46, (k) Frame 51, (l) Frame 56

Chapter 4

Tracking Labels: Optical Flow

With the use of the modified TLD tracker, it can become computationally intensive. There is an additional computational step using a saliency detector, and additionally, a set of randomized fern detectors is required to be trained: one per stroke point. Noting this drawback, we offer an alternative method that the user may choose. This is through the use of optical flow, which is another popular framework for describing the motion in a video sequence. Specifically, this thesis proposes an edge-aware temporally consistent optical flow algorithm should the scene exhibit only horizontal motion and can avoid some of the computational burden using the TLD, without the use of global optimization. This can become very attractive for implementation in dedicated parallel architectures, such as GPUs or FPGAs. As with the previous chapter, before we discuss the optical flow framework used in this thesis, we provide the necessary background in optical flow.

4.1 Literature Survey

Optical flow attempts to describe how each point in the scene moves from a frame to the next [92][93]. This problem is well studied, and is an essential component to areas, such as robot navigation [94], motion estimation and video compression [95], video interpolation for restoration and post-processing [96], and video interaction for manipulating objects and creating image composites [97]. Several aspects make this problem particularly challenging. The first of all such problems are occlusions. Occlusions are points that can appear or disappear between two frames. The second problem is what is known as the *aperture problem*, where in regions of uniform appearance, local features do not provide any information about the motion, and only partial flow information can be recovered along one-dimensional structures.

Most traditional optical flow algorithms operate by matching pixels from one frame based to the next based on their intensity values, or their colours. However, since there are usually many pixels of the same colour in a frame, this is not a discriminative feature, thus leading to errors. To increase the discriminative power, one can match blocks or groups of pixels, like what was done in [64], or the seminal work done by Horn and Schunck [92]. However, determining the size of blocks is challenging, as not all points within a block have the same displacement vector from one frame to the next. To rectify this, optical flow is traditionally solved by propagating local evidences for particular flow values across the frame, so that ambiguous regions get resolved due to nearby corners and textured areas. Essentially, optical flow is solved in a *global* optimization framework. Some energy function describing the flow is formulated and is minimized. One of the most popular methods to do this is through using a Markov Random Field (MRF), and building a model over the entire image. While the MRF formulation is well formulated, only approximate solutions can be found through the use of *iterative* methods, which can be computationally intensive. Many algorithms, such as Graph Cuts [98], Loopy Belief Propagation [99], and Tree Re-Weighted Message Passing [100][101] have been proposed for computing good approximate solutions. Other methods attempt to fuse multiple features together and formulate an energy function to be solved within an MRF scheme [102][103][104]. The runtime complexity, however, range from quadratic to cubic with respect to the number of pixels, as well as the number of labels that each pixel can take, which makes these methods computationally prohibitive in practice.

Recently, efficient methods have been proposed, relying on local operations, and their implementations placed on parallel hardware, such as GPUs [96][105][106]. To propagate information across the image domain, it must be down in an iterative fashion. Though this may bode well with low-resolution videos, this certainly does not with high-resolution videos, as this requires many iterations to reach distant points in the frame. This can be resolved with the use of multi-resolution pyramidal schemes, where a multi-scale pyramid of the pairs of frames is constructed. The pyramid contains frames of multiple resolutions, starting from a relatively small resolution (the coarsest scale), up to the original scale (the finest scale). The flow is estimated at the coarsest scale, and is propagated to the next scale, with its flow estimated again. This process is repeated until the original scale is reached. Performing a flow estimation in this fashion will allow for faster estimation and convergence. However, for high-resolution video, processing every pixel becomes a costly operation. Essentially, the running times for these approaches grow linearly, or perhaps beyond linearly as the size of the frame increases. Even optimization implementations eventually become slow due to the sheer number of pixels to process.

4.2 Focus: SimpleFlow

To address the aforementioned challenges, Tao *et al.* [107] proposed a *non-iterative*, sub-linear algorithm to estimate the flow, only computing a sparse set of samples in regions with uniform motion. Because this is a non-iterative framework, this *avoids* using global optimization all together, which also encourages implementation in parallel architectures. With the use of optical flow, to perform label propagation, one simply needs to move pixels that coincide with the user-defined strokes by the displacements determined by the optical flow vectors, and this is exactly what is done in our framework. It should be noted that this approach is suitable only for motion and objects that are moving horizontally. Currently, we have not developed a way of dynamically adjusting the depths of the objects using optical flow like in Section 3.3.3, but is currently an ongoing area of research being pursued. Specifically, if there is a method for being able to eliminate using TLD all together, and rely only on optical flow methods, the system would inevitably be faster, and how lower computational complexity.

SimpleFlow also runs in a multiscale fashion; for each scale, the pixels are processed independently, and only once. This ensures the computational complexity is at most linear. In addition, as the data is upsampled in the multi-resolution pyramid, certain areas of motion may not be as significant, and so the flow vectors can be interpolated. Rather than performing full calculations, this further expedites computational time. However, as the algorithm only considers local evidence, scenes with fast motion will inevitably fail. Also, SimpleFlow may miss small objects in the middle of regions with non-significant motion, as interpolation is applied. However, with most optical flow methods, the flow is only estimated between pairs of consecutive frames. There may be information over neighbouring frames of video that may improve accuracy. As such, we should ensure that any optical flow estimation algorithm is *temporally consistent* across the entire video sequence.

For this method, this thesis proposes to augment the SimpleFlow method by introducing *edge-aware filtering methods* that directly operate on the optical flow vectors. We also consider information over neighbouring frames to increase accuracy in the case of video. In most edge-aware filtering methods, the filtered outputs are generated by using a guidance image, serving as additional information to filtering that the output must satisfy. This is useful for any inaccuracies when interpolation is used in SimpleFlow. We can thus improve the accuracy by using the frames in the video sequence as guidance images. The flows for each frame pair can be filtered independently, but for the vectors to be temporally consistent, we also filter *temporally.* As the flow is only computed between two consecutive frames independently, it is inevitable that considering motion over all frames as a single volume will be non-smooth. As such, edge-aware filtering on a temporal basis, with frames as guidance images will inevitably allow smoothness across the video, and become temporally consistent.

In order to be fully self-contained, the SimpleFlow algorithm is briefly discussed, followed by the edge-aware filtering method we employ. After, the proposed augmentation to SimpleFlow will be given.

4.2.1 Preliminaries

We first establish some notation before we continue. A pair of consecutive frames at a particular point in time t is denoted as I_t and I_{t+1}. $I_t(x, y)$ are the spatial locations of (x, y) within frame t, while $O_t(x, y)$ is a vector-valued function, producing the horizontal and vertical flow vectors (u, v) at each $I_t(x, y)$. Optical Flow attempts to estimate (u, v) at each (x, y) in I_t so that the same point is visible at $(x+u, y+v)$ in I_{t+1}. Formally, a flow vector at (x, y) is represented as $(u(x, y), v(x, y))$, but is shortened to (u, v) whenever possible. Finally, $\vec{I}_t(x, y)$ denotes the RGB colour pixel value at (x, y) in I_t.

4.2.2 SimpleFlow Algorithm

We describe how SimpleFlow is applied on a single scale, and extend this to a multi-scale approach. Like most Optical Flow algorithms, the classical constant colour assumption is employed. The flow vectors (u, v) are found such that $\forall(x, y) \in I_t(x, y)$, they are as similar as possible to $I_{t+1}(x + u, y + v)$. This is modeled as [107]: $e(x, y, u, v) = ||\vec{I}_t(x, y) - \vec{I}_t(x + u, y + v)||^2$, which is the square difference between two colours.

The flow is assumed to be locally smooth. SimpleFlow requires the flow vector at any $I_t(x, y)$ to be a good explanation of the motion, as well as other pixels within some spatial neighbourhood $\mathcal{N}$ centred at (x, y). By incorporating information within $\mathcal{N}$, to determine the best (u, v) at (x, y), we consider a set of flow vectors Ω and choose within this so that the constant colour assumption is maintained. This is achieved by minimizing the energy collected for each pixel within $\mathcal{N}$. Tao *et al.* apply weights to this minimization to account for how pixels relate to each other, both spatially and with their colour difference. The lowest cost flow vector (u_0, v_0), centred at a pixel (x_0, y_0) in I_t is defined as:

$$(u_0, v_0) = \underset{(u,v)\in\Omega}{\arg\min} \sum_{(x,y)\in\mathcal{N}} \omega_d \omega_c e(x, y, u, v) \tag{4.1}$$

$$\omega_d = \exp\left(-(||(x_0, y_0) - (x, y)||^2)/2\sigma_d^2\right) \tag{4.2}$$

$$\omega_c = \exp\left(-||\vec{I}_t(x_0, y_0) - \vec{I}_t(x, y)||^2/2\sigma_c^2\right) \tag{4.3}$$

σ_d^2 is the spatial variance, determining how many points to include. σ_c^2 is the range variance, controlling how dissimilar two colours are. Interestingly, these weights correspond to the bilateral filter, making minimization possible using optimized schemes [108]. Minimizing will produce flow vectors that are integers. To obtain sub-pixel estimates, another bilateral filtering operation is performed directly on the flow vectors themselves, under the same $\mathcal{N}$, Ω and ω_d, ω_c weights, except a new weight, ω_r, is used to represent how reliable the flow vector estimate is at (x_0, y_0):

$$\omega_r = \underset{(u,v)\in\Omega}{\text{mean}}\, e(x, y, u, v) - \min_{(u,v)\in\Omega} e(x, y, u, v) \tag{4.4}$$

Operating at a single scale is not efficient, as large neighbourhoods need to be employed to account for large motion, bringing us to the necessity of a multi-scale approach. An image pyramid is constructed so that for each scale s, scale $s + 1$ is subsampled by a factor of 2. For the first scale, the flow is initialized to all zero. Given a flow estimate at scale $s + 1$, after minimizing Eq. 4.1, joint bilateral upsampling is used, performing bilateral filtering with a larger support window for the next scale s [107]. The results are multiplied by 2 and used as initialization for scale s. This produces flow vectors (u_s, v_s) for each pixel, and the minimization is performed at s centred at $(x_0 + u_s, y_0 + v_s)$. As a final means of efficiency, for each scale and before upscaling, pixel blocks are analyzed to determine if there is any significant motion present. This is measured by calculating an irregularity value for each block, centred at (x_0, y_0) in the flow field for each scale. The irregularity at the centre (x_0, y_0) is defined as $\max_{(x,y)\in\mathcal{N}} ||(u(x, y), v(x, y)) - (u(x_0, y_0), v(x_0, y_0))||$. Before upscaling, if this value is above a threshold τ, the full minimization is performed on the entire block. If not, then the minimization is performed only on the four corners of the patch, and the other flow vectors are determined by bilinear interpolation within the four corners.

4.2.3 Edge-Aware Filtering Method

The method we use is from Gastal and Oliveira [109]. This filters images, preserving edges and filters noise based on a dimensionality-reduction strategy, achieving significant speedups over existing techniques, while achieving high-quality results. In [109], *distance-preserving* transforms are the main tool for edge-aware filtering. The idea is that if an image is transformed into another domain, such that the colour distances in the original domain *is the same* in the transformed domain - one that is a lower dimensionality, filters designed in the transformed domain will be edge preserving, and is known as the *domain transform*. We start with determining the domain transform of a 1D signal, whose elements consist of multiple components (i.e. a single row of an RGB image). After, we move to the case of 2D signals, and discuss the filtering mechanism. If $I(x)$ is a continuous form of this 1D signal, where each element has c components, its domain transform, $ct(u)$, is defined as [109]:

$$ct(u) = \int_0^u 1 + \frac{\sigma_s}{\sigma_r} \sum_{k=1}^{c} |I'_k(x)| \; dx \quad , \tag{4.5}$$

where $I'_k(x)$ is the derivative of the k^{th} component for $I(x)$. σ_s^2 and σ_r^2 are the spatial and range variance, defined previously, and specified in the *transformed domain*. As we are dealing with discrete signals, the integral is simply replaced with a sum. Also, the original signal has its dimensions reduced from c dimensions to *one* dimension. For our purposes, $I(x)$ is a guidance signal - where its edge information is used to smooth the signal we wish to filter. This transformed data is ultimately used in the filtering operation, and the one we choose is normalized correlation. Formally, if $J(x)$ is the desired filtered signal, and Ω represents the domain of x values that $I(x)$ takes on, because the signals that we deal with are discrete in nature, $I(x)$ must be sampled and so this set of points is from the discretized set $D(\Omega)$. As such, for a point $p \in D(\Omega)$, the normalized correlation filter is defined as [109]:

$$J(p) = (1/K_p) \sum_{q \in D(\Omega)} I(q) H(ct(p), ct(q)) \quad , \tag{4.6}$$

where $K_p = \sum_{q \in D(\Omega)} H(ct(p), ct(q))$ is a normalization factor. $H(ct(p), ct(q))$ is a box kernel, defined for the transformed co-ordinates at p and q for $I(x)$, and is $H(ct(p), ct(q)) = \delta\{|ct(p) - ct(q)| \leq r\}$, where $r = \sigma_s\sqrt{3}$ is the filter radius. δ is a boolean function, where it is 1 if the argument is true, and 0 otherwise. Since the signal is discretized, when computing the domain transform, the derivative is simply $I(p+1) - I(p)$, for $p \in D(\Omega)$. For k points in this filter, the complexity is $O(k^2)$. However, in [109], they show that a moving-average approach can be used, and the filter can be performed by creating a buffer storing the transformed data. To filter the next point, a filtering operation requires only one subtraction and addition, reducing the complexity to $O(k)$.

However, the above is only for 1D signals. To filter 2D signals (i.e. images), in [109], they show that no 2D domain transform exists, so an approximation must be made, leading to iterations. Each row is independently filtered, then each column is filtered after, using the intermediate row results. Even though iterations are not desired, these allow the solution to be more accurate than using just

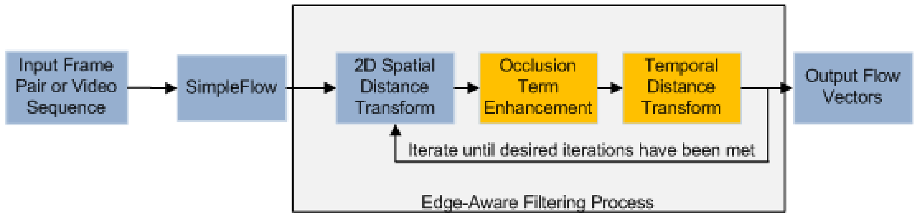

Input Frame Pair or Video Sequence
SimpleFlow
2D Spatial Distance Transform
Occlusion Term Enhancement
Temporal Distance Transform
Output Flow Vectors
Iterate until desired iterations have been met
Edge-Aware Filtering Process

Figure 4.1: Block diagram of the proposed framework. Blue blocks are mandatory steps, while orange blocks are optional depending on the user and the input

SimpleFlow. In [109], they demonstrated that between three and five iterations should suffice, but for each iteration, the filter radius must change, similar to the multi-scale approach previously mentioned. Specifically, for the i^{th} iteration, the filter radius r_i is:

$$r_i = \sigma_s \sqrt{3} \left(\frac{2^{N-i}}{\sqrt{4^N - 1}} \right) \quad , \tag{4.7}$$

where N is the total number of iterations. Once the signal is filtered at the i^{th} iteration with r_i, the result is used as input for the $(i+1)^{th}$ iteration, and filtered using r_{i+1} as the radius. One iteration is one complete filtering of the rows, followed by the columns. It should be noted that $ct(x)$ is computed only once on the guidance image, and used with all r_i for the box kernel H.

4.2.4 Proposed Method: Edge-Aware Temporally Consistent Optical Flow

Fig. 4.1 shows a block diagram of our framework. The blocks in blue are mandatory steps, while the blocks in orange are optional, depending on what the user chooses, or what the input is. We discuss each component in further detail. With both the knowledge of SimpleFlow and the domain transform, using the image frames as guidance images, and the desired signals to filter are the optical flow vectors per frame, it seems intuitive that the vectors will increase in accuracy. Also, if we wish to compute the flow from I_t to I_{t+1}, then I_t should be the guidance image. The guidance images should mitigate any errors in the flow maps due to occlusions, as their edge information will "guide" the filtered vectors. Intuitively, a 2D filtering operation on each flow map can be performed independently. However, this does not address *temporal consistency*. For a video sequence to be *temporally consistent*, information from the other frames in the sequence should be incorporated, or employed in the current frame. This information can help resolve ambiguities, such as occlusions, disocclusions, or noisy areas in certain frames. Traditional algorithms over video compute flow maps independently using two consecutive frames, and when considering the motion over the entire video sequence as a whole, the motion may be non-smooth. For the case of the two-frame case, however, only a 2D filtering operation is performed.

To address temporal consistency, one complete iteration consists of a 2D filtering operation, with the addition of a *temporal* filtering operation. In fact, this is similar to the framework developed by Lang *et al.* in [110]. However, they do not compute a dense optical flow, but determine a sparse one with corresponding points using SIFT. This flow map is filtered using the method in [109], with the addition of a temporal filtering step, similar to our approach. However, we opt to provide a dense flow, as there will be matched points that are inevitably inaccurate, and the flow map is heavily dependent on how well SIFT can match points between two frames. Should the matching be inaccurate, the flow maps will also be inaccurate. During an iteration, 2D filtering is performed using the same σ_s and σ_r, but temporally, different variances need to be specified, as the scale of the temporal domain will be different, which we denote as σ_{st} and σ_{rt} respectively.

How we filter in the temporal domain is as follows. For I_t, after a 2D filtering operation, and at a particular (x, y), each $I_t(u, v)$ serves as one component into a 1D signal for that location. As such, There will be rows $\times$ cols 1D signals whose lengths are the total number of frames. For a particular 1D signal, located at (x, y), the t^{th} element is a two-component value: $O_t(x, y)$. In addition, rows $\times$ cols 1D signals are constructed consisting of the frame data to use as guidance signals. Here, at each $I_t(x, y)$, the t^{th} component is simply $O_t(x, y)$, and the domain transform is computed for all of these to filter temporally. As in [110], one must be mindful when filtering temporally. The optical flow vectors are used to determine where points in I_t map to I_{t+1}. To determine these, one simply needs to take the every (x, y) in I_t, and determine where the pixel moves by $(x + u, y + v)$ to be mapped to I_{t+1}. By doing this, we can determine a binary map B that is the same size as I_{t+1}, where if a pixel from I_t gets mapped to (x, y) by performing $(x + u, y + v)$, the value at $(x + u, y + v)$ is 1, and 0 otherwise. The binary map determines those locations where the flow is reliable. Those locations with 0 denote that no pixel from the previous frame got mapped to these, as there were other locations in I_{t+1} that better maintained the colour constancy assumption. Also, it will be these locations that cause the motion to be non-smooth. With the aid of B, we ensure that the flow vectors are temporally consistent. To filter temporally, the same technique applied to 1D signals is performed to each of the temporal 1D signals. If $B_{(x,y)}$ is 1, then the normalized correlation is computed normally for the 1D signal located at (x, y). However, should the location be 0, the box filter is reinitialized by stepping backwards in time by the box filter radius (in the transformed domain). This is achieved by determining the points at the beginning and end of the box filter or $ct(p), ct(q)$, and compute what the values of $ct(p - r_i), ct(q - r_i)$ are (r_i defined in Eq. 4.7). Once it is re-initialized, normalized correlation proceeds as normal.

To further increase accuracy, we introduce an occlusion term, so that those locations that are likely to be occluded do not contribute substantially to the output. Similarly, for those pixels that are reliable in B between I_t and I_{t+1}, these locations should contribute a substantial amount. We denote a confidence map C where each (x, y) computes how reliable the flow vector is, going from $I_t(x, y)$ to $I_t(x + u, y + v)$. By denoting (u_f, v_f) as the *forward flow* going from I_t to I_{t+1}, and (u_b, v_b) as the *backward flow* going from I_{t+1} back to I_t, we propose the confidence for each pixel $C(x, y)$ to be based on the sigmoidal function:

$$C(x, y|(u_f, v_f), (u_b, v_b)) = \frac{2}{1 + \exp\left(\beta||(u_f, v_f) + (u_b, v_b)||\right)} \quad . \tag{4.8}$$

β is a constant controlling the rate of decay for the function. We experimentally determined β to be 3. The above equation is intuitive - theoretically, the forward and backward flows should be the negation of each other, and the output is 0 when both are summed. The more unreliable the flow at (x, y) is, the

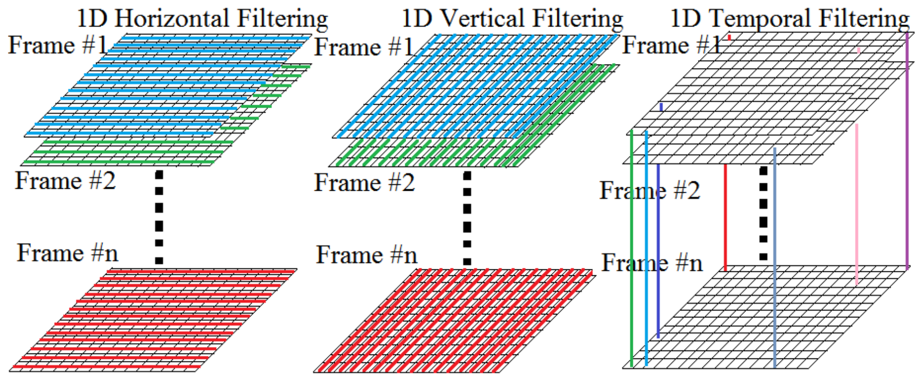


Figure 4.2: One filter iteration. Each coloured line represents the filtering direction for a particular case. For brevity, a few lines for temporal filtering are shown

higher $||(u_f, v_f) + (u_b, v_b)||$ should be, most likely dealing with occluded pixels. To properly incorporate this, confidence maps are computed for every frame *prior to* temporal filtering. For each frame, it has a C associated with it. These maps are 2D filtered by the domain transform, similar to the actual frames themselves. To incorporate the occlusions, if C_t is the confidence map for I_t, then the result used prior to temporal filtering is $(C_t \cdot I_t)'/(C_t)'$, where $(C_t)'$ represents a 2D filtering operation applied to C_t, and $(C_t \cdot I_t)$ is a per-element multiplication of C_t and I_t together. C_t is filtered separately, in addition to $(C_t \cdot I_t)$. Both results are divided on a per-element basis, and $(C_t)'$ ensures a normalization of the result [110]. The sigmoidal function is desired, as the function will never be fully zero, thus avoiding any division by zero errors. Therefore, should the confidence map be desired when filtering, at a given iteration, every I_t has a C_t computed, and the aforementioned necessary operations are performed for the temporal step. It should be noted that backward flows need to be calculated, so in addition to finding the flows from I_t to I_{t+1}, the flows from I_{t+1} back to I_t are necessary. With the computation of the backward flows, to go from I_{t+1} back to I_t, I_{t+1} must be used as the guidance image. For clarification, Fig. 4.2 shows one filtering iteration. For each frame, 2D filtering is performed, with the optional occlusion handling step, followed by a temporal filtering step for the case of video. For each frame, the rows are filtered independently with 1D horizontal filtering, whose results are processed by filtering the columns with 1D vertical filtering. If the input is video, each $I_t(x, y)$ has a 1D signal, the domain transform of each is computed, and filtering is performed temporally. The output is used for the next iteration, where the filtering is repeated.

4.3 Optical Flow Results

We show results for both the two frame case, and the multi-frame case. To take advantage of the parallel nature of SimpleFlow, we use the OpenMP paradigm to make use of the multiple cores on the CPU. These tests were performed on an Intel Core 2 Quad Q6600 with 8 GB of RAM. The implementation was on MATLAB for pre-processing, and C++ for the core algorithm. In addition, for all frames, each channel was normalized to $[0, 1]$. All run times reported include pre-processing time. The parameters we use were experimentally found. For SimpleFlow: Ω is a 11 x 11 window, $\mathcal{N}$ is 5 x 5, $\sigma_d = 5.5$, $\sigma_c = 0.08$, $\tau = 10$. For upsampling, we use: $\mathcal{N}$ as 11 x 11, $\sigma_d = 11$, $\sigma_c = 0.1$. For the domain transform, we use: $\sigma_s = 2000, \sigma_r = 0.4, \sigma_{st} = 5, \sigma_{rt} = 0.1$. For the two frame case, we use reference pairs from the Middlebury Optical Flow database, and show three examples in Fig. 4.3. The first reference image of each example is in the first column. The second column is the ground truth flow, and the last three columns show various results. All flows are colour stylized in the standard set by Middlebury. The middle column illustrates using only SimpleFlow, while the last two columns illustrate using just the 2D domain transform, followed by augmenting with occlusion terms. It should be noted that temporal filtering here is not applicable; however, we desire to show our framework for the two-frame case to ensure that the proposed formulation is valid. The first row is known as “RubberWhale”. As can be seen with SimpleFlow, there are many errors - especially around the edges of the circular object and half moon shapes. When applying the domain transform on the flow vectors, with the aid of the guidance image, this ensures that the edges are well respected, and the flow vectors bordering the edges are smooth. By referring to the earlier problems, the results have dramatically improved. Also, by applying occlusion handling, the edges around all the objects are sharper. The same observations can be said regarding the next row, known as “Urban2”. Many of the edges are not well respected, and some areas have noisy flow vectors. With the domain transform, and the help of the guidance image, these noisy areas become resolved. When including the occlusion term, the edges are more pronounced, eliminating the contributions occlusions make to the final output. Finally, the last row, known as “Venus”, is an example that shows where SimpleFlow fails, and our proposed framework helps correct the errors produced. For the Middlebury colourization, darker colours map to flow vectors that have a higher magnitude, and these vectors may extend beyond the edges of some objects. With the guidance image, this ensures that the flow vectors stay within the boundaries of the objects, and looks very similar to ground truth. In addition, there is almost no change between only using the 2D domain transform, with using occlusion terms. This is attributed to the fact that there are no occlusions in the scene, and so every point contributes greatly to the output.

To quantify the comparison, we show numerical results using the angular and endpoint error measures - two popular assessment methods for Optical Flow [93]. The angular error determines the angle in degrees between a vector from a computed optical flow, with a ground truth vector in 3D space. In other words, given that u and v are the horizontal and vertical components of the optical flow that were computed by an algorithm, and u_{GT} and v_{GT} are the ground truth horizontal and vertical components of the scene, the angular error θ_{err} is defined as:

$$\theta_{err} = \cos^{-1}\left(\frac{1.0 + u \cdot u_{GT} + v \cdot v_{GT}}{\sqrt{1.0 + u^2 + v^2}\sqrt{1 + u_{GT}^2 + v_{GT}^2}}\right) \tag{4.9}$$

The angular error measure states that the greater the angle, the higher the error. The endpoint error is the Euclidean distance between the computed flow and ground truth vectors. Using the same notation for the horizontal and vertical components for the computed optical flow and the ground truth,

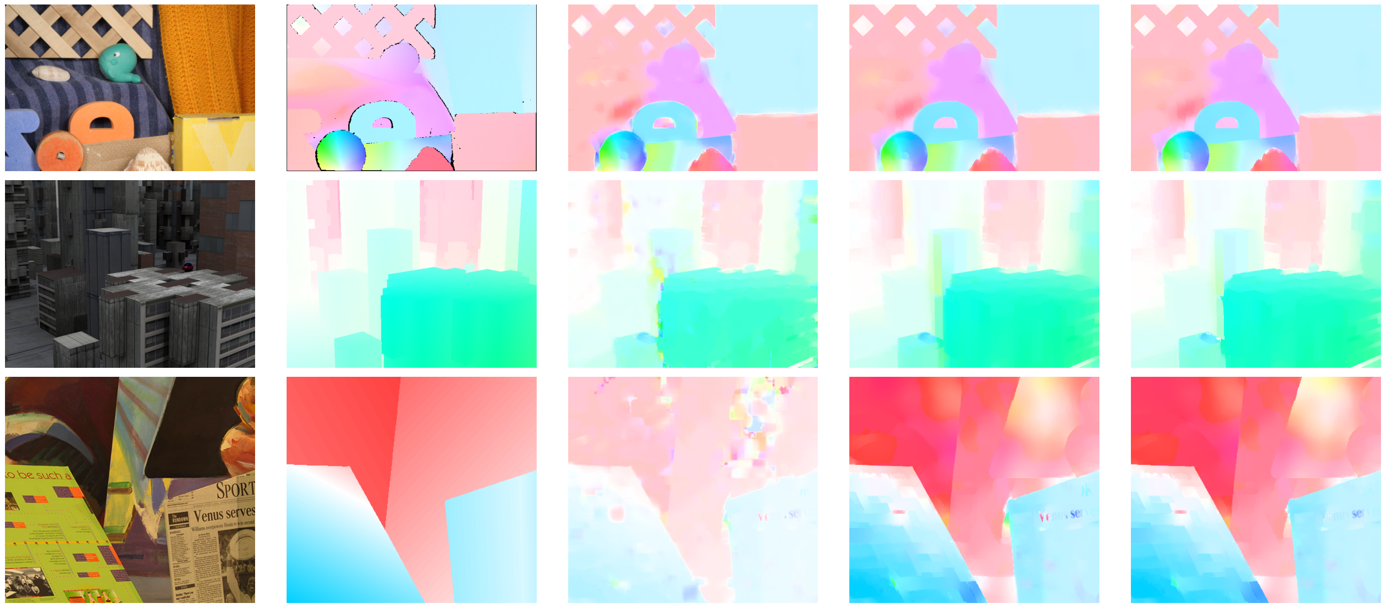


Figure 4.3: Examples from the Middlebury framework - Left column: Reference Image, 2nd Column: Ground Truth, Last Three Columns: SimpleFlow Result, Domain Transform applied to Simple Flow and with occlusion handling added

Set	Average Angular Error			Average Endpoint Error		
	SF	DT	OCC	SF	DT	OCC
RubberWhale (T)	7.61	7.54	7.45	0.27	0.26	0.26
Urban2 (T)	8.49	8.03	7.87	0.89	0.83	0.82
Venus (T)	10.24	7.21	6.88	0.80	0.60	0.57
Army (M)	6.48	3.98	3.57	0.17	0.09	0.06
Urban3 (M)	16.97	15.99	15.21	2.00	1.84	1.73

Table 4.1: Numerical results for (T)wo & (M)ulti Frames. SF - SimpleFlow, DT - Domain Transform, OCC - DT & occlusions

the endpoint error, e_{err} is defined as:

$$\sqrt{(u - u_{GT})^2 + (v - v_{GT})^2} \tag{4.10}$$

As the measure operates on a single pair of flow vectors, to assess overall performance, the measures of all vectors are computed, and the average is taken, and this is also commonly used [93]. Table 4.1 illustrates numerical results for our tests, including both the two image and the multi-frame case (discussed later). For the multi-frame case, we use the flow vectors between frames 4 and 5.

