1-1-2015

Discrete and Continuous Optimization for Motion Estimation

Ryan Kennedy
University of Pennsylvania, kenryd@gmail.com

Discrete and Continuous Optimization for Motion Estimation

Abstract
The study of motion estimation reaches back decades and has become one of the central topics of research in computer vision. Even so, there are situations where current approaches fail, such as when there are extreme lighting variations, significant occlusions, or very large motions. In this thesis, we propose several approaches to address these issues. First, we propose a novel continuous optimization framework for estimating optical flow based on a decomposition of the image domain into triangular facets. We show how this allows for occlusions to be easily and naturally handled within our optimization framework without any post-processing. We also show that a triangular decomposition enables us to use a direct Cholesky decomposition to solve the resulting linear systems by reducing its memory requirements. Second, we introduce a simple method for incorporating additional temporal information into optical flow using "inertial estimates" of the flow, which leads to a significant reduction in error. We evaluate our methods on several datasets and achieve state-of-the-art results on MPI-Sintel. Finally, we introduce a discrete optimization framework for optical flow computation. Discrete approaches have generally been avoided in optical flow because of the relatively large label space that makes them computationally expensive. In our approach, we use recent advances in image segmentation to build a tree-structured graphical model that conforms to the image content. We show how the optimal solution to these discrete optical flow problems can be computed efficiently by making use of optimization methods from the object recognition literature, even for large images with hundreds of thousands of labels.

Degree Type
Dissertation

Degree Name
Doctor of Philosophy (PhD)

Graduate Group
Computer and Information Science

First Advisor
Camillo J. Taylor

Keywords
Computer Vision, Motion, Optical Flow

Subject Categories
Computer Sciences

DISCRETE AND CONTINUOUS OPTIMIZATION FOR MOTION ESTIMATION

Ryan Kennedy

A DISSERTATION

in

Computer and Information Science

Presented to the Faculties of the University of Pennsylvania

in

Partial Fulfillment of the Requirements for the

Degree of Doctor of Philosophy

2015

Supervisor of Dissertation

Camillo J. Taylor, Professor, Computer and Information Science

Graduate Group Chairperson

Lyle Ungar, Professor, Computer and Information Science

Dissertation Committee

Kostas Daniilidis, Professor, Computer and Information Science

Jean Gallier, Professor, Computer and Information Science

Jianbo Shi, Professor, Computer and Information Science

Richard Szeliski, Microsoft Research

DISCRETE AND CONTINUOUS OPTIMIZATION FOR MOTION ESTIMATION

Acknowledgements

This thesis would not exist if not for the help of many people along the way. First, a sincere thanks to my advisor C.J. Taylor. Working with him then has been incredibly productive and