Referencing Table 4.1, for "RubberWhale", results are almost the same, as the flows between the three methods appear to be the same, with the exception of edges of the circular object and half-moon shape. The same can be said for "Urban2". Spurious flow vectors have been effectively filtered out due to the Domain Transform, and even more so with the added occlusion term. The most dramatic

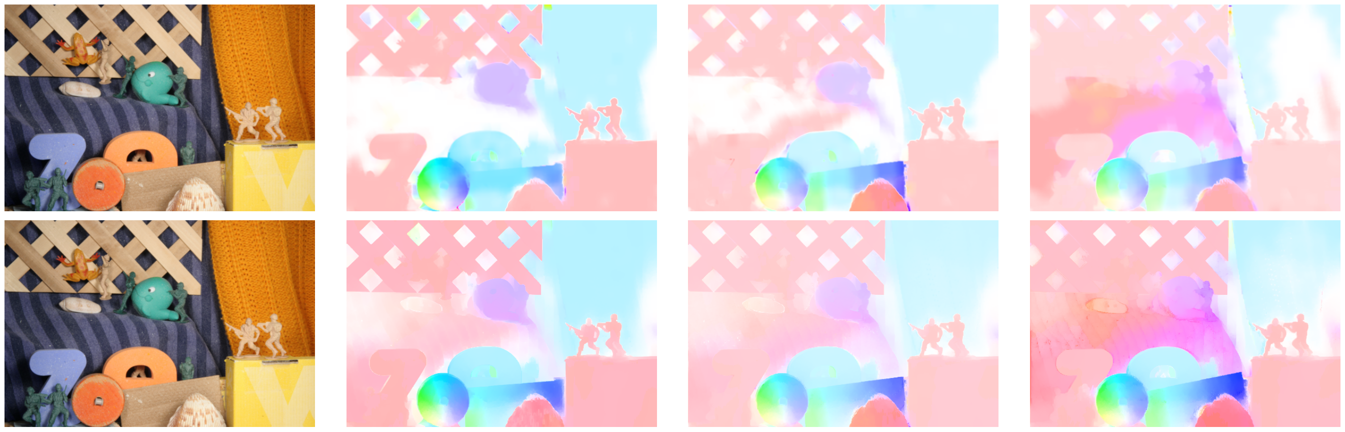


Figure 4.4: Multi-frame results: "Army". Left column - Frames 4 & 5. 2nd to last column: 1st row - SimpleFlow results for frames 1 & 2, 4 & 5 and 7 & 8. 2nd row - Results using proposed framework, with same ordering as the first row

difference is with "Venus". The errors are quite significant in the angular and endpoint error, and can be verified through Fig. 4.3.

For the case of multi-frame case, we show two examples using multi-frame datasets from the Middlebury database. Fig. 4.4 illustrates the first example, which is an 8 frame sequence known as "Army". The first column illustrates two reference frames - the fourth and fifth frame. From the second column onwards, the first row shows the results from SimpleFlow for frames 1 & 2, 4 & 5, and 7 & 8. while the second row illustrates the results in the same order, using our filtering strategy *with* temporal filtering and occlusion handling together.

For SimpleFlow, the flows are *not* temporally consistent, nor are they smooth. Also, the flows of the background change wildly over the sequence. In addition, around certain edges, specifically the main objects near the front of the camera, become obscured or are inaccurate. This is most likely attributed to the occlusions that appear near the edges. This is evident in the third column, around most of the edges of the objects. What is interesting in the last column is that the flows for the uniform background on the very right are missing in the SimpleFlow method, but are filled in by our proposed framework. This verifies that the temporal consistency, as well as the use of guidance images helps improve the accuracy of the flow maps. By referencing Table 4.1, this most likely contributes to why the angular and endpoint errors are significantly lower than SimpleFlow. Finally, we show Fig. 4.5, where results are in the same style as Fig. 4.4. This is "Urban3", and is also an 8 frame sequence. Like in Fig. 4.4, the SimpleFlow results are not temporally consistent. Spurious flow vector estimations can be seen, denoted as the red and green areas appearing in random places. In addition, information around the edges are obscured, particularly the moving elevator and its surroundings. With the proposed approach, the spurious pixels disappear with the aid of the guidance images, and the edges are more preserved. The motion seems smoother between frames, and there were spurious flow vectors filtered out. These observations are verified by referring to Table 4.1.

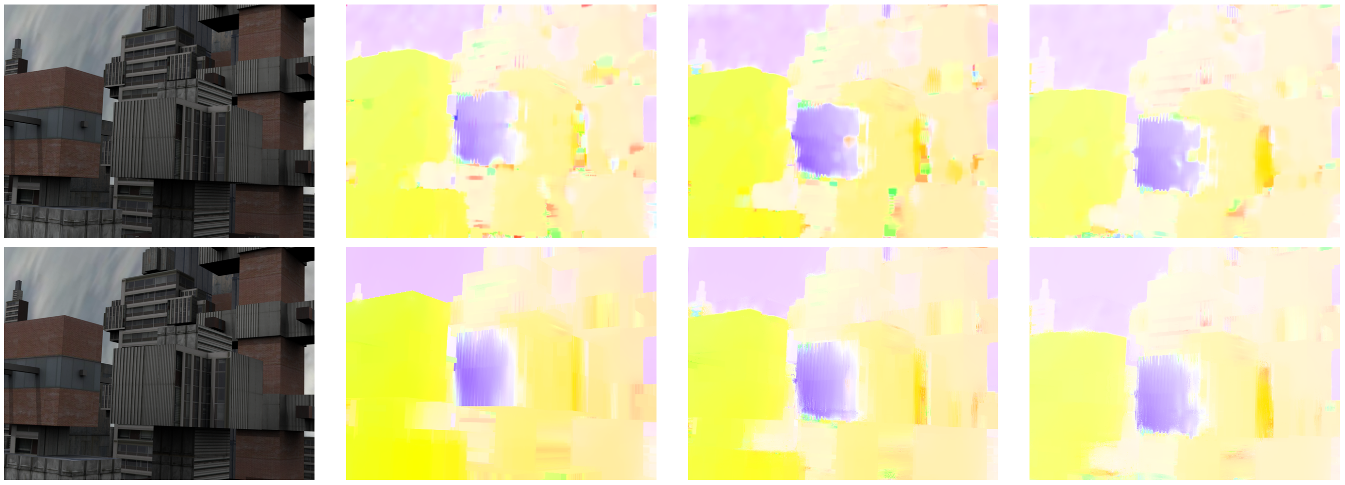


Figure 4.5: Multi-frame results: "Urban3". Left column - Frames 4 & 5. 2nd to last column: 1st row - SimpleFlow results for frames 1 & 2, 4 & 5 and 7 & 8. 2nd row - Results using proposed framework, with same ordering as the first row

4.4 Label Tracking Results

For the sake of completeness, and as seen with the TLD tracker, we present an example of how this is used for label tracking. As this method is only applicable for scenes that exhibit horizontal motion, care should be employed so that this is only selected when such a case arises. To complete the discussion of the proposed optical flow framework, Fig. 4.6 denotes how the proposed optical flow algorithm can be used for the process of depth label propagation. As said previously, any pixels in the frame that coincide with a depth stroke can simply be translated over to the next frame by the optical flow vector defined at that point.

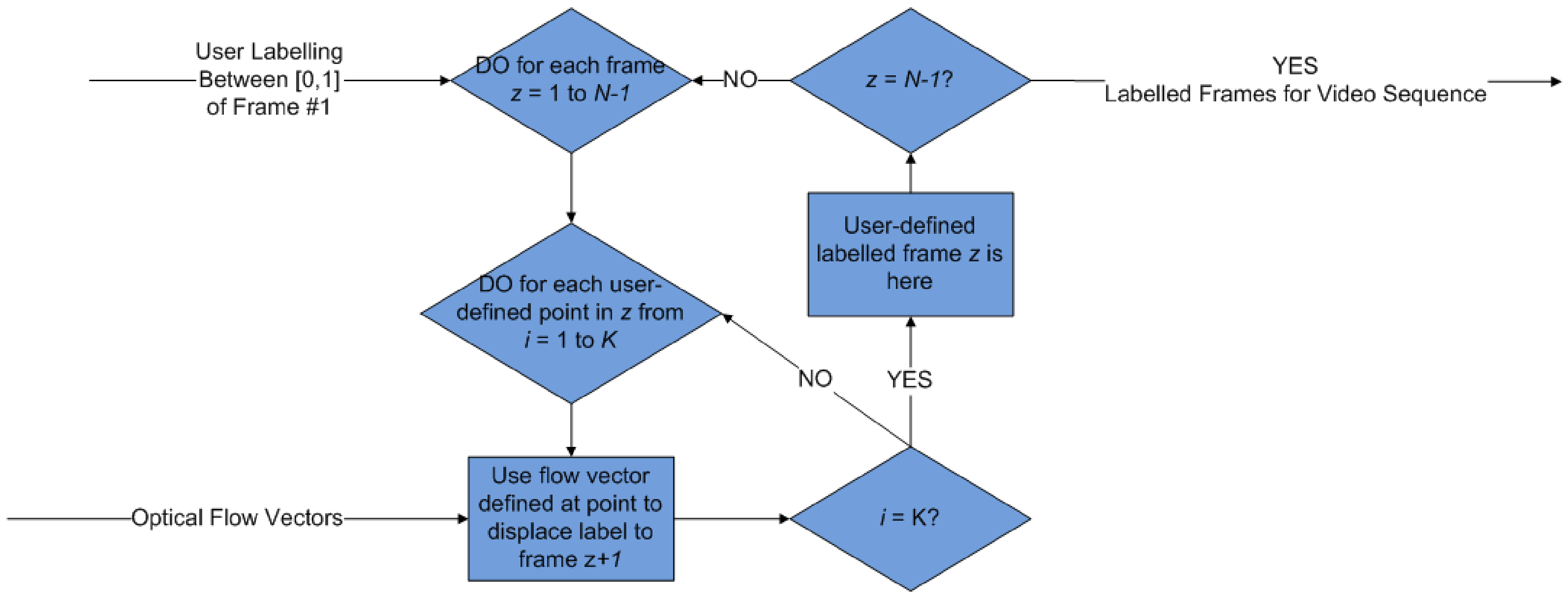

User Labelling Between [0,1] of Frame #1
DO for each frame z = 1 to N-1
NO
z = N-1?
YES Labelled Frames for Video Sequence
User-defined labelled frame z is here
DO for each user-defined point in z from i = 1 to K
NO
YES
Optical Flow Vectors
Use flow vector defined at point to displace label to frame z+1
i = K?

Figure 4.6: A block diagram of the optical flow framework for depth label tracking

To illustrate where the optical flow framework would be ideal, a shot from the *Big Buck Bunny* sequence was used, as it is an ideal candidate for tracking when only subject to horizontal motion. Specifically, these are the objects that are travelling on the ground, and hiding in the trees. Fig. 4.7(a) shows the first frame, with its user-defined labellings, and Figs. 4.7(b)-(f) illustrate the labelled frames from the optical flow algorithm, specifically for frames 11, 22, 38, 39 and 41. As can be seen in the figure, the optical flow framework performs quite well, and manages to capture both slow motion - the object on the ground, and fast motion - the object in the trees, due to the multi-resolution approach that SimpleFlow employs. In addition, the rest of the labels remain stationary, as they exhibit no motion, and is expected with respect to optical flow. As with the previous chapter, the actual depth maps and stereoscopic visualizations will be shown in the Results chapter in Chapter 5.

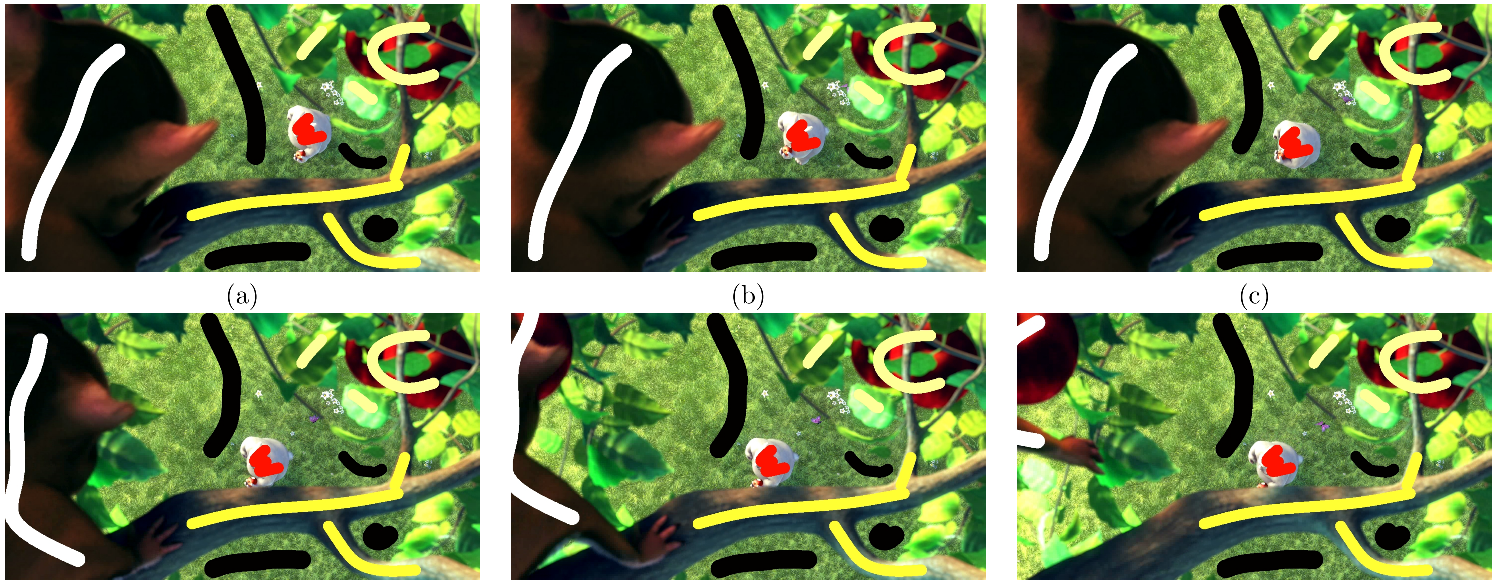


Figure 4.7: Big Buck Bunny sequence - (a) User-defined labels for the first frame, (b) Frame 11, (c) Frame 22, (d) Frame 38, (e) Frame 39, (f) Frame 41

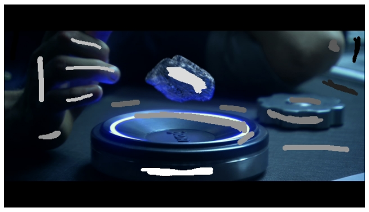
Chapter 5

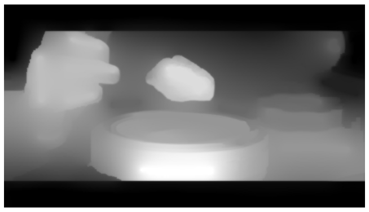
Results & Discussion

IN this chapter, we will demonstrate depth map results using our conversion framework on image and single-view video sequences. For images, these are only shown for demonstration purposes, as the user has the option of converting solely images. We will show some depth map results using a variety of different media, such as cartoon images, artwork and real scenes, as well as their corresponding anaglyph images so that the reader has a sense of how the conversion framework works for images in practice. However, the main focus of our work is converting video sequences, and will be shown after the image conversion results. With the video results, to provide a benchmark with the proposed work, this thesis compares the proposed work with an implementation of Guttmann *et al.*'s framework [1]. In the proposed framework, it is assumed that the video sequence is decomposed into shots, and the shots are converted separately. These shots can easily be found by any conventional shot detection algorithm. Because there are various components within the system, each with their own unique set of parameters, and were also discussed in the previous chapters, in order to be consistent, those same parameters are maintained for the results shown in this chapter.

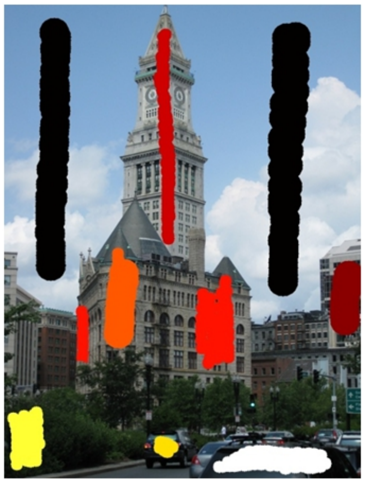
5.1 Image Conversion Results

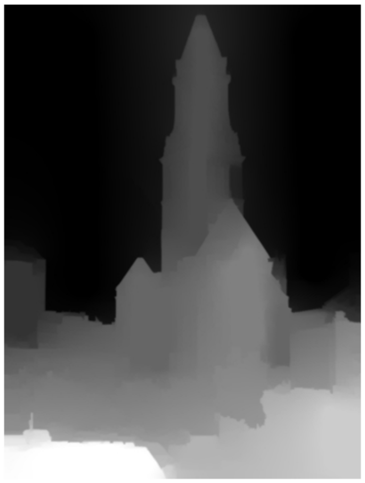
Figs. 5.1, 5.2 and 5.3 show a variety of examples using images from all areas of interest. For each figure, the first column shows the original images themselves, with their user-defined depth labels overlaid, the second column shows the depth maps for each of the images shown, and the last column shows the anaglyph images, illustrating the stereoscopic versions of the single-view images. The anaglyph images are created so that the left view is given a red hue, while the right view is given a cyan hue. As such, the left eye should correspond to the red filter, while the right eye corresponds to the cyan filter. To create the stereoscopic image, the left view serves as the original image, while the right view was created using simple Depth Image Based Rendering (DIBR). After, to perform DIBR, the depths are used as disparities, and serve as translating each pixel in the original image over to the left by a certain amount. Here, we introduce a multiplicative factor, s, for each value in the depth map, and introduce a bias, b, to


(a) Avatar

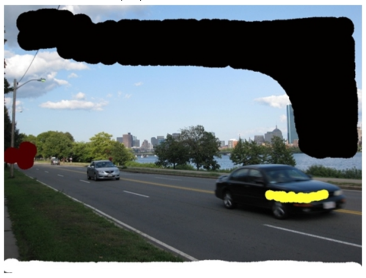

(b) Avatar Depth Map

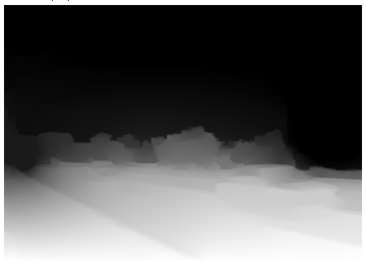
(c) Avatar Anaglyph


(d) Boston1


(e) Boston1 Depth Map

(f) Boston1 Anaglyph


(g) Boston2


(h) Boston2 Depth Map

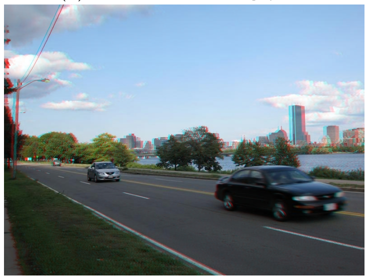
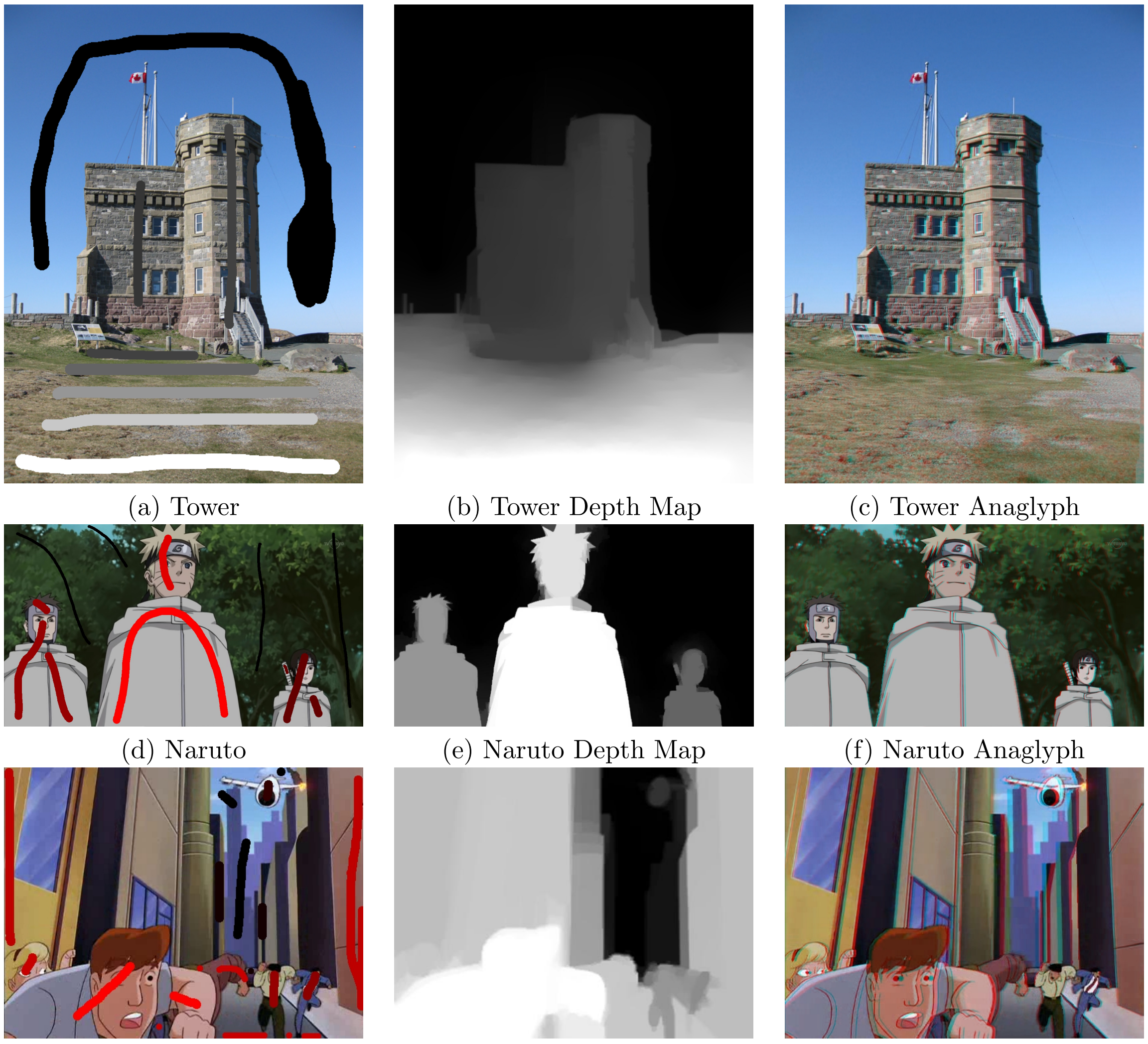

(i) Boston2 Anaglyph

Figure 5.1: The first set of conversion examples using various images from different sources of media. Left Column - Original images with depth labels overlaid. Middle Column - Depth maps for each image. Right Column - Resulting anaglyph images

adjust where the convergence occurs in the image. The purpose of s is to adjust the disparities for the particular viewing display of interest. The actual depths in the depth maps are normalized in the range of $[0, 1]$, and we render the right view using the aforementioned process. Given a value in the original depth map, D_O, the equation used to determine the shift, F_S, of each pixel from the left view, to its target in the right, is the following:

$$F_S = sD_O + b \quad . \tag{5.1}$$

In all of our rendered images, we chose $s = 30$, and $b = -15$ so that the disparities can clearly be



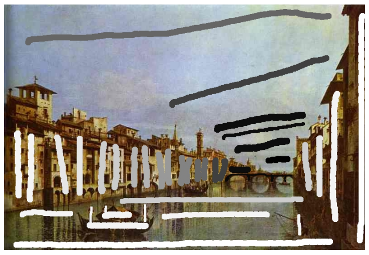
(a) Tower (b) Tower Depth Map (c) Tower Anaglyph

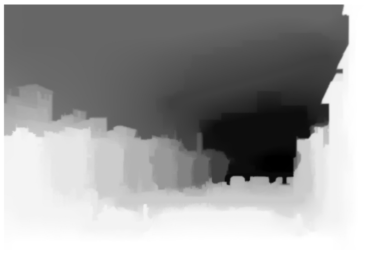
(d) Naruto (e) Naruto Depth Map (f) Naruto Anaglyph

(g) Superman (h) Superman Depth Map (i) Superman Anaglyph

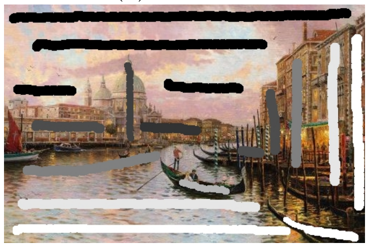
Figure 5.2: The second set of conversion examples using various images from different sources of media. Left Column - Original images with depth labels overlaid. Middle Column - Depth maps for each image. Right Column - Resulting anaglyph images

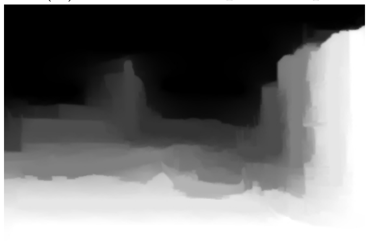
seen. In addition, we request that the reader zoom into the figures for the fullest stereoscopic effect. To demonstrate the full effect of labelling with different intensities or different colours, the figures vary in their style of labelling, and use either: (a) A grayscale range, where black denotes far pixels, and white denotes closer pixels, (b) Using only the red channel, where dark red denotes far pixels, and bright red denotes closer pixels and (c) Varying from black and red, to yellow and white. Dark colours, such as black and red, denote far pixels, while bright colours, such as yellow and white, denote closer pixels. After performing DIBR, disocclusions, or holes, will inevitably occur. These holes were filled using the SSRW framework to simulate anisotropic diffusion. To summarize, for the regions that are occluded, they are


(a) Florence


(b) Florence Depth Map

(c) Florence Anaglyph


(d) Venice


(e) Venice Depth Map

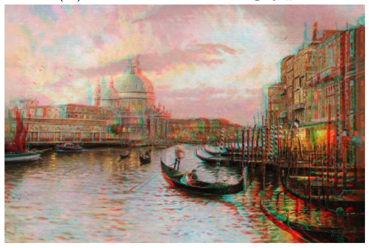
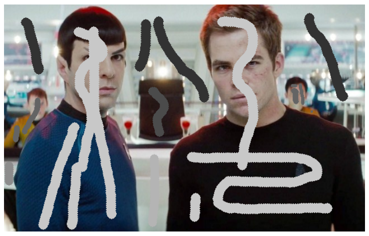

(f) Venice Anaglyph

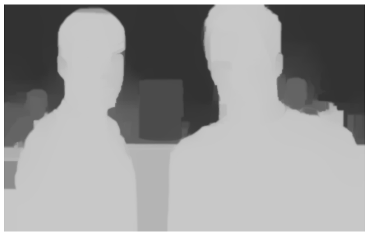
Figure 5.3: The third set of conversion examples using various images from different sources of media. Left Column - Original images with depth labels overlaid. Middle Column - Depth maps for each image. Right Column - Resulting anaglyph images

separated into their individual colour channels, and their gray-level intensities are used as "user"-defined strokes. The results from all three channels are combined using each triplet of gray-level intensities at each location to convert them to their respective colour values.

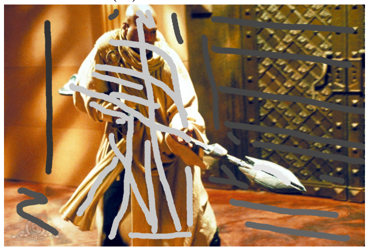
Examining Fig. 5.1, Fig. 5.1(a) shows a frame of the Avatar movie, while Figs. 5.1(b) and 5.1(c) illustrate the depth map and the stereostopic version in anaglyph format. The scene requires very few strokes in order to achieve good depth perception, as evidenced by the depth map. The edges around the prominent objects, such as the hand, the levitating platform, and the rock are very smooth, which will minimize the effect of "cardboard cutouts" upon perception on stereoscopic displays. Also, along the ground plane, there is a smooth transition near the front of the camera, all the way to the back as expected. The same can be said with Figs.5.1(d)-5.1(f) and Figs. 5.1(g)-5.1(i). These are images taken from downtown Boston, and only a few strokes were required, so long as there are some dark strokes and some bright strokes, to achieve a good quality depth map. Like the Avatar image, there are smooth gradients from the front of the camera to the back, and the edges are soft so that perception on stereoscopic displays will be comfortable. The images of Fig. 5.1 have also been chosen on purpose, as there are prominent objects in the foreground, while there are other elements that belong to the background. By assigning these objects to have depths that are closer to the camera, while the background has depths assigned to those that are farther, the depth map will naturally embed this information into the result.

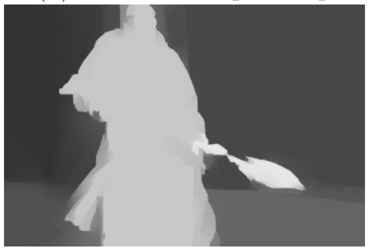
Examining Fig. 5.2, the style of presentation is exactly the same as Fig. 5.1. In the first row, Figs. 5.2(a)-5.2(c) illustrates an outdoor scene, that is similar what was seen of Boston - there are dominant objects (in this case, the tower is the only object), and the goal is to make the tower perceptually appear


(a) Star Trek

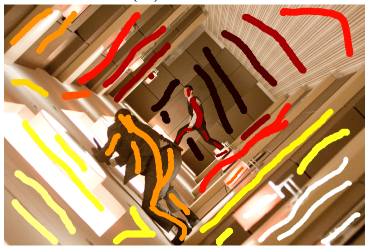

(b) Star Trek Depth Map

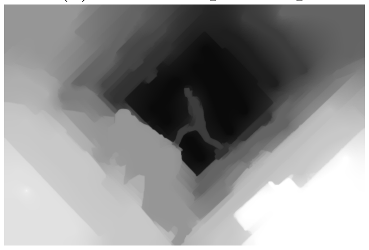
(c) Star Trek Anaglyph


(d) Teal'c


(e) Teal'c Depth Map

(f) Teal'c Anaglyph


(f) Inception


(g) Inception Depth Map

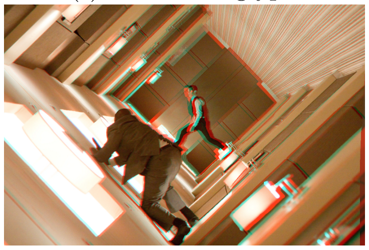

(h) Inception Anaglyph

Figure 5.4: The fourth set of conversion examples using various images from different sources of media. Left Column - Original images with depth labels overlaid. Middle Column - Depth maps for each image. Right Column - Resulting anaglyph images

to be more forward than the background. The depth map is a very good representation of how an individual would perceive the scene to be. The last two rows show examples of converting cartoons, demonstrating that our framework is limited to just scenes captured by a camera. Figs. 5.2(d)-5.1(f) and Figs. 5.1(g)-5.1(i) show a frame from the Naruto Japanese Anime, and a frame from Superman: The Animated Series. They both share the fact that they require very few strokes to obtain a good depth map, and ultimately a stereoscopic image obtaining good depth perception.

Adding to these results and by examining results in this framework in an entirely different realm, Fig. 5.3 shows examples using paintings and artwork. Specifically, Fig. 5.3(a) is a picture of Bernardo Bellotto's Florence, and Fig. 5.1(d) is a picture of Thomas Kinkade's Venice. With these images, they required more strokes to obtain a good depth perception, as there are many soft edges around the objects. As depth estimation is considered as a multi-label segmentation, objects that are classified as a certain depth may have their surroundings classified as the same depth due to their edges blending into the background. As such, extra strokes are required to be placed around the objects to minimize this "leaking". Nevertheless, the depth maps that are generated show a good estimation of how a human would perceive the depth in the scene.

Finally, Fig. 5.4 shows example shots from motion pictures, in the same style as Fig. 5.1. In the first row, Fig. 5.4(a) is a promotional shot for the re-invented Star Trek Movie (2009). With this shot, only a few strokes were required, as the objects had crisp boundaries in comparison to the background, with the resulting depth map in Fig. 5.4(b), this also shows a good representation of how a human would perceive the scene in terms of depth. Fig. 5.4(d) is a shot of Teal'c, a fictional character from the Stargate SG-1 science fiction universe. What is interesting about this shot is that there is considerable motion blur in the background. In addition, the staff weapon of the character is slightly pointing to the camera, meaning that some of the weapon will appear closer to the camera, while other parts are farther. This was taken into account when the depth strokes were brushed onto the image. Referencing Fig. 5.4(e), again the depth map generated is also a good representation of how a human would perceive the scene. For the final image, Fig. 5.4(f) is a shot from the motion picture *Inception*, and is one of the most publicized shots concerning this movie. This shot is most interesting, as the scene is on an incline, with multiple objects appearing at multiple depths. As such, this requires more strokes than what was seen previously. Nevertheless, Fig. 5.4(g) illustrates a very good reconstruction of how a human would perceive the scene. There are smooth depth gradients that originate from near the camera, outwards towards the back of the room.

As a supplement to the anaglyphs seen in this section, to further appreciate the results, Figs. 5.5 to 5.11 show higher resolution anaglyphs of the results previously discussed. Due to the fact that the *Inception* image is high resolution, the image is shown on a separate figure to ensure the best depth perception.

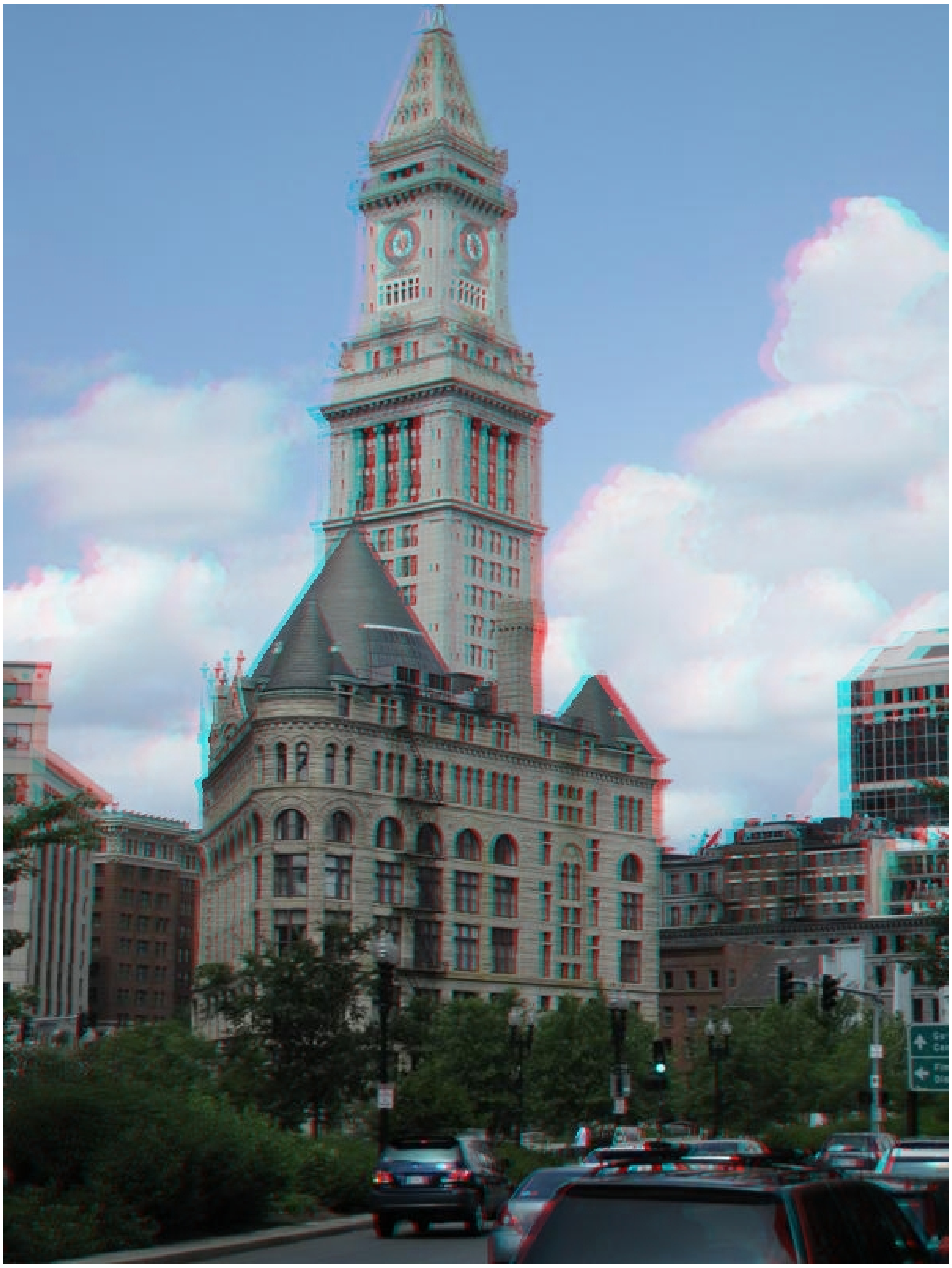


Figure 5.5: Anaglyphs discussed in Chapter 5.1

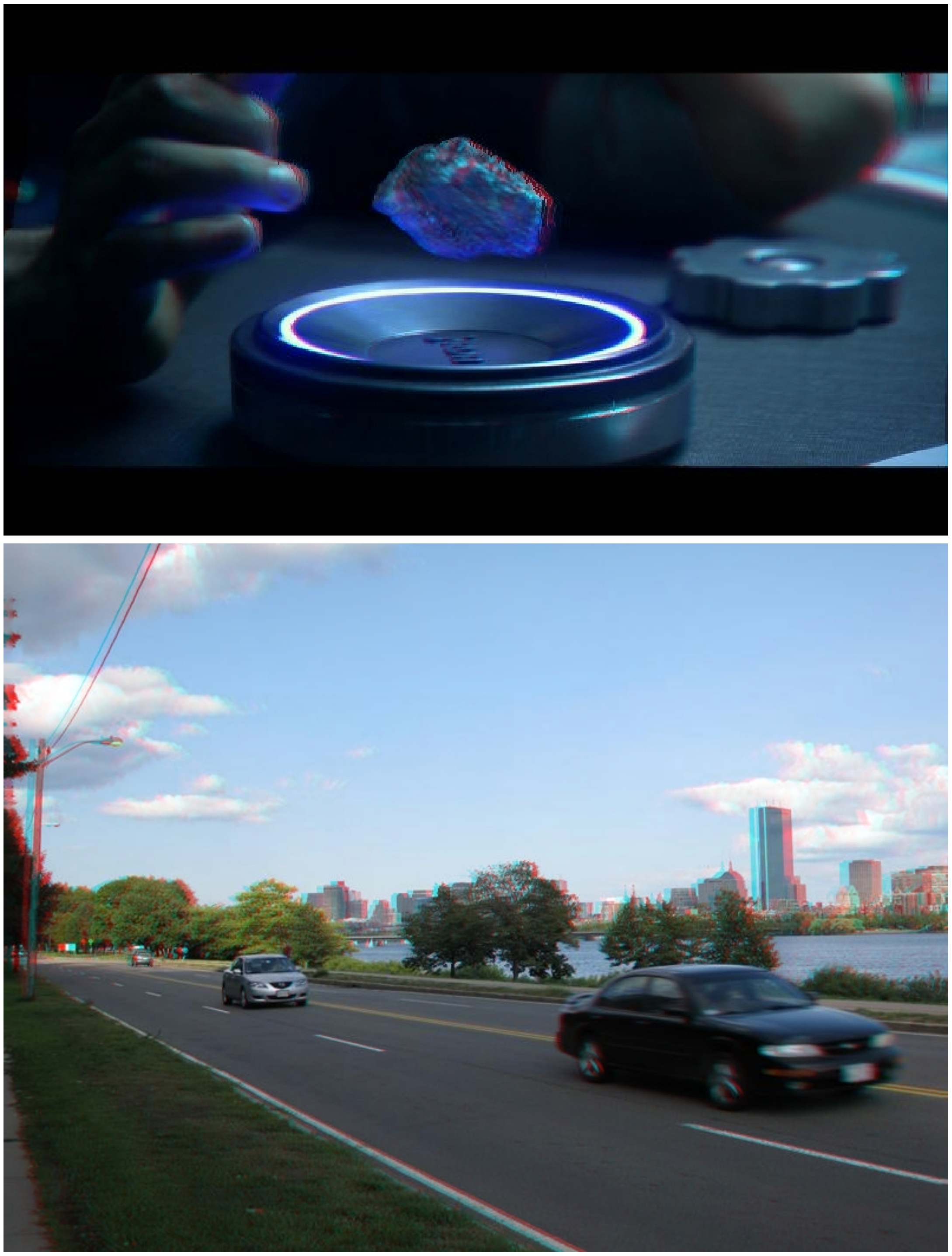


Figure 5.6: Anaglyphs discussed in Chapter 5.1

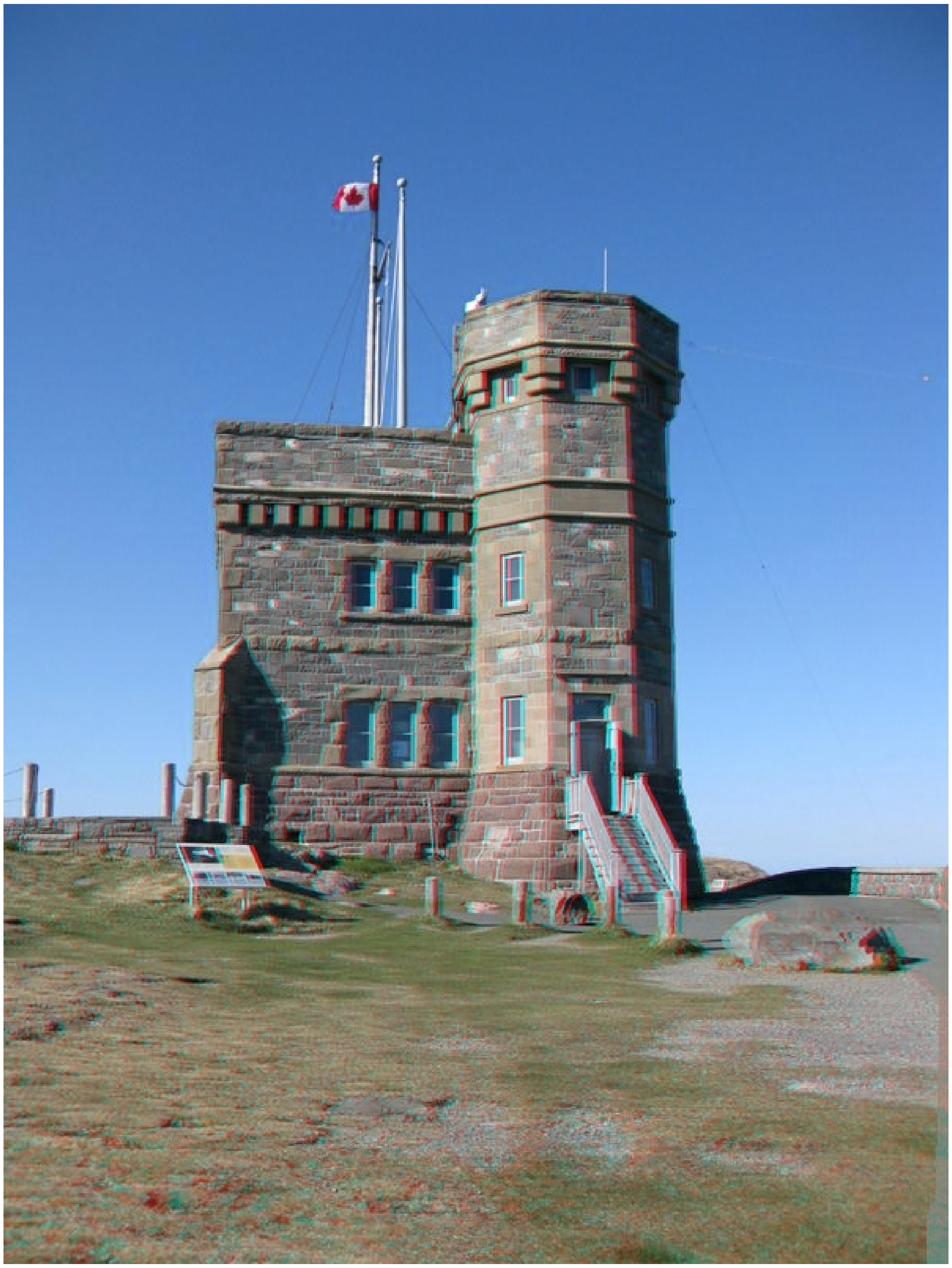


Figure 5.7: Anaglyphs discussed in Chapter 5.1

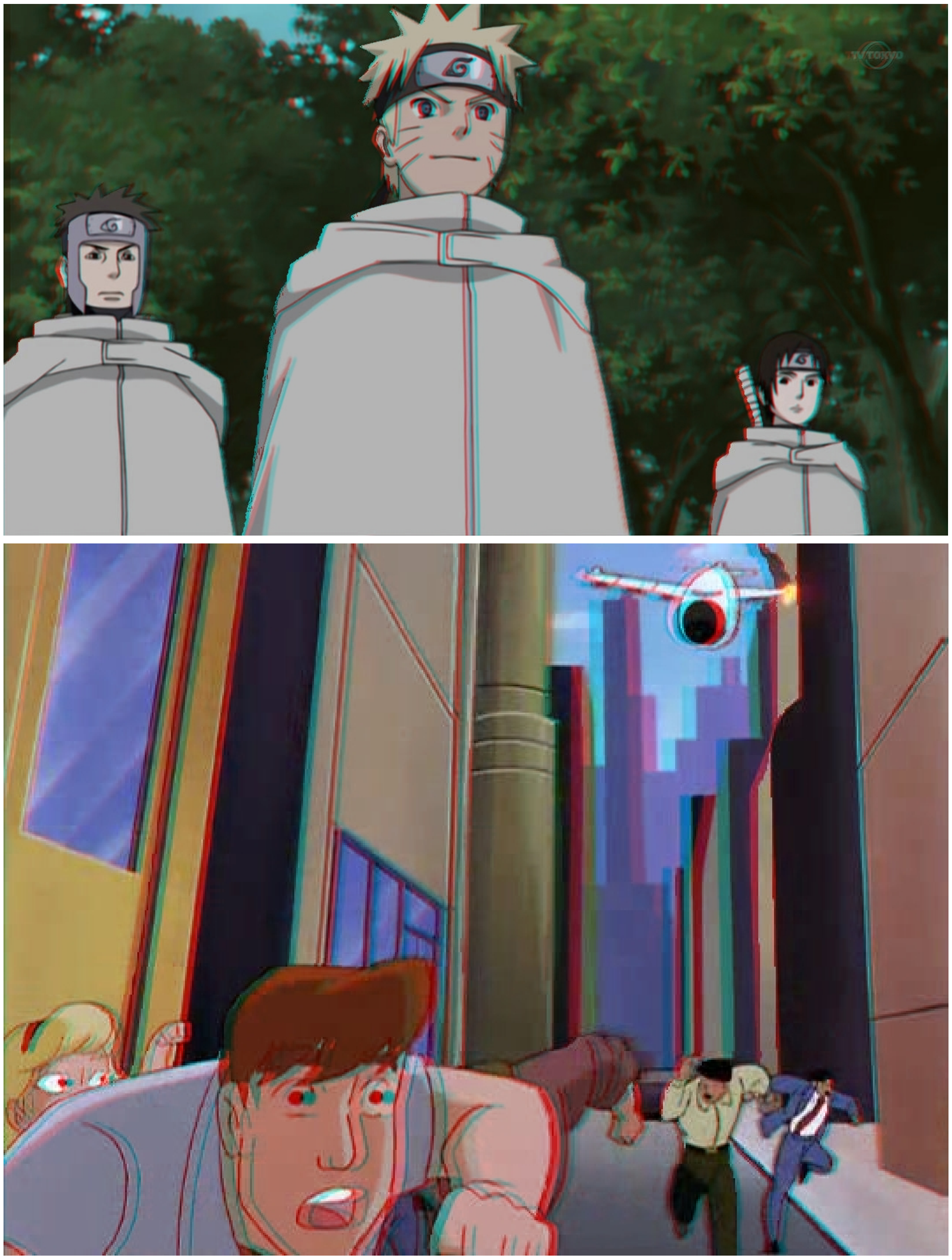


Figure 5.8: Anaglyphs discussed in Chapter 5.1

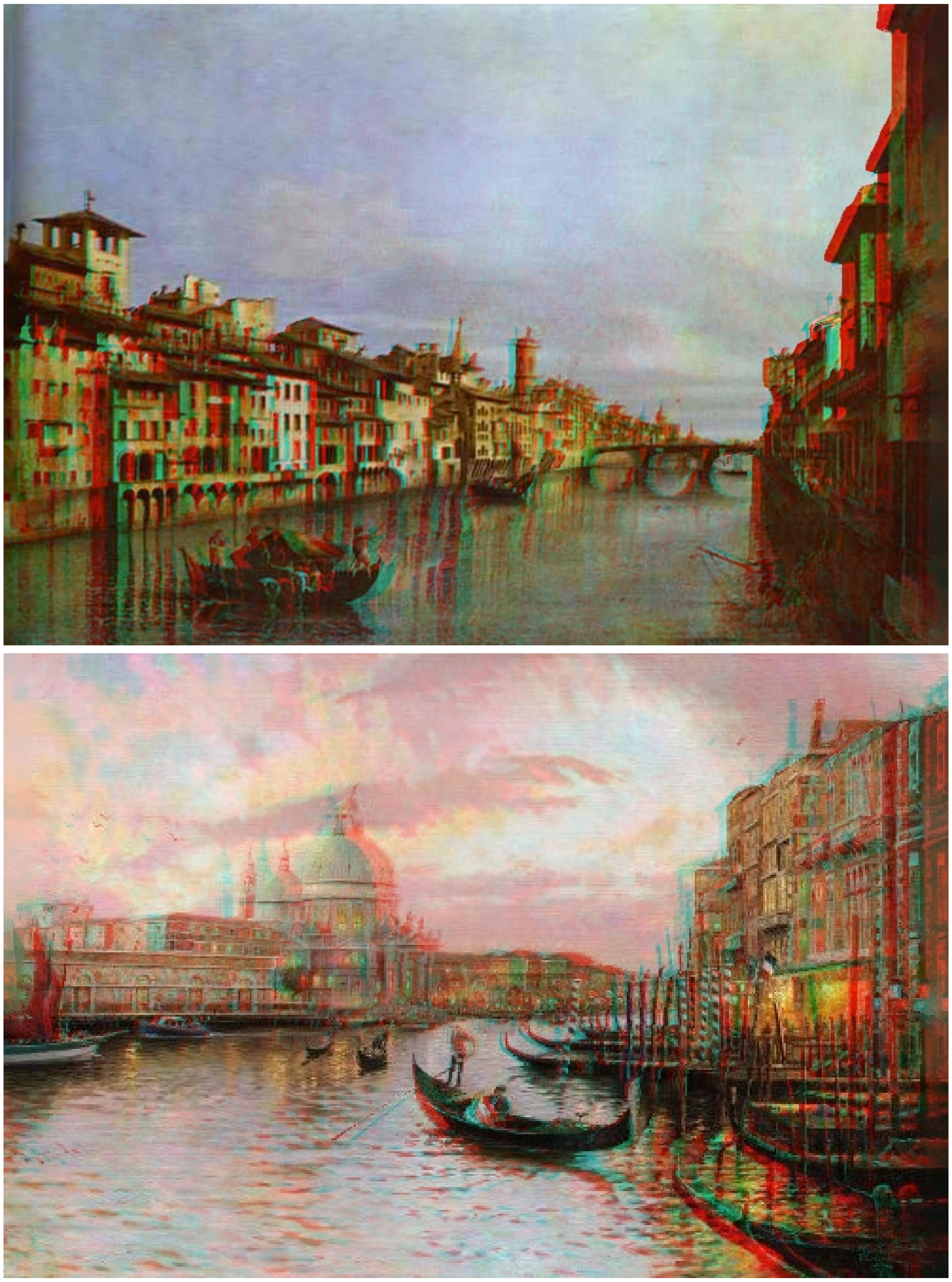


Figure 5.9: Anaglyphs discussed in Chapter 5.1

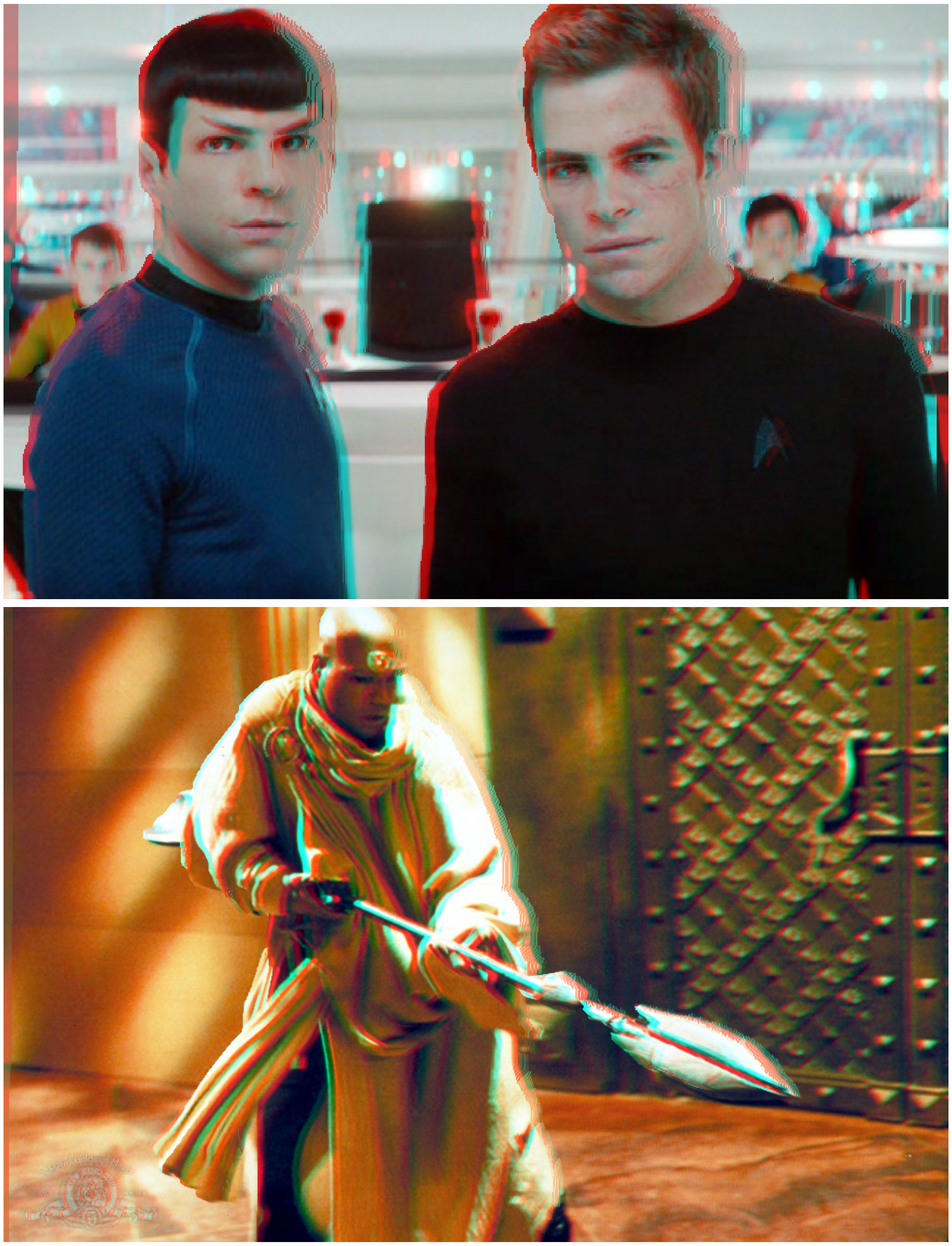


Figure 5.10: Anaglyphs discussed in Chapter 5.1

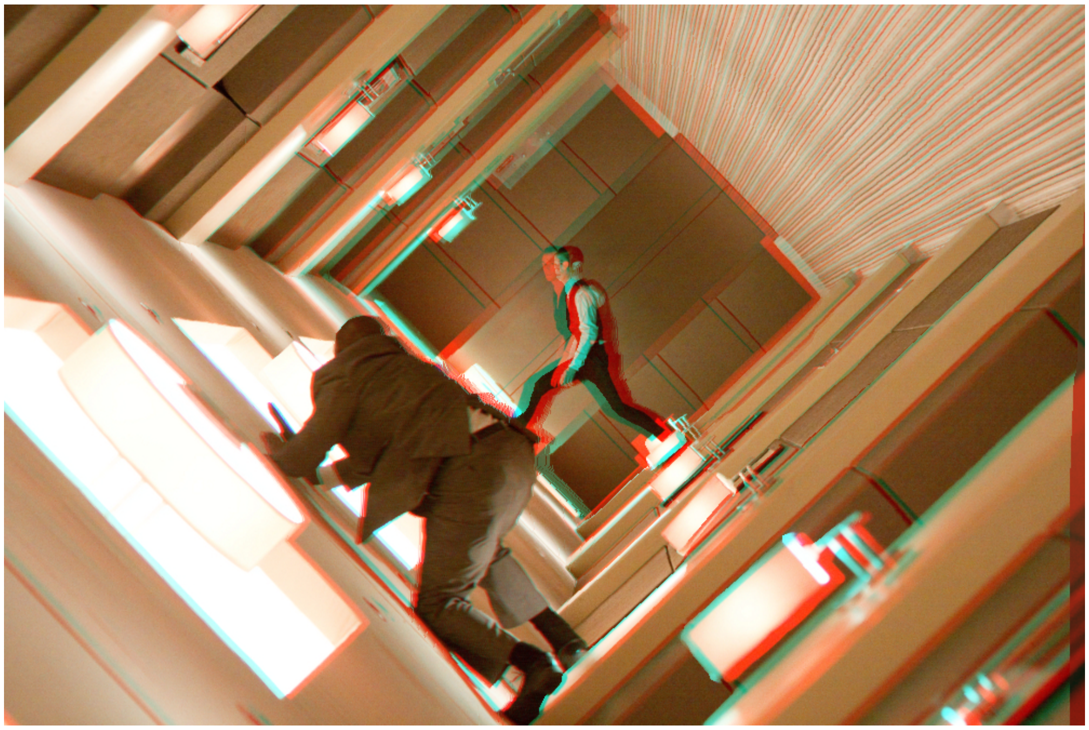


Figure 5.11: Inception Anaglyph discussed in Chapter 5.1

5.2 Video Conversion Results

In this section, we show results on single-view footage taken from a variety of different sources, and we compare to Guttmann *et al.*'s work, showing their depth map and anaglyph images on the same videos, using the same user-defined strokes to be consistent in comparison. To ensure proper compatibility, Guttmann *et al.*'s framework only marks the first and last frames of the video sequence. We maintain the fact that *all* video frames in the sequence need to be marked to ensure the highest quality possible, and so the proposed framework will demonstrate results using this fact. In order to be consistent, the depth map for the last frame that is generated by the framework in [1] is created using the user-defined labeling, while for our framework, we use the labels for the last frame generated by our label tracking framework. Certain scenes require a different label tracking algorithm. For those scenes where the objects move perpendicular to the camera axis, the computer vision tracking algorithm is employed, whereas for those scenes where the position of the objects remain unchanged, and motion is only exhibited horizontally (i.e. motion that is parallel and not perpendicular), the Optical Flow method is used instead. The result of the system generates a set of depth maps - one per frame - that is used to render each frame on stereoscopic viewing hardware. As with Section 5.1, when the artificial right views are created, the same hole filling technique is employed to eliminate disocclusions that may appear.

We illustrate six examples of video shots from all sorts of media, while we leave the next section (Section 5.5) to illustrate subjective results, showing some conversion results to test subjects. We begin our results by using one of the test sets that are readily available on the Middlebury Optical Flow database[1], known as "army", which is a small eight frame sequence. Though this sequence is rather small, it illustrates our point that the proposed framework generates better stereoscopic views for user digest in comparison to the framework in [1]. The edges and objects are very well defined, and if a conversion framework does not perform well here, then the framework will most likely not work with other video sequences. In addition, this sequence is a good candidate for using the Optical Flow based label tracking method, as the objects only exhibit horizontal motion in the sequence (see Chapter 4). As such, Fig. 5.12 illustrates a conversion result using this dataset. The first row shows the original first and last frames of the sequence, and the second row shows user-defined labels for the first and last frame. The labels for the label tracking result of the last frame are not shown for brevity. The quality of this result will be seen for the generated depth maps. The third row illustrates a sample depth map in the middle of the video sequence. The left image shows the depth map using the framework in [1], while the right image is the result using our proposed method. Finally, the last row illustrates anaglyph images, showing both the left and right views, presented in the same order as the third row. The left image is the same in both, while the right images are the one used from their respective frameworks. As seen previously, these require red/cyan anaglyph glasses to view the results.

As can be seen in the results, the depth map that Guttmann *et al.*'s framework generated does not agree perceptually with the one generated by the proposed framework. In addition, there are various black "spots" seen in the map, most likely due to the misclassification of the anchor points performed in the SVM stage of their proposed framework. There is also a noticeable streaking, most likely due to the fact that the edges in the frame are one of the factors that are used when solving for the depths. If the definition of the edges are improper, then the results will also be poor. Though some areas of the depth map generated by using the framework in [1], other areas tend to "bleed" into neighbouring areas, as the edges are relatively weak, and the definition of the edge detection is not sufficient enough to capture these edges. Our framework not only respects the edges of each object, but it allows some internal variation within the objects to minimize the cardboard cutout effect. Fig. 5.13 illustrates the next conversion result comparing both frameworks, which is a 56 frame sequence known as the Shell-Ferrari-Partizan (SFP) sequence. This video shot is of particular interest, as the object of focus (the Formula 1 race car) begins in the shot to be very close to the camera, and eventually moves away from the camera towards the end of the shot. This scene is a very good candidate for using our motion-based label tracking approach (see Chapter 3), as the depths should dynamically adjust themselves for the object, as the object moves away from the camera. The style of presentation is the same as Fig. 5.12

Again, with the observations seen in Fig. 5.12, there are misclassifications or "spots" seen in the depth maps generated by Guttmann *et al.*'s framework, and noticeable streaking in areas where there are no strong edges. Not only does our framework respect strong edges, but the depths of the car dynamically change as the car moves away from the camera. We show two more results, shown in Figs. 5.14 and 5.15. Fig. 5.14 is a clip from the "Sintel" sequence, seen previously in Chapter 2.6.1, but in this instance, Guttmann *et al.*'s framework will be used to compare with our framework. Also, the sequence

[1]`http://vision.middlebury.edu/flow/`

also exhibits motion that is parallel, and no objects are moving perpendicular to the camera axis - a good candidate for the Optical Flow based label tracking method that we propose (see Chapter 3.3). Fig. 5.15 is a clip from the Superman: The Animated Series cartoon, which shows a flying ship starting from the top of the frame, and moving towards the bottom of the frame, and gets closer to the viewer - again, a good candidate for the motion-based label tracking method that we propose (see Chapter 4). In addition, to show the flexibility of our system, we use a coloured label scheme, much like the one seen in Chapter 2. As can be seen in both the figures, the same observations can be said regarding respecting the edges of objects (Fig. 5.14 and the dynamic adjustment of the depths (Fig. 5.15).

Fig. 5.16 shows a shot from the television series "24", with the fictional character Jack Bauer as the main point of interest in the shot. This shot also exhibits purely horizontal motion, and so this would be a good candidate for use with the optical flow algorithm. Specifically, the character is moving towards the left, then points his gun to the camera. With respect to the depth map generated by the proposed framework, and despite the fact that there is some motion blur in the background, the depth map (which was the 18^{th} frame chose in this sequence, and was towards the middle of the shot) manages to capture unique depths for each of the objects, as well as several planes of depth when examining the bottom half of the image. There are various walls, buildings and structures that are supposed to have their own unique depths, which the algorithm managed to capture successfully. Unfortunately, the depth map generated by Guttmann *et al.*'s work failed completely, most likely due to the misclassification of depth points given by the anchor point detection stage. In addition, this sequence is of relatively low resolution (492 x 360), and so there may be features in the frame that are not worthy candidates to be assigned to SIFT points. Finally, Fig. 5.17 shows a shot from the television series *Game of Thrones*, with the fictional character Daenerys Targaryen as the main point of interest in the shot. The head of the character exhibits some motion that is perpendicular to the camera axis, and so this would be a good candidate for use with the modified TLD algorithm. With respect to the depth map generated by the proposed work, this again manages to capture the unique depths of each object appearing in the scene, and the spears appearing in the frame are blurry due to them being out of focus, and focusing on the fictional character. This is not the case when examining the depth map generated by Guttmann *et al.*'s work. As seen previously, due to the improper edge definition, regions of depth leak onto other coherent areas, and is not a good representation of the depth experienced in the scene.

As a supplement to the anaglyphs seen in this section, to further appreciate the results, Figs. 5.18 to 5.23 show higher resolution anaglyphs of the results previously discussed. As we are comparing to Guttmann *et al.*'s work, we also show their anaglyphs in high resolution to further exemplify the merits of the proposed work.

5.3 Execution Time

The overall time taken to convert the images is dependent on the complexity of the scene, the size of the image, and the number of user-defined labels placed on the image. For images having a significant amount of depth content, this requires more depth labellings. On average, it takes between 5 seconds to 45 seconds to label the frame, while roughly a couple of seconds to generate the depth map. These experiments were all performed on an Intel Quad 2 Core Q6600 2.4 GHz system, with 8 GB of RAM. After, should the user be unsatisfied with the results, they simply have to observe where the errors are in the depth map, place more user-defined labels, and the algorithm can be re-run. For video however, the computational complexity has significantly increased, thus resulting in an increase in computational time. To compare our work with Guttmann *et al.*, the time required for the SVM training stage also depended on how many user-defined labels there existed. The more labels that are introduced, the more SVMs that are required for training. As such, a good majority of the computational burden was in the SVM training stage. With the labels seen in our results, they varied between 7 to 10 unique user-defined labels per frame, and SVM training took between 75 - 90 seconds on average. This also depended on the resolution of each frame. To complete the depth map estimation, creating the sparse system took 10 to 20 seconds on average. Solving the actual system varied from 75 to 85 seconds. For our proposed framework, no training was necessary, and the depths were solved for directly. The label tracking framework, which is the more computationally intensive step, took roughly 50 to 65 seconds when the modified TLD framework is used, or between 5 to 20 seconds when the optical flow framework is used. It is necessary to observe the motion of the scene and the objects before choosing the right label tracking method to use, in order to ensure the most accurate results. In terms of depth map generation, this took between 35 to 60 seconds, and varied due to the varying lengths of the sequences that were used for testing. The timing for the depth map generation and the label tracking is for the entire video volume, and is illustrative of the results shown in this thesis. Even with both of these steps, our framework is faster than Guttmann *et al.*'s framework.

What is also interesting about these timing results is that this is significantly faster to perform than with the current industry accepted standard of employing rotoscopers, where IMAX is a large proponent in endorsing this technique. On average for IMAX 3D converted films, for 20 minutes of footage to be converted into stereoscopic 3D, it requires roughly one month of work. This includes the work performed by the rotoscopers in manually cutting out the objects, as well as performing image inpainting to fill in occlusions, upsampling the frames from 35mm to 70mm film format and noise removal. One month, or roughly 30 days, roughly translates to $60 \cdot 60 \cdot 24 \cdot 30 = 2,592,000$ seconds. As there are 24 frames per second in video, this corresponds to $60 \cdot 24 \cdot 20 = 28,800$ frames. Therefore, the amount of seconds to convert per frame in the current industry accepted standard roughly corresponds to $2592000/28800 = 90$ seconds *per frame*. This is in comparison to the proposed framework, with the combined execution time of 80 to 90 seconds for the *entire sequence*. The sequences that were chosen to be shown in the previous section ranged between 30 to 60 frames, and so scaling this to sequences that are 20 minutes will take a significantly less amount of time in comparison.

To further exemplify the execution time of the entire process, Tables 5.1 and 5.2 shows the overall

Image	*User Labelling (seconds)*	*Algorithm (seconds)*
Avatar	10	1.72
Boston1	7	0.74
Boston2	5	0.91
Tower	4	0.82
Naruto	6	1.33
Superman	8	1.12
Florence	20	0.92
Venice	21	0.94
Star Trek	15	1
Teal'c	20	1.2
Inception	45	2.4

Table 5.1: Table illustrating execution times for the image results shown in the proposed framework

Movie Shot	*# of Frames*	*Tracking Method*	*User Labelling (seconds)*	*Tracking (seconds)*	*Algorithm (seconds)*
Army	8	Optical Flow	25	3.4	14.5
Shell	56	TLD	30	55.4	63.5
Sintel	38	Optical Flow	21	12.6	42.4
Superman	60	TLD	22	62.5	45.6
24	30	Optical Flow	23	19.4	33.2
Game of Thrones	40	TLD	31	45.6	72.5

Table 5.2: Table illustrating execution times for the video results shown in the proposed framework

timing required for our conversion framework at each stage of the process. As far as the user is concerned, this is comprised of the user-labelling step, and for the algorithm to compute the depth maps. For video, there is an increase in execution time due to the label tracking, and is properly illustrated in Table 5.2. Also, the amount of time taken to label the video sequence only consisted of the time required to label a single frame, as the label tracking algorithm determines the rest of the labels in the video sequence automatically, and is expected to be similar to the image case.

The tables illustrate that the more user labels that are given, the less time that is required to solve for the depth maps. This obviously does not hold true with the video case, as there are still a far greater number of pixels to classify for the depths as there are user-defined labels. In addition, the execution time increased with respect to the resolution in the image case, as well as the length of the video, for the video case (and also the resolution itself). Such examples were Inception, Game of Thrones, Sintel and Naruto. These were high definition images and videos, as well as some having complex scenes, which inevitably increased the time taken for user labelling.

5.4 Failed Cases

Though many different images and video sequences from a variety of genres, there will inevitably be situations where the proposed framework will fail. Specifically, the proposed framework will fail in situations with low contrast, or with large dynamic motion within a single shot. In this section, we illustrate cases where the proposed algorithm fails, and some possible solutions to ensure high quality results.

5.4.1 Low Contrast

When images or frames have low contrast, this means that it is quite difficult to distinguish what regions are foreground, and what regions are background. A good example would be when a picture is acquired with very poor illumination, or when the light source is very strong, and instead of poor illumination, there is an overwhelming amount. The colour distribution would indicate that most of the regions seen in the image are the same, when they should not be. Another case is where some portions of the image or frame have high contrast, but some of the background regions have insufficient lighting, or when these regions have an overbearing amount of lighting. Figs. 5.24 and 5.25 illustrates three examples where low contrast plagues the resulting depth map.

In Fig. 5.24, the first row corresponds to a good case of a low contrast image - candles illuminating a dark background. There are portions of the left side of the tall candle that fade into the background. Ideally, a cylindrical region should be delineated here that labeled at a certain depth, but due to the fact that the background colour appears on some of the candle, this was classified to be part of the background. One way to eliminate this is to either contrast enhance the image using any standard technique, like histogram equalization or contrast stretching, or by attempting to add more labels around the problematic areas. The second row corresponds to a shot from the motion picture *The Matrix*, where the fictional character Trinity is the main point of focus. The problematic areas here are particularly on the left side of the character, as her jacket blends in with the background. The resulting depth map illustrates this, where some areas of the character has their depths fade into the background. Also, as the hair of the character is black, this also fades into the background as well. To combat this, one could do selective contrast enhancement of the background, and then attempt to perform the depth map estimation again. In Fig. 5.25 illustrates a shot from the Star Trek (2009) motion picture, depicting the USS Kelvin confronting a Romulan warship. As can be seen, the area is illuminated with a strong source, and so most of the regions appear to have the same distribution of colours. Because of this, most the areas would be classified as the same area when it is not supposed to. This can be seen on the left half of the image. Even though different depth strokes were assigned to the image, the area on the left half of the image was assigned to be the same depth, but we are able to discern that there are unique structures at different depths, as well as some of the hull of the USS Kelvin. For this particular case, there is not much that can be done to solve this problem, and the only task one can do is to assign as many labels as possible to ensure a high quality result.

5.4.2 Large Dynamic Motion

Though the proposed framework employs an advanced computer vision based object tracker and a edge-aware temporally consistent optical flow method, if the scene is subject to large amounts of dynamic motion, then the framework will not be able to produce accurate results. What is meant by dynamic motion is that the shot consists of large amounts of motion that is constantly changing from the first frame to the last frame. A good example of dynamic motion is the infamous *Bullet Time* sequence in the motion picture *The Matrix*, where the protagonist Neo dodges bullets shot from an Agent by diving backwards with his arms flailing backwards in the air. Fig. 5.26 shows a sample of six frames from the sequence, with the first and last frames, while Fig. 5.27 illustrates the resulting depth maps.

As can be seen in the figures, as the camera is rotating around Neo, his distance from the camera should also change. However, the algorithm classifies that the depths are the same throughout the sequence. The most likely culprit is due to the limits restricted on searching for similar patches in the TLD algorithm. Also, with respect to the TLD algorithm, as most of Neo roughly stays the same size, the dynamic adjustment of the depths yields no change. Neo's arms are constantly changing location, and they are also the same size, yielding no change with respect to the dynamic depth adjustment. In addition, the background appears to be a bit noisy, mostly due to the high amount of motion that is seen in the background. Another observation is when examining Neo's trench coat, there is much motion that is exhibited, with parts of the cloak appearing closer to the camera, while others are farther. This is unfortunately not captured with the output result, due to the large amounts of motion exhibited in the scene. Should the amount of patches to search for increase, then the depth maps will most likely improve. This can also be resolved by labeling more frames in the sequence, which is unavoidable when it comes to a sequence such as this.

5.5 User Poll: Video

As a supplement to the previous section on showing results over video, performance cannot be measured unless subjective tests are taken, as the overall goal for this system is to create content for the stereoscopic perception for users to see. We gathered 10 graduate students, and 25 undergraduate students with different acuities and viewpoints on their preference on viewing 3D content. The range of ages from these students is from 20 to 30. The viewing medium we chose was using a PC platform, equipped with an Alienware 1080p AW2310 23" stereoscopic 3D monitor, using the nVidia 3D Vision Kit [8]. These tests were performed in a laboratory of low risk, as it comprised of mostly computers, server racks and office desks. The lighting was also controlled to ensure that the student is able to see his or her surroundings. Essentially, the lighting is reminiscent of all "lights being turned on", and mirrored the lighting conditions of many laboratories and classrooms that exist on the Ryerson University campus. However, the lighting conditions for perceiving 3D is not a factor [39][1], but we ensure that they are controlled for consistent results. In addition to the results that we have generated for this thesis, we generated other stereoscopic content, ranging from motion picture movies, to cartoon footage, and CGI-based films. The duration of each media ranged between 30 seconds to one minute, and so the conversion times took longer in comparison to what was shown in this thesis, as most that we have shown consisted of shots of a few seconds. The user labelling is also affected, as there are more keyframes to label. Even though the user only needs to mark the first frame, this is only valid for a single shot. If the sequence consists of multiple shots, the user labelling must be performed at the beginning of each shot. With respect to Guttmann *et al.*'s work, each shot required two frames to be labelled - the first and the last frame, which ensured valid comparisons to the proposed framework. Each student was instructed on how the test was to be performed: (1) The student was to sit roughly 1 metre away from the viewing screen. (2) The student was told that 12 different video shots were to be presented to them stereoscopically. (3) For each stereoscopic shot, the student was shown one version of the shot, and then the other version of the shot immediately after. (4) The student then votes on which of the stereoscopic shots they preferred, and why. One version of the shots was converted using the proposed method, while the other was converted using Guttmann *et al.*'s method. To ensure unbiased results, for each shot, we randomized which version of the shot that was to be seen by the user. Specifically, for some shots, some viewers viewed the results for our method, and then Guttmann *et al.*'s method, and vice-versa. This order was randomized for each shot that was shown to each user. Table 5.3 shows a table of those subjective results previously mentioned. The left column shows what video shot was presented to the user, the next column illustrates what type of shot it is, while the next two columns denote the percentage of students that preferred our method, and Guttmann *et al.*'s method respectively. As can be seen in the table, an overwhelming majority of students preferred our method, most likely due to the fact that the depth maps that were generated in our work are much more complete. They respect edges much better, and propagate the depths properly throughout the scene. With respect to the depth maps generated by Guttmann *et al.*, a substantial amount of points were classified as having no depth, and so no depth perception could be properly perceived. However, there were a few that did prefer this content more, most likely because they were not receptive to 3D content in the beginning, and preferred the monoscopic version of the scene, which is essentially what the depth maps generated using the system in [1] made. Curiously, the cartoon footage was the content that the students tended to prefer more with our method, most likely because cartoons have very precise and crisp edges, which naturally is favored in our method. The content that performs the "worst" are the television shows. This is most likely because there can be quite a bit of motion blur, as we chose shows that exhibited a high amount of motion. This is understandable, as it is common for viewers of stereoscopic content to experience visual discomfort with high amounts of motion, as the disparities can widely change through out the shot, and the visual cortex requires some time to adjust. As such, this could possibly explain why some of the viewers preferred the method in [1], as it is very close to perceiving the content in 2D.

Movie Shot	*Type*	*% for proposed*	*% for [1]*
Avatar	Motion Picture	82.8%	17.2%
Superman	Cartoon	91.4%	8.6%
Naruto	Cartoon	88.6%	13.4%
The Pacific	Television Show	62.9%	37.1%
Star Trek: The Next Generation	Television Show	71.4%	28.6%
Open Season	CGI	88.6%	13.4%
Big Buck Bunny	CGI	91.4%	8.6%
Sintel	CGI	82.8%	17.2%
24	Television Show	77.1%	28.9%
Planet Earth	Nature Documentary	80%	20%
Game of Thrones	Television Show	82.8%	17.2%

Table 5.3: Table illustrating subjective results recorded by 35 students, using a variety of different shots from different media

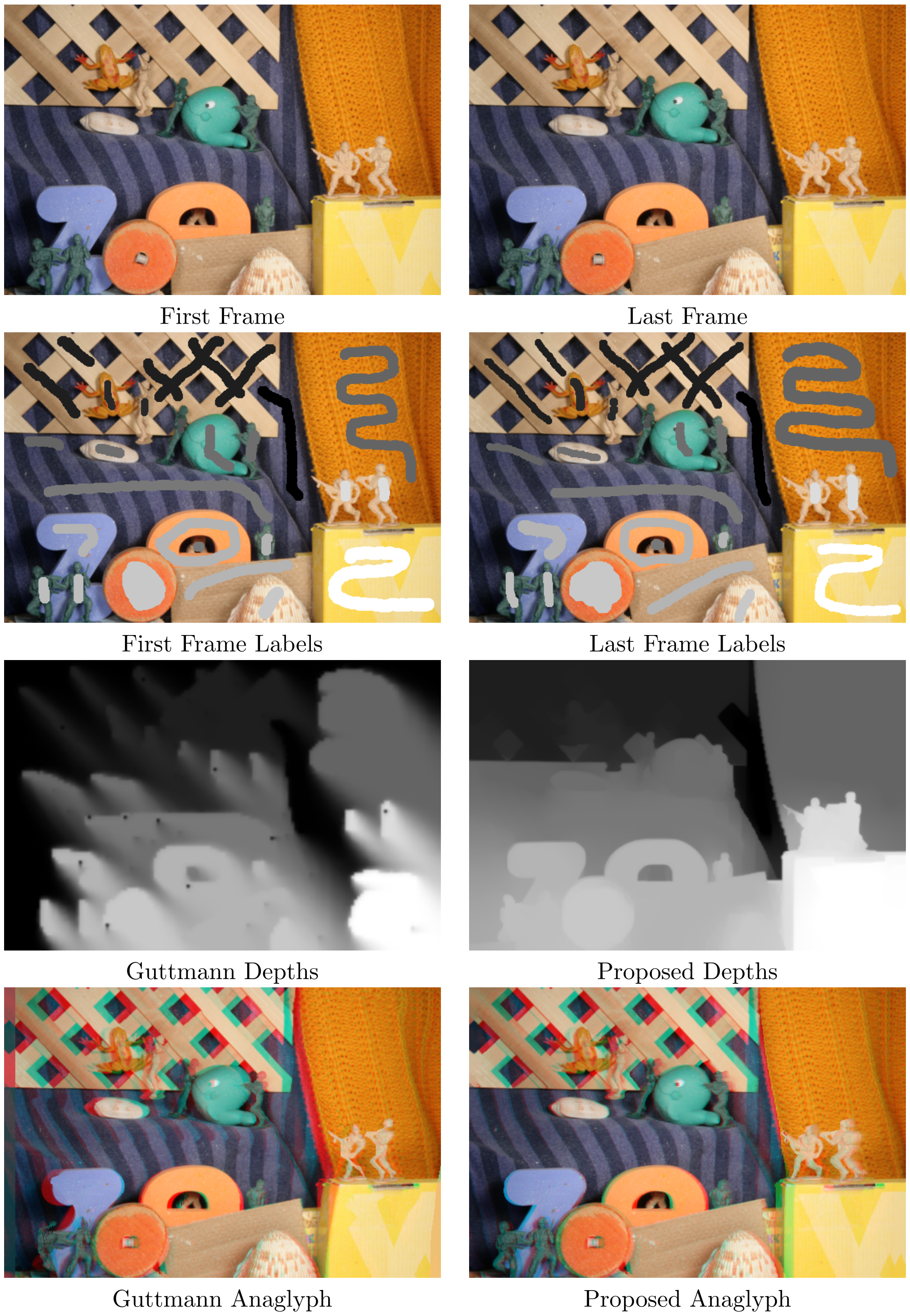


Figure 5.12: “Army”. First row - First and last frames. Second row - Labels overlaid. Third row - A depth map using the framework in [1] (left) and proposed (right). Fourth row - Anaglyph images using the third row

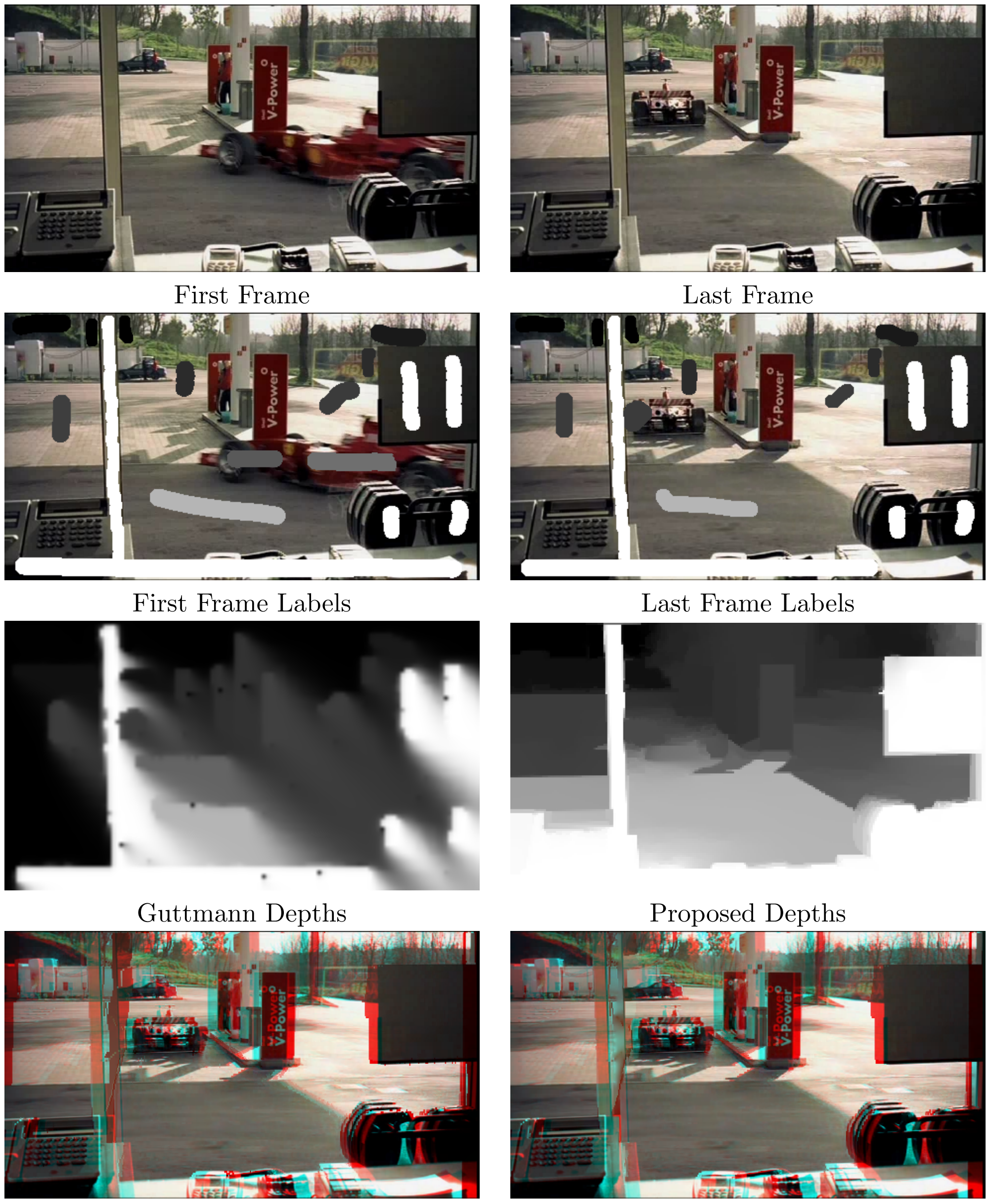
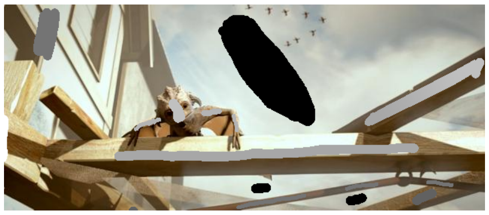
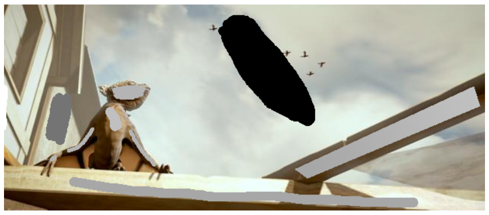


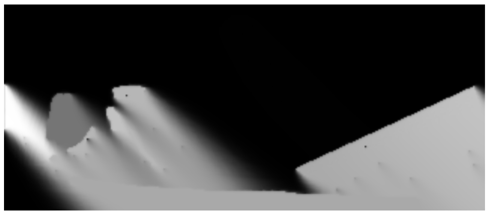
Figure 5.13: SFP. First row - First and last frames. Second row - Labels overlaid. Third row - A depth map using the framework in [1] (left) and proposed (right). Fourth row - Anaglyph images using the third row

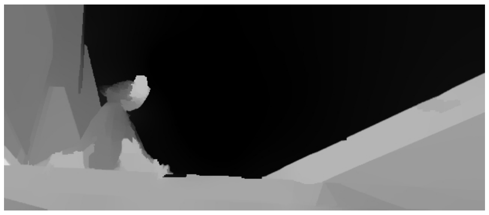
First Frame

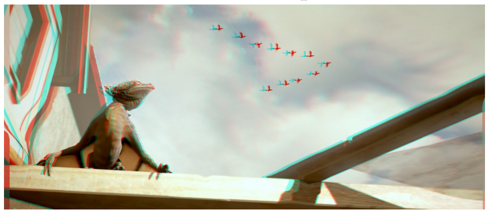
Last Frame


First Frame Labels


Last Frame Labels


Guttmann Depths


Proposed Depths


Guttmann Anaglyph

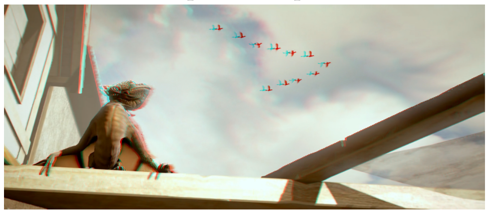
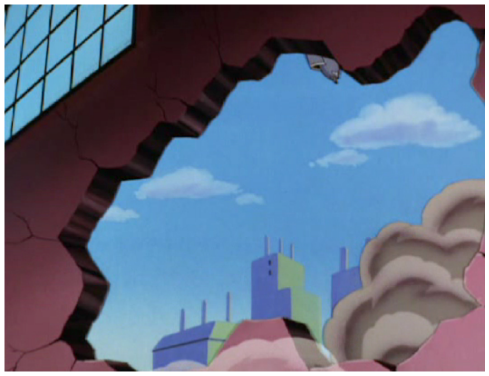
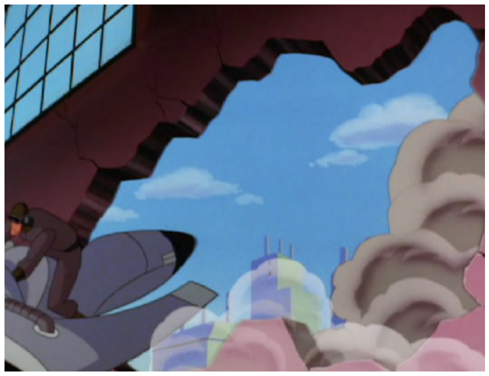
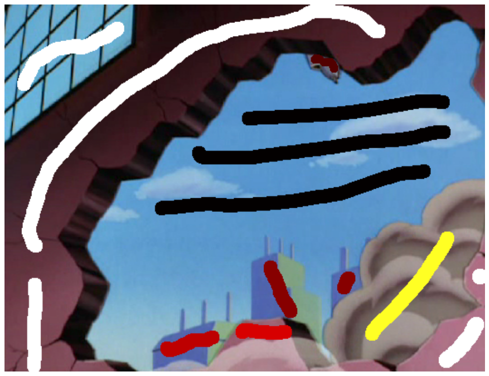
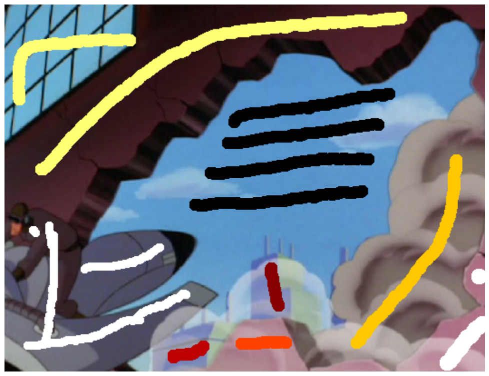

Proposed Anaglyph

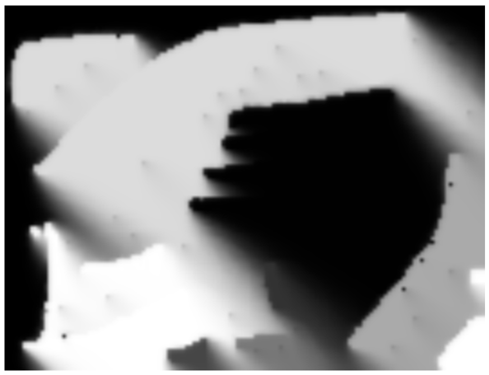
Figure 5.14: Sintel Dragon. First row - First and last frames. Second row - Labels overlaid. Third row - A depth map using the framework in [1] (left) and proposed (right). Fourth row - Anaglyph images using the third row

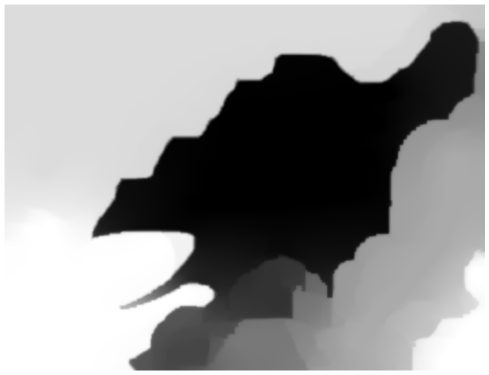

First Frame

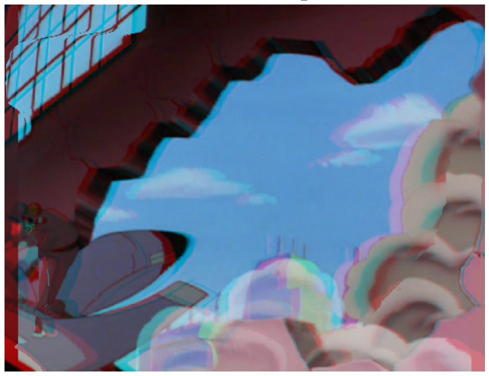

Last Frame


First Frame Labels


Last Frame Labels


Guttmann Depths


Proposed Depths


Guttmann Anaglyph

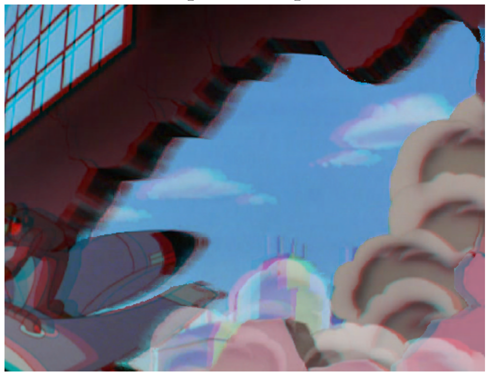

Proposed Anaglyph

Figure 5.15: Superman. First row - First and last frames. Second row - Labels overlaid. Third row - A depth map using the framework in [1] (left) and proposed (right). Fourth row - Anaglyph images using the third row

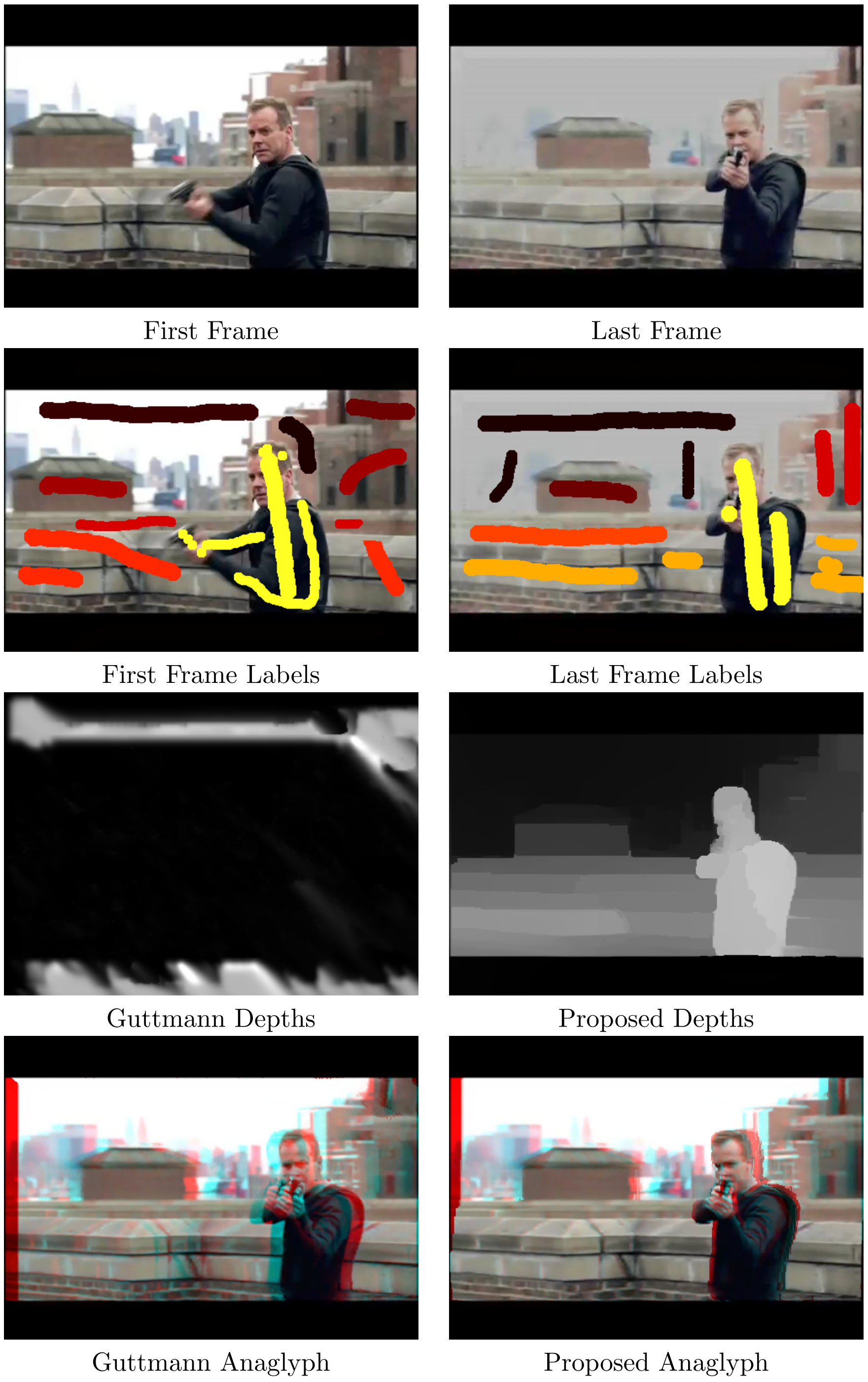


Figure 5.16: ‘24’. First row - First and last frames. Second row - Labels overlaid. Third row - A depth map using the framework in [1] (left) and proposed (right). Fourth row - Anaglyph images using the third row

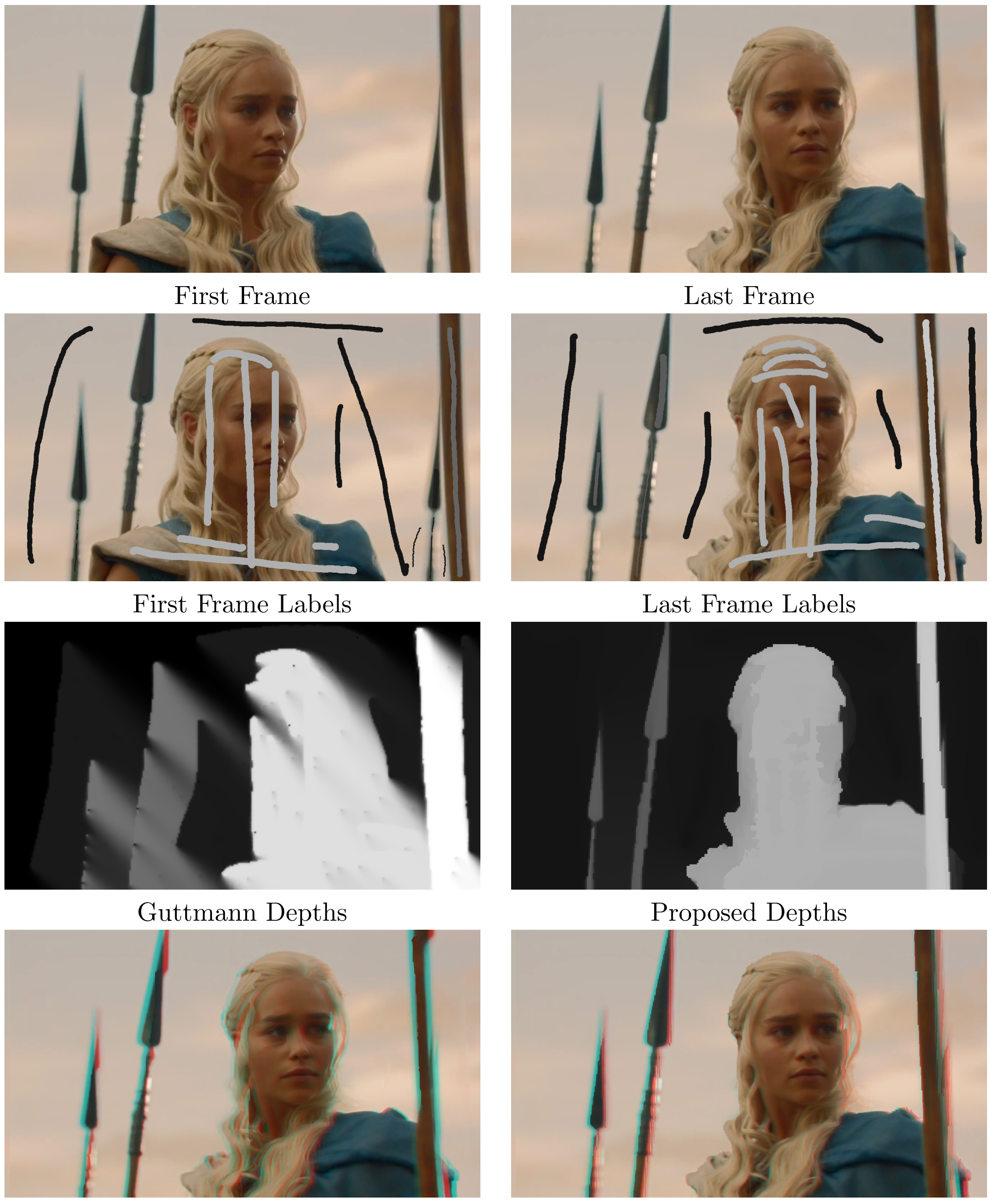


Figure 5.17: Game of Thrones. First row - First and last frames. Second row - Labels overlaid. Third row - A depth map using the framework in [1] (left) and proposed (right). Fourth row - Anaglyph images using the third row

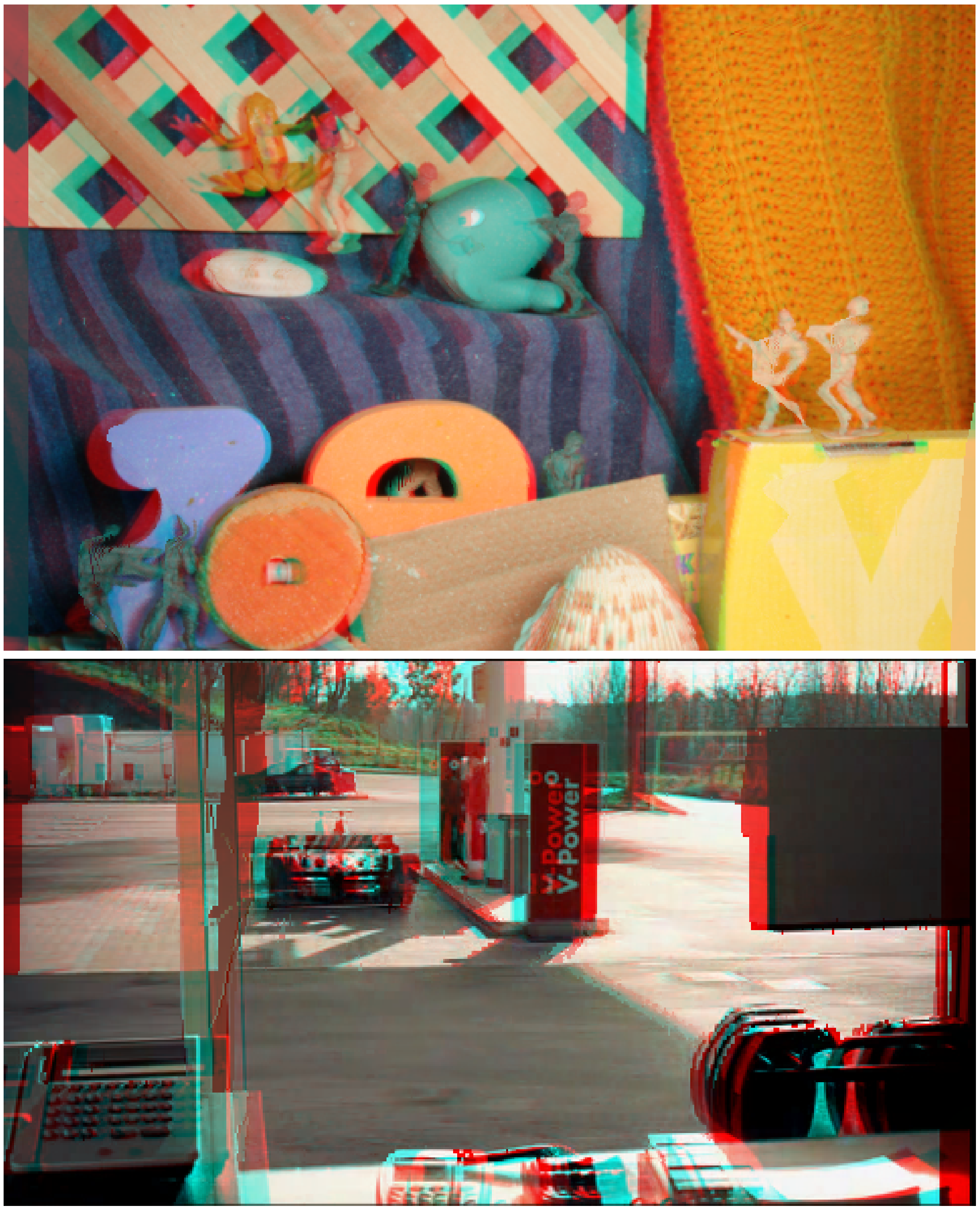


Figure 5.18: Guttmann anaglyphs discussed in Chapter 5.2

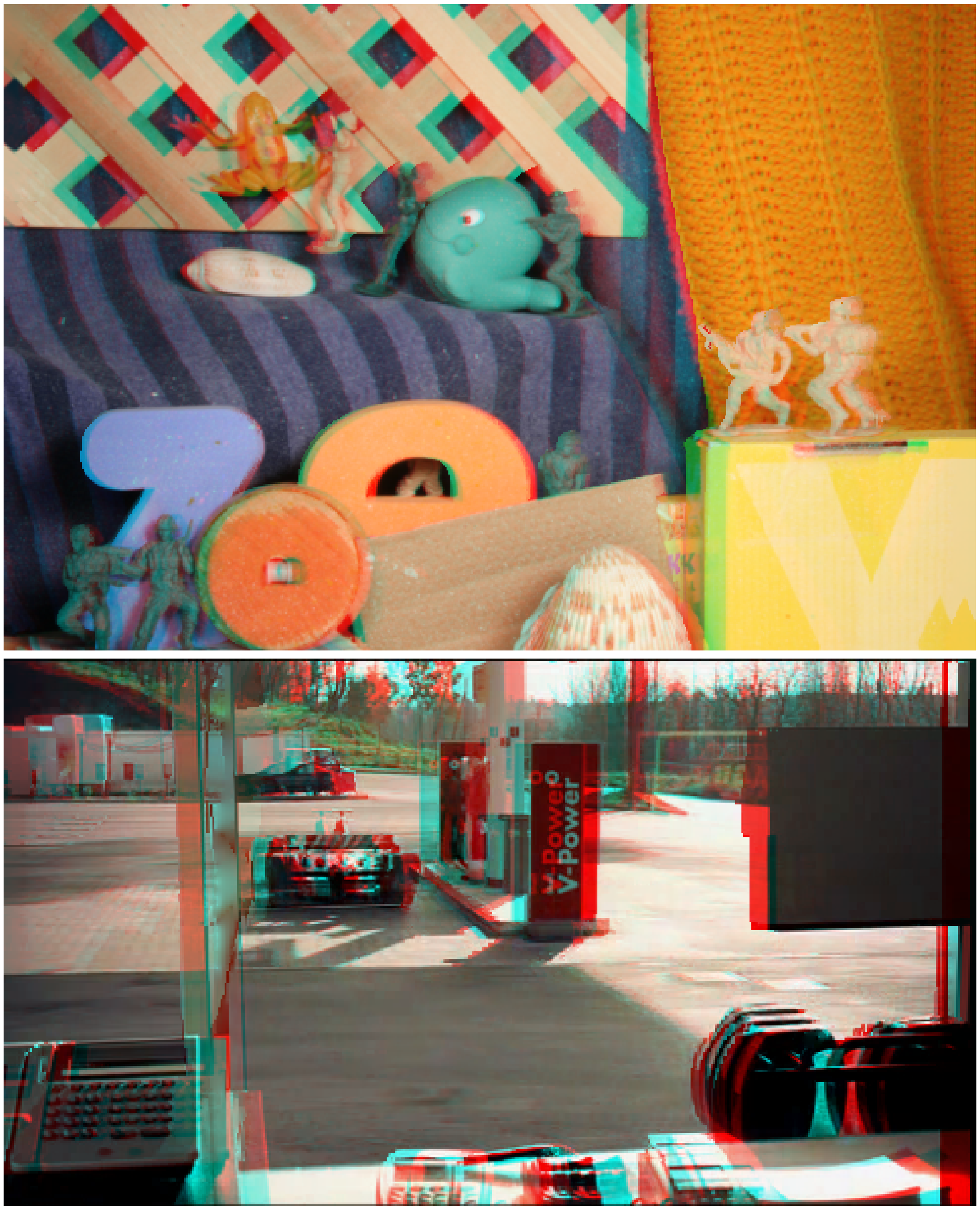


Figure 5.19: Proposed anaglyphs discussed in Chapter 5.2

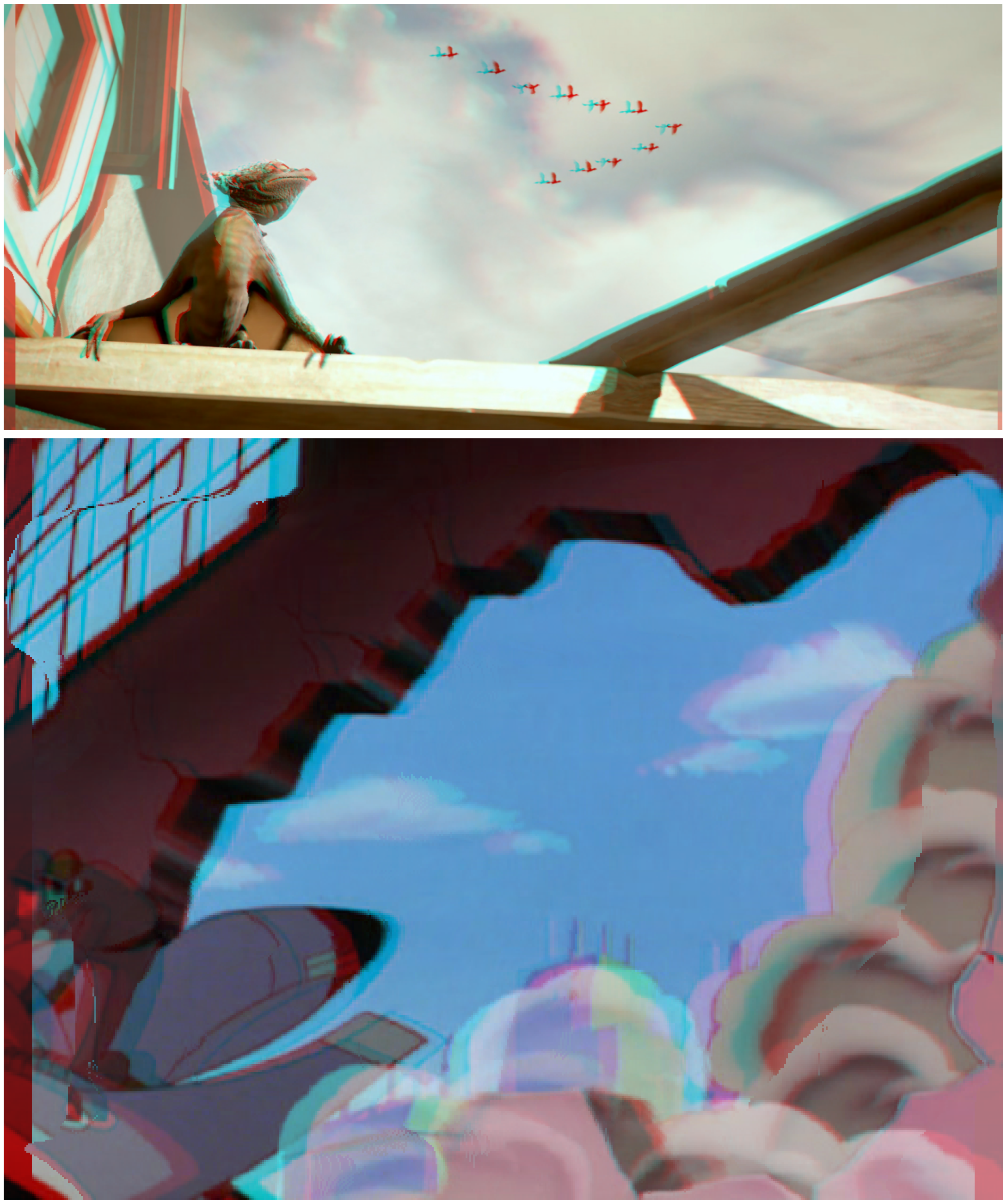


Figure 5.20: Guttmann anaglyphs discussed in Chapter 5.2

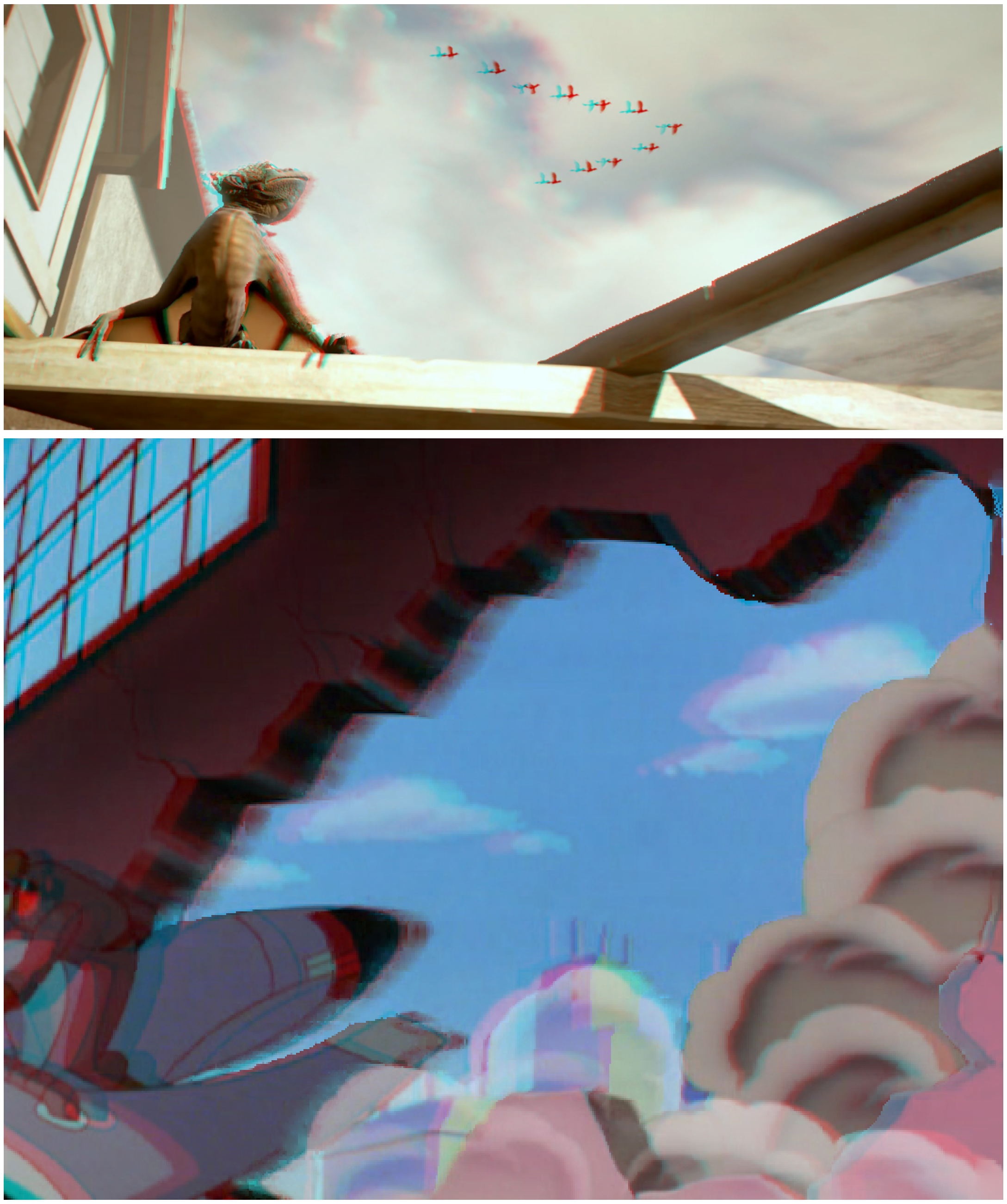


Figure 5.21: Proposed anaglyphs discussed in Chapter 5.2

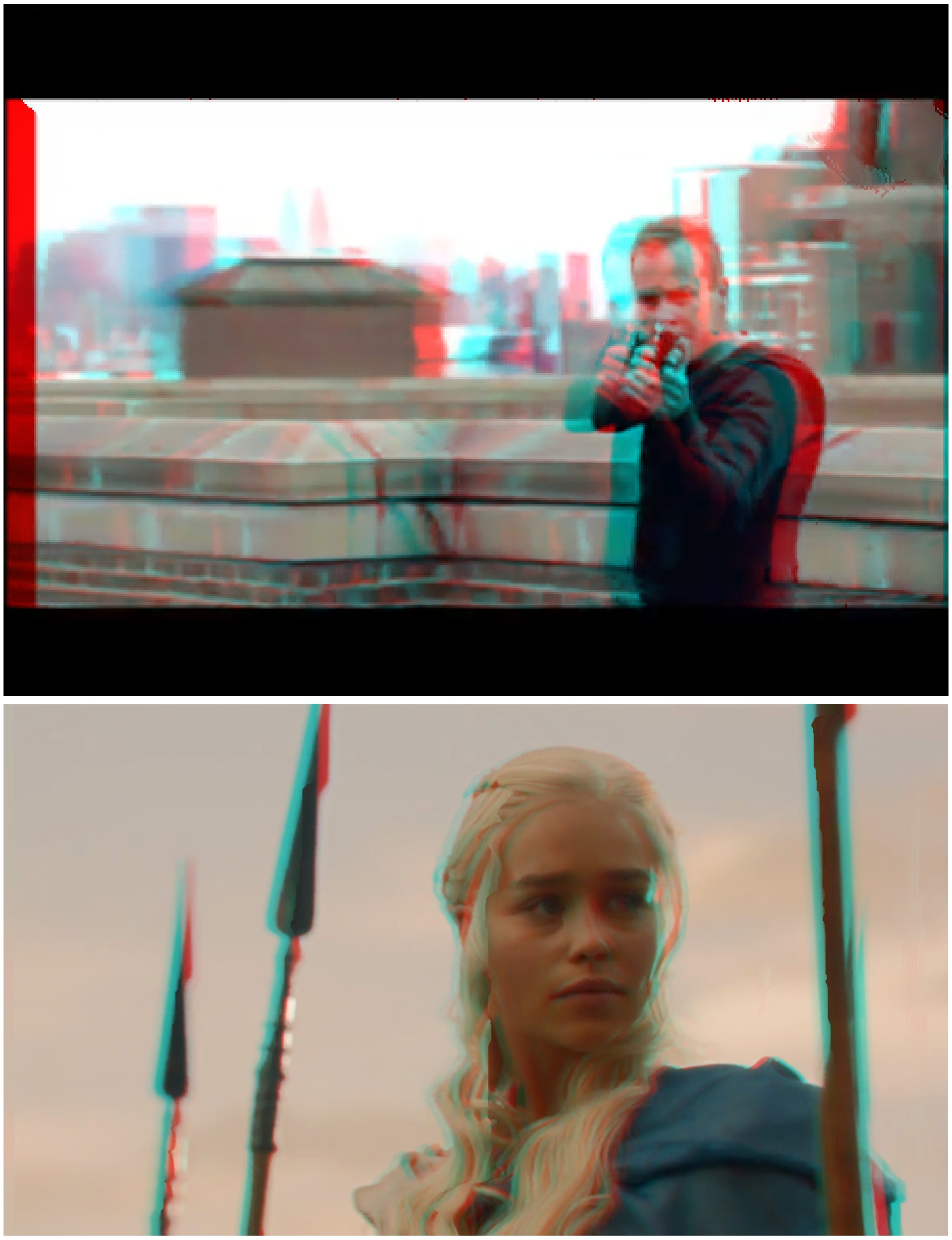


Figure 5.22: Guttmann anaglyphs discussed in Chapter 5.2

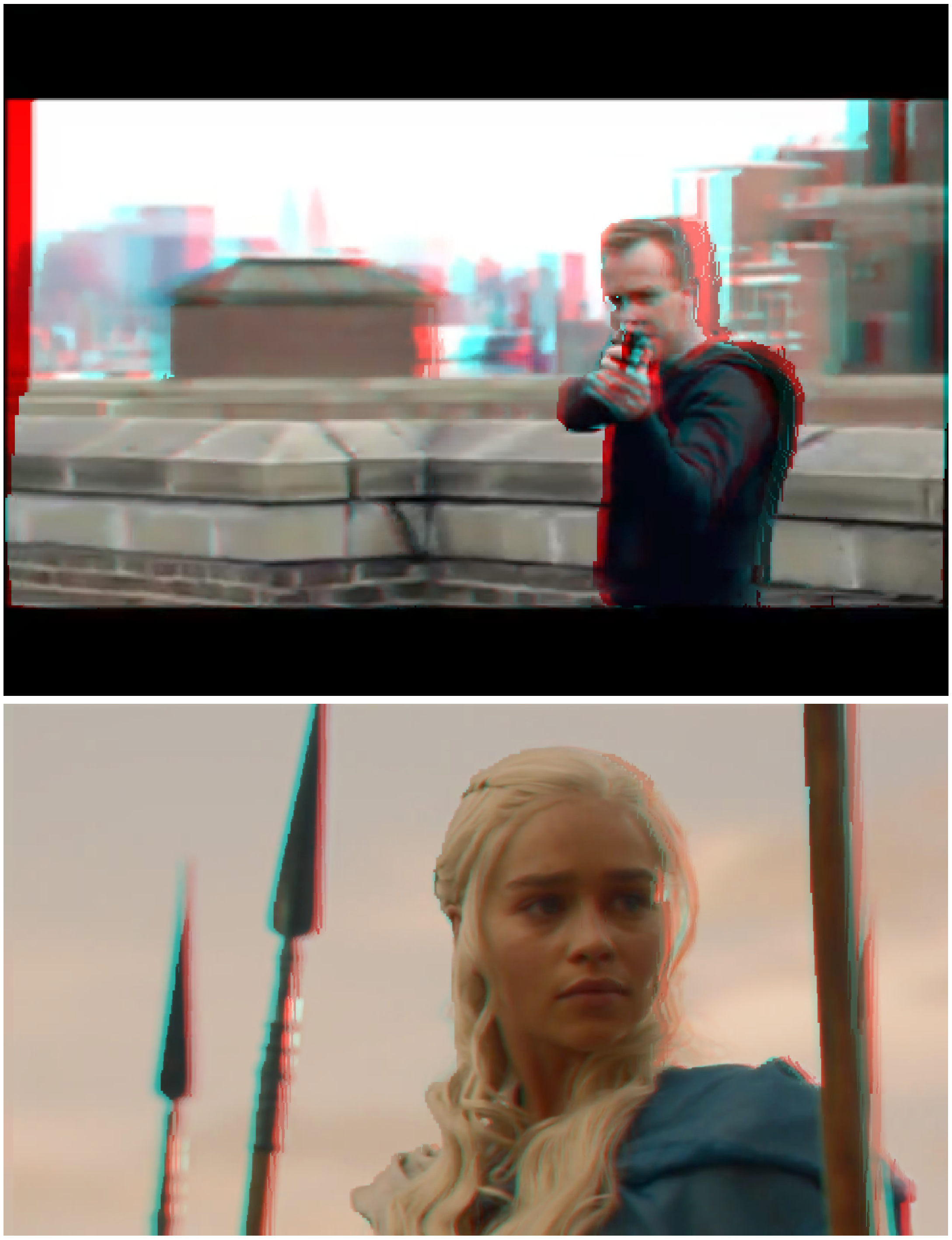
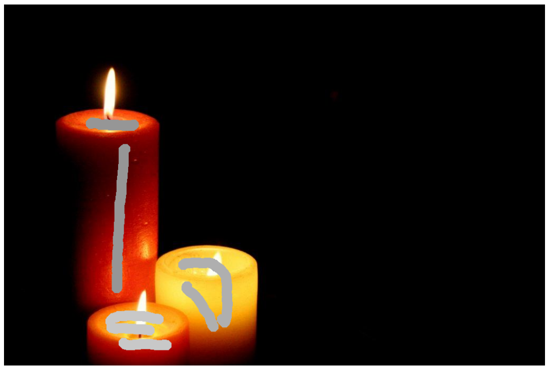
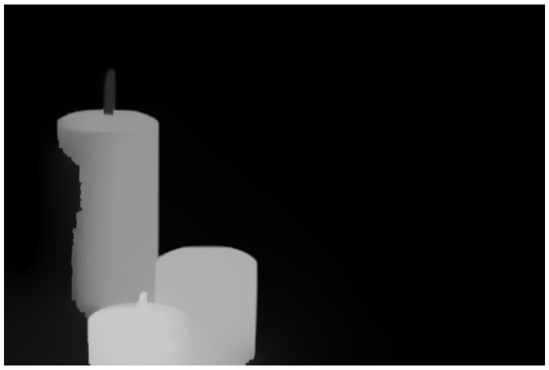
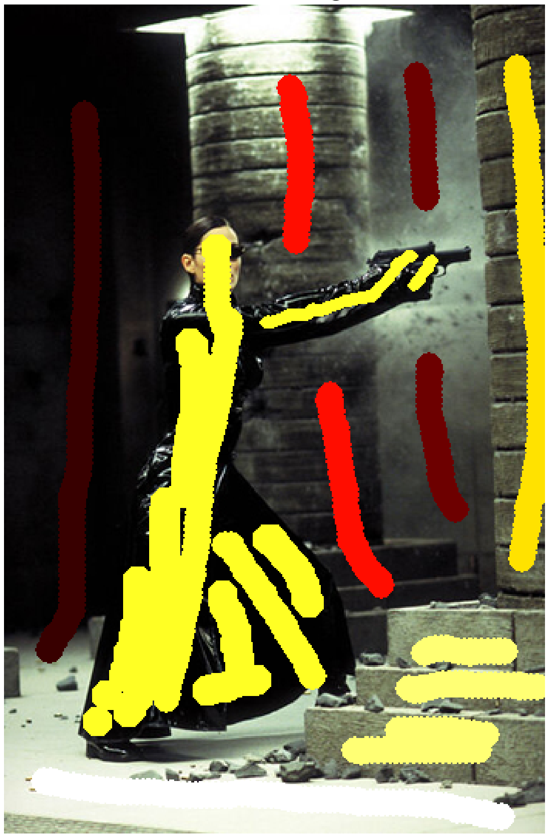


Figure 5.23: Proposed anaglyphs discussed in Chapter 5.2


Candle Image


Candle Depth Map


Trinity Image

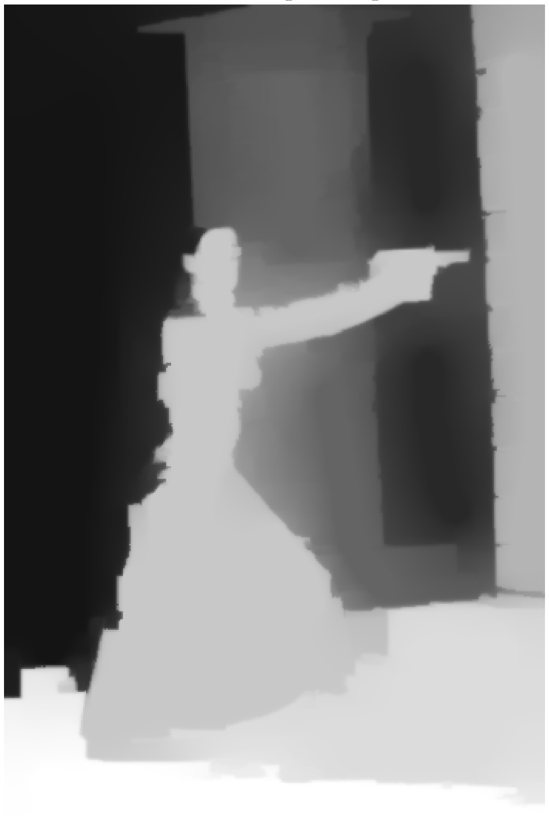
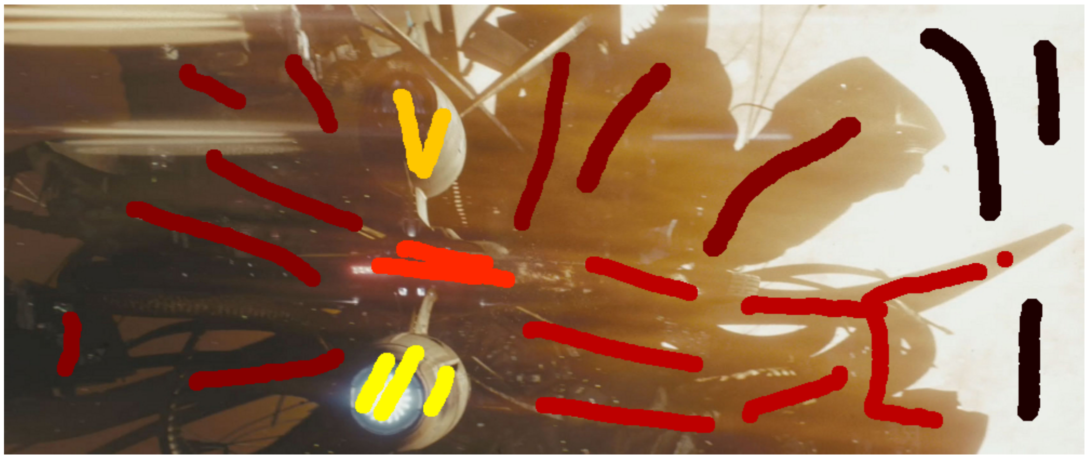
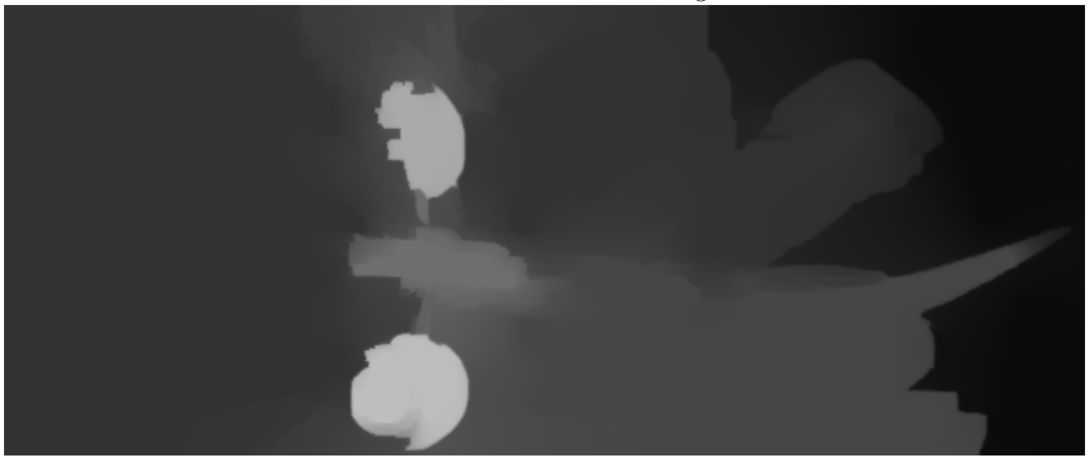

Trinity Depth Map

Figure 5.24: Failure Cases - Low Contrast



Star Trek - USS Kelvin Image



Star Trek - USS Kelvin Depth Map

Figure 5.25: Failure Cases - Low Contrast

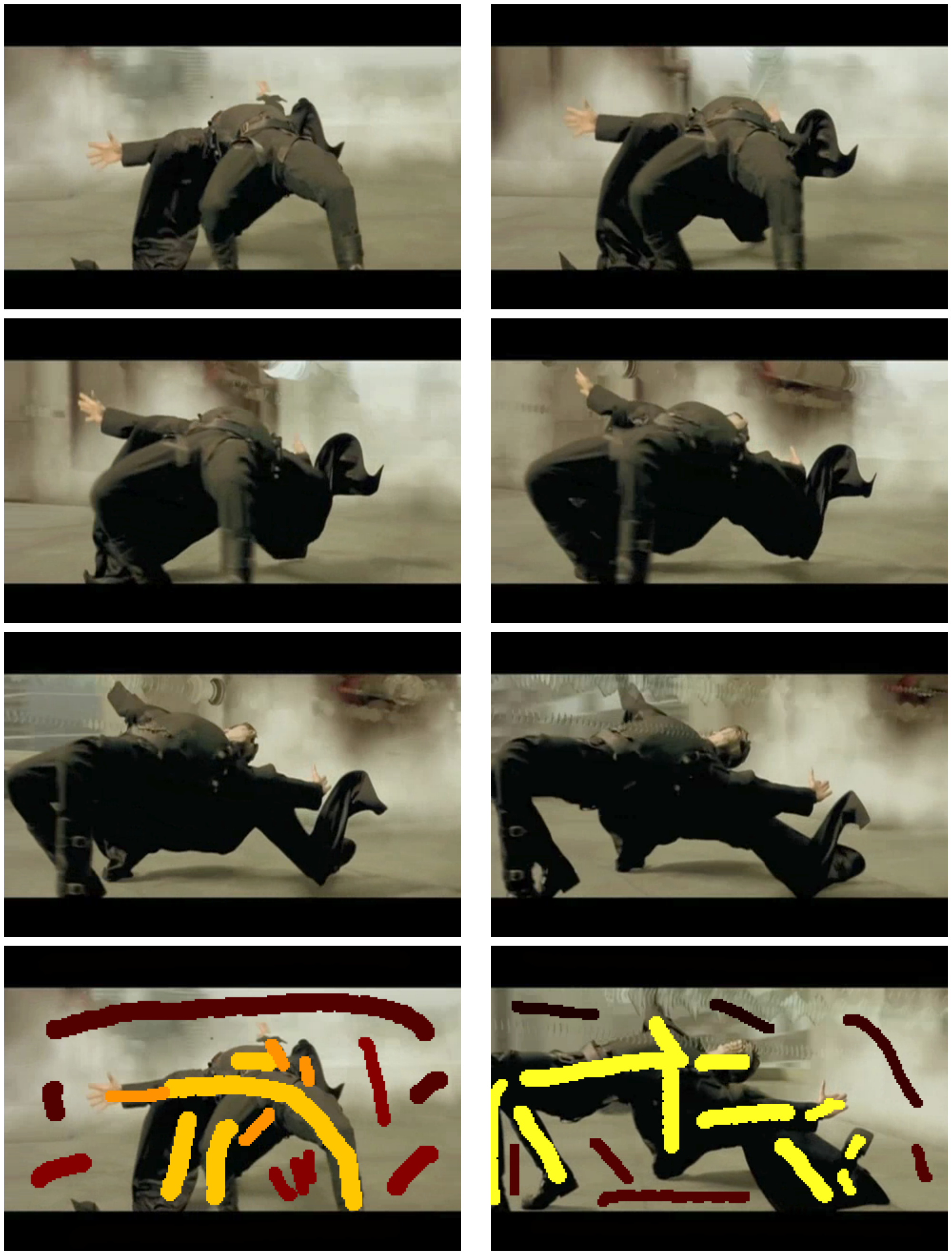


Figure 5.26: The Matrix: Bullet Time Sequence - Frames proceed from left to right, top to bottom, as well as the labels for the first and last frame

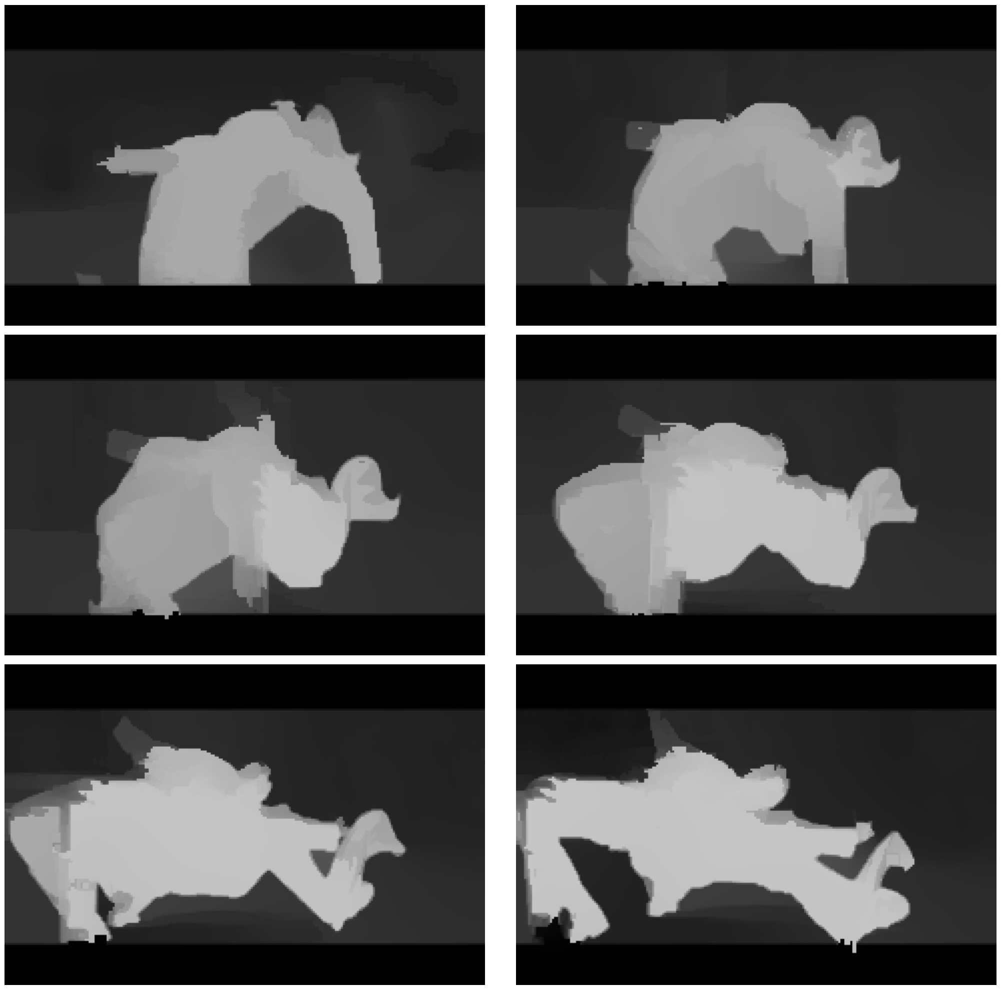


Figure 5.27: The Matrix: Bullet Time Depth Maps - Frames proceed from left to right, top to bottom

Chapter 6

Conclusions and Future Work

6.1 Conclusions

IN this thesis, semi-automatic framework was presented for obtaining depth maps for both conventional images and video sequences, for the purpose of converting this content into stereoscopic 3D. With minimal user effort, good quality stereoscopic content can be generated that is very well suited for human visual perception. Semi-automated systems are preferable to automated ones, as we can directly control the perceived depth for objects in the scene. Our work is similar to the framework designed by Guttmann *et al.*, but we show that our results are more pleasing to the viewer, and is better suited for viewing over stereoscopic hardware. The core of our system incorporates two existing semi-automatic image segmentation algorithms in a novel way to produce stereoscopic image pairs. The incorporation of Graph Cuts into the Random Walks framework produces a result that is better than either on its own.

A modified version of this that ensures temporal consistency was created for converting stereoscopic video sequences. We provided the user with a method to track labels based on a computer vision based tracker as a means of ensuring high accuracy, alleviating much user input. In addition, we provide an alternative means of tracking using a temporally consistent Optical Flow based method for scenes that exhibit only horizontal motion. As such, the user need only mark the first frame of the video sequence, and one of two label propagation and tracking algorithms are used to distribute the labels to the rest of the frames. Depending on the kind of motion that is seen in the scene, either the motion-based algorithm is used, or the Optical Flow based algorithm is used, thus emphasizing on the tracking of depth labels in between keyframes. We also allow the user to manually choose the keyframes, allowing the user to have full control of the user labeling, if they so desire.

In summary, the following scientific contributions were made to attempt to solve the problem of converting conventional images and video sequences into their stereoscopic counterparts:

- Modified both the Random Walks and Graph Cuts schemes, which are semi-automatic binary image segmentation algorithms, for the purposes of performing multi-label segmentation. The way images are represented in the proposed framework are in terms of graphs, with edges that connect to other pixels that represent the similarity between pixels, and other pixels connected to them. Multi-label image segmentation can be seen as a way of generating depth maps in a semi-automated fashion, where the user simply places strokes on objects or regions on what they believe is closer or farther from the camera. The result generates a dense depth map that is representative of the scene, using the user as a powerful tool to guide the accuracy in generating the result. Both Random Walks and Graph Cuts are modified in such a way that intuitively extends the binary segmentation case to the multi-label case.

- Graph Cuts is used to generate a depth prior - a rough estimate of the depths in the scene. This rough estimate respects the boundaries of objects quite well, but does not have internal depth variation. As such, these objects would be perceived as cardboard cutouts, and would not appear to be perceptually realistic. However, this serves as additional information into the Random Walks algorithm, thus serving as the method of merging the two methods together. Random Walks allows for internal variation of the depth of objects, but does not respect edge boundaries very well. By combining these two frameworks together, both merits are present in a depth map, to generate the best possible result for enhanced user perception. This depth map is ultimately used in generating an artificial stereoscopic view through Depth Image Based Rendering, (DIBR) for the purposes of displaying on stereoscopic hardware.
- An extension of this algorithm was made to video sequences. Essentially, videos are just a sequence of images at a given frame rate. To extend this to video, the graph representation of these are three-dimensional, where edges in the graph connect temporally between frames. The result of this is a sequence of depth maps, one per frame in the video sequence, and each frame in the video is converted to its stereoscopic counterpart through DIBR. After, the stereoscopic sequence can be shown in compatible hardware.
- Because video is essentially a sequence of images at a specified frame rate, more than one frame within the video sequence needs to be marked by the user so that the depths are estimated consistently throughout the sequence. The ideal situation is that all of the frames within the sequence are marked, but this would be a significant undertaking for the user, and would be quite tedious. As such, the proposed framework has the option of marking only the first frame of the sequence, and these labels are propagated through the rest of the sequence, ultimately providing "user"-defined labels. However, these were labeled automatically by a label propagation algorithm.
- There are two methods to performing this label propagation algorithm. The first is through modifying a computer vision object tracking algorithm, which has the ability of dynamically adjusting the depths of regions as they come closer or farther from the camera. The modification adopts Colour Edge Gradient Co-occurrence Histograms, with a colour quantization scheme that is shown to mimic the way humans perceive colours. This ultimately improves the tracking algorithm, as the original algorithm operates on monochromatic videos.
- The first method can be computationally intensive, as there are many stages involved before tracking can begin. As such, the proposed framework suggests an alternative method. This is

by creating an edge-aware temporally consistent optical flow algorithm should the scene exhibit only horizontal motion and can avoid some of the computational burden in comparison the first method, without the use of global optimization. The second method can become very attractive for implementation in dedicated parallel architectures, such as GPUs or FPGAs. To make the optical flow algorithm temporally consistent, the distance transform is used, which is an enhancement framework that smoothes signals, and simultaneously ensures that the edges are well maintained.

6.2 Future Work

As the reader may have noticed, the quality of the final depth map is completely dependent on the user input, as well as the depth prior. With this, we introduced the constant α to control the depth prior contribution, mitigating some of the less favorable effects. For future research, we are currently investigating how to properly set this constant, as it is currently is static and selected *a priori*. We are investigating possible means for adaptively changing α based on some confidence measure to determine whether one paradigm is preferred over the other. As Graph Cuts respects edges well, the edge strength could be used as a weighting to the framework.

For video (as well as images), the most obvious limitation is that the depth maps are completely dependant on the user labeling. In practice, the actual labeling process is quite intuitive and straightforward. Once the user has provided the labeled keyframes, if there are errors, they can simply augment or adjust the labels so that they are properly placed on the different objects in the scene. Verifying that the depths are correct can be done quickly by examining the depths for just a single frame without considering any of the other frames. Once the user is satisfied with the labelling, they can allow the system to solve for the entire frame sequence. One possible future avenue of research that can be sought out is to explore the use of Active Frame Selection [111], where a prediction model determines the best keyframes in the video sequence that should be selected for the user-defined labeling. Once these frames have been determined, any user labeling that result using these selected frames will generate the segmentation of objects in the sequence, and thus the best possible stereoscopic video sequence, given our proposed framework.

Another possible future avenue of research is to discard the motion tracking based algorithm for propagating labels, and to rely on the Optical Flow method completely, as it can be more efficiently calculated. However, there is no scheme to dynamically adjust the depths of areas and locations by simply looking at the Optical Flow vectors themselves. As such, perhaps some classification scheme, similar to the SVM stage seen in Guttmann *et al.*'s framework may possible work here. Overall, our framework is conceptually simpler than other related approaches, and it is able to achieve similar results (if not better). Because our work is one of a semi-automatic approach, this makes it possible to correct errors in the depth maps, something not easily possible with automatic methods.

Appendix 1

Guttmann Implementation Details

IN this chapter, the implementation details, design choices and assumptions are described for Guttmann *et al.*'s framework in this thesis. For a given video volume for frames, indexed by $t = [0, 1, 2, \ldots, N_F - 1]$, where N_F is the number of frames in the given sequence, at each spatial location (x, y) at a frame t, the disparities at each location can be quantified as $d(x, y, t)$. Specifically, a tuple of spatial locations (x, y), at a given frame t is defined as (x, y, t), and we seek to determine the disparity $d(x, y, t)$ at each tuple defined in the video volume. Every disparity value $d(x, y, t)$ are essentially the unknowns that we are required to solve for, with the exception of the user-defined strokes and the anchor points (more on this later). As a consequence, we will let $I(x, y, t)$ be the colour pixel value at a particular (x, y) co-ordinate at frame t, excluding the user-defined strokes given. The user-defined strokes and the anchor points are known information, and will contribute to solving for the disparities for the rest of the video sequence. To briefly summarize the method, it follows three points, and we will cover each point, as well as the implementation choices, in detail.

Step #1: Detect Anchor Points over All Frames: A rough estimate of the depth for the first and last frame is found, and Support Vector Machine (SVM) classifiers [40] are trained on both frames separately. Each SVM is trained on classifying a unique depth seen in the rough estimate, using the Scale-Invariant Feature Transform (SIFT) [41], combined with the grayscale intensity at the particular point as the input space. To detect anchor points, SIFT points are detected throughout the entire video sequence, and are put through the SVM classifiers. For each SIFT point, a One-Vs-All scheme is employed, choosing the highest similarity out of all the classifiers. If this surpasses a high-confidence threshold, this point is an anchor point, and the depth that is assigned is the one belonging to the SVM classifier tuned for the particular depth.

Step #2: Build Sparse Linear System: To solve for the rest of the depths in the video sequence, an energy function is minimized by least squares, where the function is a combination of spatial and temporal terms, colour information, the anchor points and user strokes. These terms contribute to a sparse set of linear equation, and a sparse linear system is created.

Step #3: Solving for Depths over All Frames: Least squares minimization is performed by directly solving the sparse linear system given in Step #2.

1.1 Detect Anchor Points

In [1], only the first and last frame have user-defined information, and Guttmann *et al.* maintain that using only the last two frames results in a sparser representation of data so that storage in memory is more efficient. The user only needs to mark two out of hundreds of consecutive frames, and is more user friendly. In order to help with solving depths for the rest of the video sequence, [1] devise a method of detecting what are known as *anchor points*, which are points in the video volume that have a strong probability of determining that a pixel in between the first and last frame can be successfully classified to belong to a particular depth. Essentially, a set of SVM classifiers will be used to achieve this classification, which ultimately generate these anchor points, to be used for solving the depths for the rest of the video sequence. Ultimately, the system will not only depend on just the user-defined strokes, but on the depths that these anchor points are classified as within the video sequence. To detect anchor points, three individual stages must be pursued: (a) The training stage, (b) the calibration stage, and (c) the classification stage.

1.1.1 Training Stage

In this stage, a set of SVM classifiers must be created, and trained on both the first frame, and the last frame individually. This results in two sets of SVM classifiers, where one set is for the first frame, and the other for the last frame. As there are user-defined scribbles from both frames, each with their own unique depths, the combination of these two together will aid in determining the anchor points throughout the entire video sequence. Before performing SVM training, a rough estimate of the depths in the scene is required first, as the SVM classifiers will need some form of training data, in order to proceed with the classification. This rough estimate is also solved with a sparse linear system of equations as well, but only using spatial information, as well as the information given from the user defined strokes themselves.

For the spatial information, the sparse system will depend on weights that reflect the local edge energy at the location of each pixel in the video volume. Pixels that lie on edges are less constrained to be similar than with their neighbouring pixels. The spatial weight of a particular pixel at a tuple (x, y, t) in the video volume is $W_E(x, y, t)$, and is defined as:

$$W_E(x, y, t) = \sqrt{2 - \left[\left(\frac{\partial I(x, y, t)}{\partial x}\right)^2 + \left(\frac{\partial I(x, y, t)}{\partial y}\right)^2\right]} \quad . \qquad (1.1)$$

$W_E(x, y, t)$ is related to the L_2-norm of the image gradient, and all colour values in the video volume should be normalized between a range of $[0, 1]$ for this to function properly. $\frac{\partial I(x,y,t)}{\partial x}$ and $\frac{\partial I(x,y,t)}{\partial y}$ are the discrete derivatives in the x and y direction respectively at tuple (x, y, t). As Guttmann *et al.*'s implementation of the derivative is not defined in the paper, we take this to be a simple discrete approximation of the derivative, where their horizontal and vertical derivatives, or x and y derivatives are defined as:

$$\frac{\partial I(x,y,t)}{\partial x} \approx I(x,y,t) - I(x-1,y,t) \tag{1.2}$$

$$\frac{\partial I(x,y,t)}{\partial y} \approx I(x,y,t) - I(x,y-1,t) \tag{1.3}$$

With these spatial weights, a set of linear equations is introduced, so that the collection of them create a sparse linear system, and solving these will produce the best solution under a least squares framework [1]. Each set of linear equations will contribute one portion to building a sparse linear system of equations to solve for each disparity in the video volume $d(x,y,t)$. These linear equations are defined as:

$$c_1 W_E(x,y,t)(d(x,y,t) - d(x-1,y,t)) = 0 \tag{1.4}$$

$$c_1 W_E(x,y,t)(d(x,y,t) - d(x,y-1,t)) = 0 \tag{1.5}$$

c_1 controls how much contribution the spatial and temporal terms will contribute to the linear system. In Guttmann *et al.*'s work, this is set to 1. Finally, the user-defined strokes must be incorporated. Taking into account that the user is the best person to determine what the depths are in certain user-defined regions, these terms to be introduced into the sparse linear system of equations should be weighted relatively high, as this information should contribute the most in determining the final disparities. This weight, c_3 is set to 10 in Guttmann *et al.*'s work. By defining $V(x,y,t)$ to be the user-defined depth at tuple (x,y,t), the following equation is added for each user-defined pixel in the video volume:

$$c_3 d(x,y,t) = c_3 V(x,y,t) \tag{1.6}$$

To finally build the sparse system of equations, a linear system of the form $A\vec{d} = b$ is finally created, where d is a rows $\times$ cols $\times$ N_F column vector, containing the disparity values for each pixel in the video volume, and ultimately the values we need to solve for. A is a sparse matrix of coefficients that are fundamental in solving for the disparities. Each i^{th} row of A in the sparse matrix corresponds to the i^{th} disparity to solve for in the volume. For each i^{th} row, the non-zero columns will correspond to the coefficients for each of the terms in the set of linear equations that ultimately solve for the i^{th} disparity, and placed in their correct columns. Specifically, in a least squares minimization framework, letting l_i be single index that references one pixel in the tuple of (x,y,t), the i^{th} set of linear equations will simply add the spatial and user-defined terms together. The index l_i can be calculated using the tuple (x,y,t) in the following fashion: $l_i = \text{rows} \times \text{cols} \times t + x \times \text{cols} + y$, assuming that x, y and t begin with 0-indexing. As such, for a particular disparity value $d(x,y,t)$, l_i determines the right row of where to populate the non-zero coefficients of the sparse matrix A. Also, the vector b will store the user-defined information, corresponding to the depths assigned to the particular pixels that the user has chosen, where each i^{th}

row of this vector is either 0 if the pixel has an unknown disparity, or $c_3V(x,y,t)$ if the pixel is unknown and needs to be solved for. The index l_i is used to place coefficients in this vector, and is calculated using the aforementioned indexing function. To put all this together, the i^{th} equation to solve for the disparity at pixel i, and placing the equation in row l_i of the matrix A is the following:

$$c_1W_E(x,y,t)(d(x,y,t)-d(x-1,y,t))+c_1W_E(x,y,t)(d(x,y,t)-d(x,y-1,t)) = 0, \text{if } d(x,y,t) \text{ is unknown } . \tag{1.7}$$

By collecting like terms, we thus get the following equation:

$$2c_1W_E(x,y,t)d(x,y,t) - c_1W_Ed(x-1,y,t) - c_1W_Ed(x,y-1,t) = 0, \text{if } d(x,y,t) \text{ is unknown } . \tag{1.8}$$

The above is when the disparity $d(x,y,t)$ is not user-defined, and requires being solved. Similarly, should $d(x,y,t)$ be given by the user, the equation is to include an additional term of $c_3d(x,y,t)$ on both the left and right hand sides of the equation. In other words:

$$\begin{gathered} c_1W_E(x,y,t)(d(x,y,t)-d(x-1,y,t)) + c_1W_E(x,y,t)(d(x,y,t)-d(x,y-1,t))+ \\ c_3d(x,y,t) = c_3V(x,y,t), \text{if } d(x,y,t) \text{ is user-defined } . \end{gathered} \tag{1.9}$$

By collecting like terms, the following equation is produced:

$$(2c_1+c_3)W_E(x,y,t)d(x,y,t)-c_1W_Ed(x-1,y,t)-c_1W_Ed(x,y-1,t) = c_3V(x,y,t), \text{if } d(x,y,t) \text{ is user-defined} \tag{1.10}$$

By using the linear indexing equation mentioned previously, by letting l_i be the index defined for tuple (x,y,t), l_j be the index defined for tuple $(x-1,y,t)$ and l_k be the index defined for tuple $(x,y-1,t)$, for each row l_i of the matrix A, there will be exactly three terms in this row, where the coefficients for the $d(x,y,t)$, $d(x-1,y,t)$ and $d(x,y-1,t)$ in the aforementioned equation are placed in the corresponding columns, referenced by l_i, l_j, and l_k respectively. In addition, for the vector b, $c_3V(x,y,t)$ is placed in row l_i if the depth at this pixel was user-defined, and is 0 otherwise. In Guttmann *et al.*'s work, these depths are solved using a direct solver, or in other words: $\vec{d} = A^{-1}b$. However, as the video sequence can be large, this will inevitably exhaust all available memory on the machine employed to solving the depths - an oversight not mentioned in Guttmann *et al.*'s work. As such, we turn to *iterative* methods, and the Successive Overrelaxation (SOR) method [52], with an initial solution vector of all zeroes is used. The SOR weight of $\omega = 0.4$ was used, with a maximum number of iterations set to 3000, as this was experimentally chosen to ensure that the solution of the system converged, without allowing it to become unstable. It has been shown [52] that choosing a weight between $0 \leq \omega < 1$ allows for more stability, at the expense of require more iterations to solve the system. A weight of $1 < \omega < 2$ allows for faster convergence, but may unintentionally introduce stability if the system is poorly conditioned.

After obtaining a rough estimation of these disparities, there will inevitably be new disparities that were not originally introduced by the user. As the SVMs require an integer number of labels, we must determine a "hard" labeling of these disparities once they are solved. To perform this, a unique list of user-defined labels from the first frame is obtained. After, for each solved disparity in the first frame, a hard label for this pixel in the first frame is assigned to be the disparity that is the closest from the unique list. This will create a hard label map for the first frame $\mathcal{H}_f$, where each pixel is given a user-defined depth value that is the closest to any one within the unique list. A unique list of user-defined labels for the last frame is obtained, and the procedure is repeated to obtain a hard label map for the last frame, $\mathcal{H}_l$.

The next step of this process is to detect SIFT points for the first and last frame. As SIFT itself is protected under Intellectual Property, some details within Lowe's original paper [41] are missing. In addition, Guttmann *et al.* do not mention how they implemented their SIFT algorithm, or what tools they used to do the detection. As such, this thesis uses an open-source "out-of-box" implementation from the *VLFeat* toolbox [112], which provides an almost accurate implementation of the SIFT algorithm. As SIFT is originally defined over grayscale images, the video volume is converted to their grayscale equivalents for use in SIFT. In addition, we change no parameters from *VLFeat* to ensure that the algorithm can be replicated, should the reader be interested in recreating Guttmann *et al.*'s work. The SIFT algorithm generates a set of reliable and robust feature points detected in the frame, where each point is represented as a 128 element vector, capturing the accumulation of edges, angles, and other features at different orientations. Guttmann *et al.* experimented with the fact that appending the grayscale value for each SIFT keypoint would help in allowing the SVM classification stage to become more robust. This ultimately results in a 129 element vector: 128 elements from the SIFT keypoint, with the additional grayscale value defined at that point. For the first frame, SIFT points are detected, and the purpose of this is to detect robust, and repeatable features. These are ultimately used as input into the classifiers, where the input features should be reliable and robust. This, in total, will generate N_1 SIFT points within the first frame, with each point accompanied by a 129 element vector.

After locating where the SIFT points are located throughout the first frame, the hard label values for these locations are obtained from $\mathcal{H}_f$. This ultimately provides a set of training examples, where each SIFT point is associated with a user-defined depth taken from $\mathcal{H}_f$. For the first frame, by letting D_f be the number of user-defined depths in the frame, there will be D_f SVMs, where each is trained for each user-defined depth in the first frame, resulting in a total of D_f SVM classifiers for the first frame. SVM classifiers operate on *binary* classification, which classifies an input into one of two classes. To train an SVM, a set of *positive* and *negative* examples are provided. Positive examples are inputs provided for training, and should classify this to belong to the first class, while negative examples are those where these inputs should be classified to belong to the second class. However, for multi-class classification, when attempting to classify a quantity into one of a multiple set of classes, what is usually done is a *One-Vs-All* approach. Essentially, the input is placed into each classifier, and a probability measure is computed. One-Vs-All simply states that out of all of the classifiers, whichever classifier generates the highest probability, the input is to be classified as the label that the particular classifier is trained for. In this case, the positive examples to train the m^{th} user-defined depth value in the first frame are those where the positive input examples into the classifier belong to the m^{th} class. Specifically, these are where the SIFT points have the user-defined depth of m, when examining where the SIFT point is located, and using the spatial locations to reference $\mathcal{H}_f$. The other input examples that do not share the same user-defined depth of m are deemed negative examples. In this case, the input space into the SVM classifiers are the 129 element vectors that were previously mentioned. The output of the SVM classifier will thus become a probability, which determines how well this 129 element vector belongs to the m^{th} class or user-defined depth value. For each user-defined depth, positive and negative examples are segregated in the same fashion, where positive examples are the 129 element vectors that belong to the corresponding class, while negative examples are the other SIFT points in the first frame. The outcome of this will generate D_f SVM classifiers, all trained within a One-Vs-All scheme. This same process is repeated for the last frame, where there are D_l user-defined unique depths assigned, and results in a total of D_l SVM classifiers, with N_2 SIFT points detected, and the grayscale values at each of these points are appended to create 129 element vectors to serve as input for training the classifiers. In Guttmann *et al.*'s work, these 129 element vectors are also known as *SIFT+Gray* features.

In Guttmann *et al.*'s work, many of the SVM training details are omitted. In addition, there are many publicly available tools that perform SVM training and classification, and have been proven to be highly effective. Guttmann *et al.* do not disclose which tool for SVMs they have used, nor do they mention if they implemented the SVM process themselves. As such, this thesis uses the *LIBSVM* toolbox [113], which is a well-known, open-source and publicly available library for training SVMs, performing classification, and has support for a variety of environments and programming languages. For training the SVMs, due to the nature of the input space into the SVMs, we use multilayer perceptron kernels to model the likelihood of inputs belonging to a certain class, and set the maximum number of iterations to 3000. Unfortunately, setting the maximum number of iterations varies between each video sequence, and so depending on what video sequence is used, 3000 iterations may not be enough for the training to converge. If this situation happens, the iterations are increased by increments of 1000 until the training converges.

1.1.2 Calibration Stage

The next stage of SVM training is to calibrate the classifiers, and adjust them so that classification can be as reliable as possible. In the training stage, the SVM classifiers are trained based on appearance alone. To account for high confidence classifications, a cross-validation step is performed by essentially swapping the input feature vectors to be placed in the classifiers of the other frames. In other words, the D_f classifiers from the first frame use the N_2 SIFT+Gray features detected in the last frame and are used as input. Similarly, the D_l classifiers from the last frame use the N_1 SIFT+Gray features detected in the first frame. The details regarding this calibration stage in Guttmann *et al.*'s work are rather ambiguous, and so this section describes the interpretation taken to implement this framework for this thesis.

Each SIFT+Gray feature from the last frame is run through each of the D_f classifiers from the first frame, thus resulting in D_f probability values for each SIFT+Gray feature from the last frame. The end result will generate N_2 $D_f \times 1$ vectors, where the m^{th} vector is the probability that each of the N_2 SIFT+Gray feature points is classified as the user-defined depth m from the first frame. We denote this set of vectors to be in the set $\mathcal{D} = \{\vec{D}_1, \vec{D}_2, \vec{D}_3, \ldots, \vec{D}_m, \ldots, \vec{D}_{D_f}\}$, where $\vec{D}_m$ is the $D_f \times 1$ probability vector for the user-defined depth m. It should be noted that the elements of each vector in $\mathcal{D}$ are sorted in ascending order, and the purpose of this will be apparent later. After, the "ground truth" labeling for the last frame in the calibration stage is created. To do this, the unique list of user-defined depths from the first frame are used in conjunction with the rough estimate of depths found in the last frame by just using the spatial and user-defined terms (refer to Section 1.1.1). The ground truth labeling is simply a new "hard" labeling, $\mathcal{H}'_l$, which is similar to $\mathcal{H}_l$, except that the unique list of labels consulted are not from the last frame, but the first frame instead. At each of the corresponding (x, y) location of each N_2 SIFT+Gray feature, their corresponding depth values are obtained by consulting $\mathcal{H}'_l$ at these locations, and these "ground truth" depth labels are denoted as $\mathcal{G}$. For each m^{th} classifier from the first frame, it is calibrated so that the *False Positive Rate (FPR)* for this classifier is less than 5%. Guttmann *et al.* also do not define what their definition of FPR is, and so it is interpreted in the following way. To calibrate each classifier by exploiting the FPR, for each probability in $\vec{D}_m$, this is used as a threshold so that any SIFT+Gray features for the last frame, when put through the SVM classifiers, should any exceed this threshold are classified as belonging to m from $\mathcal{G}$, while anything less does not belong to m from $\mathcal{G}$. By denoting A_1 to be the total number of SIFT+Gray feature points from the last frame that were classified as m from $\mathcal{G}$, that **should not** have been classified as m by consulting $\mathcal{G}$, and A_2 as the total number of SIFT+Gray feature points that were not classified as m, and also agrees by consulting $\mathcal{G}$, the FPR is defined as:

$$FPR = 100 \left(\frac{A_1}{A_1 + A_2} \right) \quad . \tag{1.11}$$

Each probability in $\vec{D}_m$ is tested by calculating the FPR, and a linear search is performed until a probability is found within $\vec{D}_m$ that produces a FPR of less than 5%. If such a probability cannot be found, then the equiprobable value of 0.5 is the probability used. As such, the m^{th} SVM classifier is calibrated so that when a SIFT+Gray feature point is put through this classifier, should the probability exceed this threshold, it is considered to belong to the user-defined depth m. Finally, this calibration stage is repeated in a similar fashion for the first frame, and each SIFT+Gray feature from the first frame is run through each of the D_l classifiers from the last frame. Essentially, the entire stage is repeated by swapping the roles of the first and last frames, and the calibration is performed on the first frame, using the scheme that was just mentioned.

1.1.3 Classification Stage

Once the calibration stage is finished, there are two sets of calibrated classifiers that now exist: One set from the first frame, and one from the last frame. The goal now is to search through each of the frames in the video sequence, *excluding* the first and last frame, as there are already user-defined strokes placed in these frames, and determine which points are anchor points. These are performed by going through each frame from the second frame to the second-last frame, detecting SIFT+Gray feature points for each frame, and running each of these feature points through all classifiers (both for the first and last frame). For each SIFT+Gray feature point detected in the i^{th} frame, running this feature point through each of the classifiers will result in $D_f + D_l$ possible probability values. To determine which user-defined depth value this SIFT+Gray feature point belongs to, the one with the highest probability is selected, but must pass the threshold of this classifier that was calculated by using the FPR from the calibration stage. Should this feature point surpass this threshold, this location is deemed an *anchor point*, and is used for solving for the final calculated depths. The actual value of the anchor point is the actual user-defined depth that the particular SVM classifier was trained on for the particular anchor point. The final output of this whole process is to produce a set of user-defined depths, sparsely located throughout the entire video volume, using the first and last frames as a guideline - the SVM training and classification stages help in producing such depths. However, as SVMs can inevitably misclassify values, should the anchor points become misclassified, this will have an impact in the final output, which will be demonstrated later in the results chapter (Chapter 5).

1.2 Build Sparse System

Once the appropriate anchor points have been found, another sparse linear system is created and solved. The sparse system is essentially the same as the one previously discussed in Section 1.1.1, but there are now additional terms to guide the solution. In addition to the spatial and user-defined constraints, there should also be a way to capture similarities of colours between patches, as colour similarity can also determine whether certain areas are uniform, and they would most likely correspond to the same depth. In actuality, this modeling is very similar to the Laplacian Matrix seen in Chapter 2.2. This concept was also borrowed from Levin *et al.* a means of artificially colourizing black and white images, given user-defined colours placed in areas around the image [114]. With the user-defined strokes, the objective was to colour in the rest of the image, given this user-defined information. The weights that are calculated are borrowed from this paper, which is also stated in [1]. Consulting the sparse matrix A in Section 1.1.1, more terms are required to be augmented into this system. The same linear indexing scheme is used, where the i^{th} row is mapped to the linear index l_i using the tuple (x, y, t), such that $l_i = \text{rows} \times \text{cols} \times t + x \times \text{cols} + y$. As such, the colour weights that are appended to the matrix A are done in the following fashion. Let u be a particular pixel at (x, y, t), and let v be a pixel that is connected to u in an 8-connected fashion. By recalling the graph connectivity in Chapter 2, a pixel can be connected to its 8 neighbouring pixels, and there are weights that connect to each of these neighbouring pixels with respect to the centre. These weights will model the colour similarity between the centre pixel and each of the neighbouring pixels in an 8-connected fashion. Also, we will account for the fact that the 8-pixel neighbourhood also includes the centre u, and the similarity should produce a normalized value of 1, as the exact same colour values are being compared to each other. To calculate the colour weight between

u and v, denoted as $\omega_{u,v}$, it is defined in the following way:

$$\omega_{u,v} = \begin{cases} 1 & \text{if } u = v \\ -\exp([||\vec{I}(u) - \vec{I}(v)||^2]/2\sigma^2)/M & \text{if } u \text{ is connected to } v \end{cases} . \tag{1.12}$$

M is a normalization factor, such that the sum of all the weights in the 8-pixel neighbourhood that surround u is 1. This ensures that all weights are subject to the range of $[0, 1]$. $\vec{I}(u)$ would be the RGB colour pixel value in the video volume at (x, y, t). Though a more perceptual colour space should be used here, Guttmann opt to use the RGB colour space, and to ensure an accurate implementation of their work, this should be done, and could be a reason as to why certain areas of the results do not agree with human perception. These results will be shown in the results chapter (Chapter 5). Essentially, the weights are calculated by using the L_2 norm of the difference between the two colour pixels, which are three-dimensional vectors, located at locations u and v. As such, we examine the 8 pixel neighbourhood that surrounds the pixel u, and compute the linear indices l_j by taking the row and column co-ordinates for those pixels v. In addition, we also compute the linear index for pixel u, which is l_i. For the l_i^{th} row, the l_i^{th} column will be assigned to 1, and for all of the indices l_j, we take each of their $\omega_{u,v}$ weights, and place their weights in their proper columns given by l_j. However, in [1], σ was not given, and for the purposes of this thesis, this parameter was empirically estimated to be 0.08, provided that the colour channels of the video frame are normalized to the range of $[0, 1]$.

As this is a video volume, not only should spatial information be accounted for, but temporal information should be used, in order for the results to be temporally consistent. To incorporate temporal information into the sparse matrix, the sparse system will depend on weights that reflect the amount of motion that exists between two consecutive frames. The logic behind this is that for corresponding pixels from one frame to the next, should the difference in appearance and in motion appear to be small, this should contribute more to the final solution, as estimating the depth is more reliable at these co-ordinates. As the motion becomes more complex, the weighting of these corresponding pixels should decrease. As stated by Guttmann *et al.* [1], the temporal weight is set to exploit the common characteristics of interplay between depth and motion in the scene, and is also a unique characteristic of the disparity estimation problem. In addition, the temporal weight also captures when objects are moving only horizontally, and thus have no change in perceived depth, or when objects are also moving closer or farther away from the camera. When objects undergo the latter condition, the depth should change dynamically. This is also true should the camera start to zoom in, or zoom out of the scene. Guttmann *et al.* state that they use optical flow estimation to compute these temporal weights. To capture this dynamic property of the objects, as well as the camera itself, the temporal weight at a tuple (x, y, t), defined as $W_M(x, y, t)$, is calculated in the following way. By letting $m_h(x, y, t-1)$ and $m_v(x, y, t-1)$ denote the amount of horizontal and vertical displacement between $(x, y, t-1)$ and (x, y, t) respectively, this weight is calculated in pixel units between two consecutive frames $t-1$ and t. $W_M(x,y,t)$ is thus:

$$W_M(x,y,t) = \begin{cases} 1 & \text{if } m_h(x,y,t-1) = 0 \text{ and } m_v(x,y,t-1) = 0 \\ 1 & \text{if } m_h(x,y,t-1) \neq 0 \text{ and } m_v(x,y,t-1) = 0 \\ 0.5 & \text{if } m_h(x,y,t-1) = 0 \text{ and } m_v(x,y,t-1) > 2 \\ 0.2 & \text{if } m_h(x,y,t-1) > 2 \text{ and } m_v(x,y,t-1) > 2 \end{cases} \tag{1.13}$$

In other words, $W_M(x,y,t)$ is set to 1 if there is no motion, or horizontal motion only, or it is set to 0.5 if the vertical motion exceeds a threshold of 2 pixels, or is set to 0.2 if there is horizontal and vertical motion above a threshold of 2 pixels. In optical flow algorithms, due to ambiguities, in most cases, motion will never be *exactly* zero. As Guttmann *et al.* did not clarify this in their work, we introduce a tolerance where any motion that is 0.01 pixels or less in any direction is considered to exhibit “no motion” in that direction. In addition, the work does not describe what optical flow estimation algorithm they use. As such, in order to be consistent, the edge-aware, temporally consistent optical flow method that is described in this thesis is the one that was ultimately used in the end. With these temporal weights, another linear equation is introduced to solve for each disparity in the video volume $d(x,y,t)$. This linear equation is defined as:

$$c_2 W_M(x,y,t)(d(x,y,t) - d(x,y,t-1)) = 0 \tag{1.14}$$

c_2 controls how much contribution the temporal term will contribute to the linear system. In Guttmann *et al.*’s work, this is set to 3. The last piece of information that needs to be appended into the system are the anchor points themselves. These need to be augmented into the vector b from Section 1.1.1 in the following way. Taking into account that the anchor points may produce some misclassifications, as the SVM classifiers are not perfect, these terms to be introduced into the sparse linear system of equations should be weighted relatively low, as this information should not contribute as much in determining the final disparities. This weight, c_4 is set to 10 in Guttmann *et al.*’s work. By defining $T(x,y,t)$ to be the detected anchor point at a tuple (x,y,t), the following equation is added for each user-defined pixel in the video volume:

$$c_4 d(x,y,t) = c_4 T(x,y,t) \tag{1.15}$$

For each location in the volume that is an anchor point, the tuple (x,y,t) is converted into its linear index l_i, with c_4 being appended in the l_i^{th} row and l_i^{th} column of A, while $c_4 T(x,y,t)$ being placed in the l_i^{th} row of b. To combine all of the terms together, for the l_i^{th} row of the sparse matrix A, given that the index l_i is created through the linear indexing equation, the following equation is placed in that sparse matrix.

$$c_1 W_E(x,y,t)(d(x,y,t) - d(x-1,y,t)) + c_1 W_E(x,y,t)(d(x,y,t) - d(x,y-1,t)) + \\ c_2 W_M(x,y,t)(d(x,y,t) - d(x,y,t-1)) + \sum_{v \in \mathcal{N}(u)} \omega_{u,v} = 0, \text{if } d(x,y,t) \text{ is unknown} \tag{1.16}$$

By re-arranging the above equation to collect like terms, we are thus left with:

$$(2c_1 W_E(x,y,t) + c_2 W_M(x,y,t))d(x,y,t) - c_1 W_E d(x-1,y,t) - c_1 W_E d(x,y-1,t) - \\ c_2 W_M(x,y,t)d(x,y,t-1) + \sum_{v \in \mathcal{N}(u)} \omega_{u,v} = 0, \text{if } d(x,y,t) \text{ is unknown} \tag{1.17}$$

Similarly, if the disparity at tuple (x, y, t) is known, this is defined as a user-defined depth, and the equation for placing into the sparse matrix is defined as:

$$c_1 W_E(x,y,t)(d(x,y,t) - d(x-1,y,t)) + c_1 W_E(x,y,t)(d(x,y,t) - d(x,y-1,t)) + \\ c_2 W_M(x,y,t)(d(x,y,t) - d(x,y,t-1)) + \sum_{v \in \mathcal{N}(u)} \omega_{u,v} + c_3 d(x,y,t) = c_3 V(x,y,t), \text{if } d(x,y,t) \text{ is user-defined} \tag{1.18}$$

Again, by re-arranging the above equation to collect like terms, we thus have:

$$(2c_1 W_E(x,y,t) + c_2 W_M(x,y,t) + c_3)d(x,y,t) - c_1 W_E d(x-1,y,t) - c_1 W_E d(x,y-1,t) - \\ c_2 W_M(x,y,t)d(x,y,t-1) \sum_{v \in \mathcal{N}(u)} \omega_{u,v} = c_3 V(x,y,t), \text{if } d(x,y,t) \text{ is known} \tag{1.19}$$

Finally, if the disparity value at tuple (x, y, t) is classified as an anchor point, the equation for placing into the sparse matrix is defined as:

$$c_1 W_E(x,y,t)(d(x,y,t) - d(x-1,y,t)) + c_1 W_E(x,y,t)(d(x,y,t) - d(x,y-1,t)) + \\ c_2 W_M(x,y,t)(d(x,y,t) - d(x,y,t-1)) + \sum_{v \in \mathcal{N}(u)} \omega_{u,v} + c_4 T(x,y,t) = c_4 T(x,y,t), \text{if } d(x,y,t) \text{ is an anchor point} \tag{1.20}$$

Again, re-arranging the above to collect like terms gives:

$$(2c_1W_E(x,y,t) + c_2W_M(x,y,t) + c_4)d(x,y,t) - c_1W_Ed(x-1,y,t) - c_1W_Ed(x,y-1,t) -$$
$$c_2W_M(x,y,t)d(x,y,t-1)\sum_{v\in\mathcal{N}(u)}\omega_{u,v} = c_4T(x,y,t), \text{if } d(x,y,t) \text{ is an anchor point} \tag{1.21}$$

Here, u is the pixel located at tuple (x,y,t), and $\mathcal{N}(v)$ are those pixels within an 8-pixel spatial neighbourhood that surrounds u. By letting l_i be the index defined for tuple (x,y,t), l_j be the index defined for tuple $(x-1,y,t)$, l_k be the index defined for tuple $(x,y-1,t)$ and l_l be the index defined for tuple $(x,y,t-1)$, for each l_i^{th} row of the matrix A, there will be coefficients for the $d(x,y,t)$, $d(x-1,y,t)$, $d(x,y-1,t)$ and $d(x,y,t-1)$ terms from the above equations, and are placed in the corresponding columns, referenced by positions of l_i, l_j, l_k and l_l. In addition, for the vector b, $c_3V(x,y,t)$ is placed in the l_i^{th} row if the depth at this pixel was user-defined, $c_4T(x,y,t)$ is placed in the l_i^{th} row if this pixel is an anchor point, and is 0 otherwise.

1.3 Solving for All Depths

Once the sparse matrix A has been constructed, as well as the vector b, the sparse system of equations is solved. As seen previously, the system is solved using Successive Overrelaxation, with the SOR weight set to $\omega = 0.4$. As there will be additional terms in the sparse matrix, the need for iterative methods is even more desired, as additional memory will be required, thus increasing the amount of memory that direct solvers would need to solve the system. The number of iterations is also set to 3000. Once the system has been solved, the vector of estimated disparities, $\vec{d}$, is reshaped into a sequence of maps that total the number of frames, where each of them are of the same horizontal and vertical dimensions. As the linear indexing function is one-to-one, the indices generated by these functions can simply be reversed in order to generate the disparities are the right locations of the tuples (x,y,t). These disparities are then used for DIBR which, in conjunction with the single-view 2D video, create the two-view stereoscopic counterpart of the single-view image or video to display on compatible hardware.

Appendix 2

List of Thesis Publications

The publications that were produced during the research and implementation of this thesis work were the following:

2.1 Conference Publications

- Raymond Phan, Dimitrios Androutsos, *'Edge-Aware Temporally Consistent SimpleFlow: Optical Flow without Global Optimization'*, **IEEE Conf. on Digital Signal Processing**, Santorini, Greece, Jul. 1 -37, 2013

- Raymond Phan, Dimitrios Androutsos, *'A Semi-Automatic 2D to Stereoscopic 3D Image and Video Conversion System in a Semi-Automated Segmentation Perspective'*, **International Society of Optical Engineering (SPIE) Electronic Imaging Conference**, Stereoscopic Displays and Applications XXIV Track, San Francisco, California, Feb. 4 - 7, 2013

- Raymond Phan, Richard Rzeszutek, Dimitrios Androutsos, *'Unconstrained 2D to Stereoscopic 3D Image and Video Conversion using Semi-Automatic Energy Minimization Techniques'*, **Society of Motion Picture and Television Engineers (SMPTE) Annual Technical Conference**, Hollywood, California, Oct. 22 - 25, 2012.

- Raymond Phan, Richard Rzeszutek, Dimitrios Androutsos, *'2D Art Painting to Stereoscopic 3D Conversion using Semi-Automatic Depth Map Generation'*, **Electronic Imaging & the Visual Arts 2012**, Florence, Italy, May 9 - 11, 2012.

- Mohammad Fawaz, *Raymond Phan*, Richard Rzeszutek, Dimitrios Androutsos, *'Adaptive 2D to 3D Image Conversion using a Hybrid Graph Cuts and Random Walks Based Approach'*, **IEEE Intl. Conference on Acoustics, Speech and Signal Processing (ICASSP) 2012**, Kyoto, Japan, Mar. 25 - 30, 2012.

- Raymond Phan, Richard Rzeszutek, Dimitrios Androutsos, *'Semi-Automatic 2D to 3D Image Conversion Using Scale-Space Random Walks and a Graph Cuts Based Depth Prior'*, **IEEE Intl. Conference on Image Processing (ICIP) 2011**, Brussels, Belgium, Sep. 11 - 14, 2011.

- Richard Rzeszutek, *Raymond Phan.* Dimitrios Androutsos, *'Semi-automatic Synthetic Depth Map Generation For Video Using Random Walks'*, **IEEE Intl. Conference on Multimedia and Expo (ICME) 2011**, Barcelona, Spain, July 11 - 15, 2011.

- Raymond Phan, Richard Rzeszutek, Dimitrios Androutsos, *'Semi-Automatic 2D to 3D Image Conversion Using A Hybrid Random Walks and Graph Cuts Based Approach'*, **IEEE Intl. Conference on Acoustics, Speech and Signal Processing (ICASSP) 2011**, Prague, Czech Republic, May 22 - 27, 2011.

2.2 Journal Publications

- Raymond Phan and Dimitrios Androutsos, *'Robust Semi-Automatic Depth Map Generation in Unconstrained Images and Video Sequences for 2D to Stereoscopic 3D Conversion'*, *accepted to appear in* **IEEE Transactions on Multimedia**. Reference Number: MM-004919-R1

2.3 Other Publications

- Antoinette Mercurio, *'Electrical engineering student receives prestigious Vanier Canada Graduate Scholarship'*, **Ryerson Today**, May 21st, 2010. (*This article profiles my research, and the winning of the Vanier Canada Graduate Scholarship*) - `http://www.ryerson.ca/news/news/General_Public/20100521_vanier.html`

- Ryerson University - Everyone Makes a Mark Campaign Video Short - `http://www.ryerson.ca/marks/ray/index.html` (*A video short about my research and the Vanier Canada Graduate Scholarship*)

- Andy Lee, *'As Hollywood goes 3-D, video processing algorithms help make conversion of films more efficient, faster'*, **Ryerson Research News & Events**, Feb. 20th, 2009. (*This article features the 3D research performed in our laboratory*) - `http://www.ryerson.ca/news/news/Research_News/20090220_Androutsos.html`

References

[1] M. Guttmann, L. Wolf, and D. Cohen-Or, "Semi-automatic Stereo Extraction from Video Footage," *Proc. IEEE Intl. Conf. on Computer Vision (ICCV)*, 2009.

[2] I. E. Richardson, *The H.264 Advanced Video Compression Standard.* Wiley, 2010.

[3] R. Ebert, "Why I hate 3-D and you should too," *Online: Newsweek. http://www.thedailybeast.com/newsweek/2010/04/30/why-i-hate-3-d-and-you-should-too.html - Written on April 30, 2010. Accessed on March 1, 2013*, 2010.

[4] Quantel, "3d films drive box office sales," *Online. http://blog.quantel.eu/2010/11/3d-film/ - Written on November 17, 2010. Accessed on February 22, 2013*, 2010.

[5] K. Elliot and D. Hutchison, "Stereoscopic Architecture of 3-d Ready DLP-based HDTVs," *Proc. ACM SIGGRAPH 2008 Workshop on Immersive Projection Technologies / Emerging Display Technologies*, no. 12, 2008.

[6] "Texas Instruments - DLP 3D HDTVs," *http://www.dlp.com/hdtv/dlp-features/3d-hdtv.aspx - Accessed on April 24th, 2013.*

[7] "IMAX Corporation," *http://www.imax.com - Accessed on April 24th, 2013.*

[8] nVidia, "NVIDIA 3D Vision," *Online: http://www.nvidia.ca/object/3d-vision-main.html - Accessed on March 25, 2013*, 2013.

[9] E. Trucco and A. Verri, *Introductory Techniques for 3-D Computer Vision.* Prentice-Hall Publishers, 1998.

[10] R. Klette, K. Achluns, and K. Schluns, *Computer Vision - Three-Dimensional Data from Images.* Springer Science Publishers, 1998.

[11] M. Lang, A. Hornung, O. Wang, S. Poulakos, A. Smolic, and M. Gross, "Nonlinear disparity mapping for stereoscopic 3D," *ACM Trans. on Graphics*, vol. 29, no. 3, pp. 10–19, 2010.

[12] T. Yan, R. W. H. Lau, Y. Xu, and L. Huang, "Depth Mapping for Stereoscopic Videos," *Intl. Jnl. Comp. Vis.*, vol. 102, pp. 293–307, 2013.

[13] D. Scharstein and R. Szeliski, "A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms," *Intl. Jnl. Comp. Vis.*, vol. 47, pp. 7–42, 2002.

[14] L. Zhang, W. J. Tam, and D. Wang, "Stereoscopic image generation based on depth images," *Proc. IEEE ICIP*, pp. 2993–2996, 2004.

[15] C. Fehn, R. de la Barre, and S. Pastoor, "Interactive 3-DTV Concepts and Key Technologies," *Proc. of the IEEE*, vol. 94, no. 3, pp. 524–538, March 2006.

[16] A. Redert, M. O. de Beeck, C. Fehn, W. Ijsselsteijn, M. Pollefeys, L. V. Gool, E. Ofek, I. Sexton, and P. Surman, "ATTEST: Advanced Three-Dimensional Television System Technologies," *IEEE Intl. Symp. on 3D Data Processing Visualization and Transmission (3DPVT)*, June 2002.

[17] Mobile3DTV, "Mobile 3DTV Content Delivery Optimization over DVB-H System," *http://sp.cs.tut.fi/mobile3dtv/ - Accessed on April 4th, 2013.*

[18] Z.-W. Gao, W.-K. Lin, and Y.-S. Shen, "Design of Signal Processing Pipeline for Stereoscopic Cameras," *IEEE Trans. on Consumer Electronics*, vol. 56, no. 2, pp. 324–331, May 2010.

[19] H. Wang, Y. Yang, L. Zhang, Y. Yang, and B. Liu, "2D-to-3D Conversion Based on Depth-from-Motion," *Proc. IEEE Intl. Conf. on Mechatronic Science, Electric Engineering and Computer*, 2011.

[20] J. Feng, H. Huang, Y. Zheng, and W. Zhu, "Stereoscopic 3D Conversion from 2D Video in H.264/AVC Compressed Domain," *Proc. IEEE Fourth Intl. Conf. on Image and Signal Processing*, 2011.

[21] L. Liu, C. Ge, N. Zheng, Q. Li, and H. Yao, "Spatio-temporal Adaptive 2D to 3D Video Conversion for 3DTV," *Proc. IEEE Intl. Conf. on Consumer Electronics*, 2012.

[22] Y. Chen, R. Zhang, and M. Karczewicz, "Low-complexity 2D to 3D Video Conversion," *Proc. SPIE Electronic Imaging: Stereoscopic Displays and Applications XXII*, vol. 7863, pp. 78 631I–1–78 631I–9, 2011.

[23] W. J. Tam, C. Vazquez, and F. Speranza, "Three-dimensional TV: A Novel Method for Generating Surrogate Depth Maps using Colour Information," *Proc. SPIE Electronic Imaging - Stereoscopic Displays and Applications XX*, 2009.

[24] Y. Feng, J. Ren, and J. Jiang, "Object-Based 2D-to-3D Video Conversion for Effective Stereoscopic Content Generation in 3D-TV Applications," *IEEE Trans. on Broadcasting*, vol. 57, no. 2, pp. 500–509, June 2011.

[25] T.-Y. Kuo, Y.-C. Lo, and C.-C. Lin, "2D-to-3D Conversion for Single-View Image Based on Camera Projection Model and Dark Channel Model," *Proc. IEEE ICASSP*, pp. 1433–1436, 2012.

[26] Z. Wang, R. Wang, S. Dong, W. Wu, L. Huo, and W. Gao, "Depth Template Based 2D-to-3D Video Conversion and Coding System," *Proc. IEEE ICME*, pp. 308–315, 2012.

[27] K. Han and K. Hong, "Geometric and Texture Cue Based Depth-map Estimation for 2D to 3D Image Conversion," *Proc. IEEE Intl. Conf. on Consumer Electronics*, pp. 651–654, 2011.

[28] J. Zhang, Y. Yang, and Q. Dai, "A Novel 2D-to-3D Scheme by Visual Attention and Occlusion Analysis," *Proc. IEEE 3DTVCON*, 2011.

[29] F. Yu, J. Liu, Y. Ren, J. Sun, Y. Gao, and W. Liu, "Depth Generation Method for 2D to 3D Conversion," *Proc. IEEE 3DTVCON*, 2011.

[30] Z. Zhang, Y. Wang, T. Jiang, and W. Gao, "Visual Pertinent 2D-to-3D Video Conversion by Multi-Cue Fusion," *Proc. IEEE ICIP*, pp. 909–912, 2011.

[31] L. Schnyder, O. Wang, and A. Smolic, "2D to 3D Conversion of Sports Content using Panoramas," *Proc. IEEE ICIP*, 2011.

[32] J. Caviedes and J. Villegas, "Real Time 2D to 3D Conversion: Technical and Visual Quality Requirements," *Proc. IEEE Intl. Conf. on Consumer Electronics*, 2011.

[33] S.-F. Tsai, C.-C. Chung, C.-T. Li, and L.-G. Chen, "A Real-Time 1080p 2D-to-3D Video Conversion System," *IEEE Trans. on Consumer Electronics*, vol. 57, no. 2, pp. 915–922, May 2011.

[34] E. Ramos-Diaz, V. Kravchenko, and V. Ponomaryov, "Efficient 2D to 3D Video Conversion Implemented on a DSP," *EURASIP Journal on Advances in Signal Processing*, vol. 106, 2011.

[35] Y.-K. Lai, Y.-F. Lai, and Y.-C. Chen, "An Effective Hybrid Depth-Perception Algorithm for 2D-to-3D Conversion in 3D Display Systems," *Proc. IEEE Intl. Conf. on Consumer Electronics*, 2012.

[36] J. Konrad, G. Brown, M. Wang, P. Ishtar, C. Wu, and D. Mukherjee, "Automatic 2D-to-3D Image Conversion using 3D Examples from the Internet," *Proc. SPIE Electronic Imaging: Stereoscopic Displays and Applications XXIII*, vol. 8288, pp. 82 880F–1–82 880F–12, 2012.

[37] J. Konrad, M. Wang, and P. Ishtar, "2D-to-3D Image Conversion by Learning Depth from Examples," *Proc. 3D Cinematography Workshop (3DCINE'12) at IEEE CVPR*, pp. 16–22, 2012.

[38] M. Park, J. Luo, A. Gallagher, and M. Rabbani, "Learning to Produce 3D Media from a Captured 2d Video," *Proc. ACM Multimedia*, pp. 1557–1560, 2011.

[39] J. J. Koenderink, A. J. van Doorn, A. M. Kappers, and J. T. Todd, "Ambiguity and the 'mental eye' in pictorial relief," *Perception*, vol. 30, no. 4, pp. 431–448, 2001.

[40] C. Cortes and V. Vapnik, "Support-Vector Networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

[41] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Intl. Jnl. Comp. Vis.*, vol. 60, no. 2, pp. 91–110, 2004.

[42] O. Wang, M. Lang, M. Frei, A. Hornung, A. Smolic, and M. Gross, "StereoBrush: Interactive 2D to 3D Conversion using Discontinuous Warps," *Proc. Eighth Eurographics Symp. on Sketch-Based Interfaces and Modeling*, pp. 47–54, 2011.

[43] R. Phan and D. Androutsos, "Content-Based Retrieval of Logo and Trademarks in Unconstrained Color Image Databases using Color Edge Gradient Co-occurrence Histograms," *Elsevier CVIU*, vol. 114, no. 1, pp. 66–84, 2010.

[44] K. N. Plataniotis and A. N. Venetsanopoulos, *Color Image Processing and Applications.* Springer-Verlag: Berlin, 2000.

[45] L. Grady, "Random Walks for Image Segmentation," *IEEE TPAMI*, vol. 28, no. 11, pp. 1768–1783, 2006.

[46] Y. Boykov and G. Funka-Lea, "Graph Cuts and Efficient N-D Image Segmentation," *Intl. Jnl. of Comp. Vis.*, vol. 2, no. 70, pp. 109–131, 2006.

[47] E. N. Mortensen and W. A. Barrett, "Intelligent Scissors for Image Composition," *Proc. ACM SIGGRAPH*, pp. 191–198, 1995.

[48] A. Gururajan, H. Sari-Sarraf, and E. Hequet, "Interactive Texture Segmentation via IT-SNAPS," *Proc. IEEE SSIAI*, pp. 129–132, 2010.

[49] G. Friedland, K. Jantz, and R. Rojas, "SIOX: Simple Interactive Object Extraction in Still Images," *Proc. IEEE ISM2005*, p. 253259, 2005.

[50] Y. Boykov, O. Veksler, and R. Zabih, "Fast Approximate Energy Minimization via Graph Cuts," *IEEE TPAMI*, vol. 23, no. 11, pp. 1222–1239, 2002.

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

[52] Y. Saad, *Iterative Methods for Sparse Linear Systems, Second Edition.* Society for Industrial and Applied Mathematics, 2003.

[53] R. Rzeszutek, T. El-Maraghi, and D. Androutsos, "Image Segmentation using Scale-Space Random Walks," *Proc. Intl. Conf. DSP*, pp. 1–4, July 2009.

[54] K. Kelly and D. Judd, "Color Universal Language and Dictionary Names," *National Bureau of Standards, Washington DC, USA: Publication 440, U.S. Government Printing Office*, 1976.

[55] A. Criminisi, P. Perez, and K. Toyama, "Object Removal by Exemplar-based Inpainting," *Proc. IEEE Conf. CVPR*, vol. 2, pp. II–721 – II–728, June 2003.

[56] "FFmpeg," *http://www.fmpeg.org - Accessed on August 11, 2013.*

[57] L. Wang, W. Hu, and T. Tan, "Recent Developments in Human Motion Analysis," *Pattern Recognition*, vol. 36, no. 3, pp. 585–601, 2003.

[58] D. Ramanan, D. A. Forsyth, and A. Zisserman, "Tracking People by Learning their Appearance," *IEEE Trans. on PAMI*, vol. 29, no. 1, pp. 65–81, 2007.

[59] P. Buehler, M. Everingham, D. P. Huttenlocher, and A. Zisserman, "Long Term Arm and Hand Tracking for Continuous Sign Language TV Broadcasts," *Proc. British Machine Vision Conference*, 2008.

[60] S. Birchfield, "Elliptical Head Tracking using Intensity Gradients and Color Histograms," *Proc. IEEE CVPR*, 1998.

[61] M. Isard and A. Blake, "CONDENSATION - Conditional Density Propagation for Visual Tracking," *Intl. Jnl. Comp. Vis*, vol. 29, no. 1, pp. 5–29, 1998.

[62] C. Bibby and I. Reid, "Robust Real-Time Visual Tracking using Pixel-Wise Posteriors," *Proc. British Machine Vision Conference*, 2008.

[63] ——, "Real-Time Tracking of Multiple Occluding Objects using Level Sets," *Proc. IEEE CVPR*, 2010.

[64] B. D. Lucas and T. Kanade, "An Iterative Image Registration Technique with an Application to Stereo Vision," *Proc. Intl. Joint Conf. on AI*, vol. 81, pp. 674–679, 1981.

[65] D. Comaniciu, V. Ramesh, and P. Meer, "Kernel-Based Object Tracking," *IEEE Trans. PAMI*, vol. 25, no. 5, pp. 564–577, 2003.

[66] J. Shi and C. Tomasi, "Good Features to Track," *Proc. IEEE CVPR*, 1994.

[67] I. Matthews, T. Ishakawa, and S. Baker, "The Template Update Problem," *IEEE Trans. PAMI*, vol. 26, no. 6, pp. 810–815, 2004.

[68] N. Dowson and R. Bowden, "Simultaneous Modeling and Tracking of Feature Sets," *Proc. IEEE CVPR*, 2005.

[69] A. Rahimi, L. P. Morency, and T. Darrell, "Reducing Drift in Differential Tracking," *Elsevier CVIU*, vol. 109, no. 2, pp. 97–111, 2008.

[70] A. D. Jepson, D. J. Fleet, and T. F. El-Maraghi, "Robust Online Appearance Models for Visual Tracking," *IEEE Trans. PAMI*, vol. 25, no. 10, pp. 1296–1311, 2003.

[71] A. Adam, E. Rivlin, and I. Shimshoni, "Robust Fragments-based Tracking using the Integral Histogram," *Proc. IEEE CVPR*, pp. 798–805, 2006.

[72] M. J. Black and A. D. Jepson, "Eigentracking: Robust Matching and Tracking of Articulated Objects using a View-Based Representation," *Intl. Jnl. Comp. Vis.*, vol. 26, no. 1, pp. 63–84, 1998.

[73] D. Ross, J. Lim, R. Lin, and M. Yang, "Incremental Learning for Robust Visual Tracking," *Intl. Jnl. Comp. Vis*, vol. 77, pp. 125–141, 2007.

[74] J. Kwon and K. M. Lee, "Visual Tracking Decomposition," *Proc. IEEE CVPR*, 2010.

[75] M. Yang, Y. Wu, and G. Hua, "Context-Aware Visual Tracking," *IEEE Trans. PAMI*, vol. 31, pp. 1195–1209, 2009.

[76] H. Grabner, J. Matas, L. V. Gool, and P. Cattin, "Tracking the Invisible: Learning Where The Object Might Be," *Proc. IEEE CVPR*, 2010.

[77] S. Avidan, "Support Vector Tracking," *IEEE Trans. PAMI*, vol. 26, no. 8, pp. 1064–1072, 2004.

[78] R. Collins, Y. Liu, and M. Leordeanu, "Online Selection of Discriminative Tracking Features," *IEEE Trans. PAMI*, vol. 27, no. 10, pp. 261–271, 2005.

[79] S. Avidan, "Ensemble Tracking," *IEEE Trans. PAMI*, vol. 29, no. 2, pp. 1631–1643, 2007.

[80] H. Grabner and H. Bischof, "On-line Boosting and Vision," *Proc. IEEE CVPR*, 2006.

[81] B. Babenko, M.-H. Yang, and S. Belongie, "Visual Tracking with Online Multiple Instance Learning," *Proc. IEEE CVPR*, 2009.

[82] H. Grabner, C. Leistner, and H. Bischof, "Semi-Supervised On-line Boosting for Robust Tracking," *Proc. ECCV*, 2008.

[83] F. Tang, S. Brennan, Q. Zhao, and H. Tao, "Co-tracking using Semi-Supervised Support Vector Machines," *Proc. IEEE ICCV*, pp. 1–8, 2007.

[84] Q. Yu, T. B. Dinh, and G. Medioni, "Online Tracking and Reacquisition using Co-trained Generative and Discriminative Trackers," *Proc. ECCV*, 2008.

[85] Z. Kalal, J. Matas, and K. Mikolajczyk, "Online Learning of Robust Object Detectors during Unstable Tracking," *Proc. IEEE ICCV - 3rd On-Line Learning for Comp. Vis. Workshop*, 2009.

[86] ——, "P-N Learning: Bootstrapping Binary Classifiers by Structural Constraints," *Proc. IEEE CVPR*, 2010.

[87] Z. Kalal, K. Mikolajczyk, and J. Matas, "Tracking-Learning-Detection," *IEEE TPAMI*, vol. 6, no. 1, pp. 1–14, 2010.

[88] S. C. Chapra and R. P. Canale, *Numerical Methods for Engineers, 6th Edition.* McGraw-Hill: New York City, NY, 2010.

[89] M. Ozuysal, P. Fua, and V. Lepetit, "Fast Keypoint Recognition in Ten Lines of Code," *Proc. IEEE CVPR*, 2007.

[90] J. Luo and D. Crandall, "Robust Color Object Detection using Spatial-Color Joint Probability Functions," *IEEE TIP*, vol. 15, no. 6, pp. 1443–1453, 2006.

[91] S. Goferman, L. Zelnik-Manor, and A. Tal, "Context-Aware Saliency Detection," *Proc. IEEE CVPR*, pp. 2376–2383, 2010.

[92] B. K. P. Horn and B. G. Schunck, "Determining optical flow," *Artificial Intelligence*, vol. 17, pp. 185–203, 1981.

[93] S. Baker, D. Scharstein, J. P. Lewis, S. Roth, M. J. Black, and R. Szeliski, "A Database and Evaluation Methodology for Optical Flow," *Intl. Jnl. Comp. Vis.*, vol. 92, no. 1, pp. 1–31, 2011.

[94] K. Souhila and A. Karim, "Optical Flow based robot obstacle avoidance," *Intl. Jnl. Adv. Robotic Sys.*, vol. 4, no. 1, pp. 13–16, 2007.

[95] K. Mukherjee, "Joint optical flow motion compensation and video compression using hybrid vector quantization," *Proc. IEEE DCC*, 1999.

[96] M. Werlberger, T. Pock, M. Unger, and H. Bischof, "Optical flow guided tv-l^1 video interpolation and restoration," *Proc. Eighth Intl. Conf. on EMMCVPR*, pp. 273–286, 2011.

[97] D. B. Goldman, C. Gonterman, B. Curless, D. Salesin, and S. Seitz, "Video object annotation, navigation, and composition," *Proc. ACM UIST*, pp. 3–12, 2008.

[98] Y. Boykov, O. Veksler, and R. Zabih, "Fast approximate energy minimization via graph cuts," *IEEE Trans. PAMI*, vol. 23, no. 11, pp. 1–18, 2001.

[99] J. Pearl, "Fusion, Propagation and Structing in Belief Networks," *Artificial Intelligence*, vol. 29, no. 3, pp. 241–288, 1986.

[100] V. Kolomogorov, "Convergent Tree-Reweighted Message Passing for Energy Minimization," *IEEE Trans. PAMI*, vol. 28, no. 10, pp. 1568–1583, 2006.

[101] M. Wainwright, T. Jaakkola, and A. Willsky, "MAP Estimation via Agreement on Trees: Message-Passing and Linear Programming," *IEEE Trans. Information Theory*, vol. 51, no. 11, pp. 3697–3717, 2005.

[102] B. Glocker, H. Heibel, N. Navab, P. Kohli, and C. Rother, "Triangleflow: Optical Flow with Triangulation-based Higher-Order Likelihoods," *Proc. ECCV*, 2010.

[103] S. Roth and M. Black, "Fields of Experts: A Framework for Learning Image Priors," *Proc. IEEE CVPR*, 2005.

[104] V. Lempitsky, S. Roth, and C. Rother, "Fusionflow: Discrete-Continuous Optimization for Optical Flow Estimation," *Proc. IEEE CVPR*, 2008.

[105] P. Gwosdek, H. Zimmer, S. Grewenig, A. Bruhn, and J. Weickert, "A highly efficient GPU implementation for variational optic flow based on the Euler-Lagrange Framework," *Proc. ECCV CVGPU Workshop*, 2010.

[106] N. Sundaram, T. Brox, and K. Keutzer, "Dense point trajectories by GPU-accelerated large displacement optical flow," *Proc. ECCV*, 2010.

[107] M. W. Tao, J. Bai, P. Kohli, and S. Paris, "SimpleFlow: A non-iterative, sublinear optical flow algorithm," *Computer Graphics Forum - Eurographics 2012*, vol. 31, no. 2, 2012.

[108] S. Paris and F. Durand, "A fast approximation of the bilateral filter using a signal processing approach," *Intl. Jnl. of Comp. Vis.*, vol. 81, no. 1, pp. 24–52, 2009.

[109] E. S. L. Gastal and M. M. Oliveira, "Domain transform for edge-aware image and video processing," *ACM Trans. on Graphics*, vol. 30, no. 4, pp. 69:1–69:12, 2011.

[110] M. Lang, O. Wang, T. Aydin, A. Smolic, and M. Gross, "Practical temporal consistency for image-based graphics applications," *ACM Trans. on Graphics*, vol. 31, no. 4, pp. 34:1–34:8, 2012.

[111] S. Vijayanarasimhan and K. Grauman, "Active Frame Selection for Label Propagation in Videos," *Proc. ECCV*, 2012.

[112] A. Vedaldi and B. Fulkerson, "VLFeat: An open and portable library of computer vision algorithms," *Online: http://www.vlfeat.org/index.html - Accessed on March 16, 2013*, 2008.

[113] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, pp. 27:1–27:27, 2011, software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[114] A. Levin, D. Lischinski, and Y. Weiss, "Colorization using Optimization," *ACM Trans. on Graphics*, vol. 23, no. 3, pp. 689–694, 2004.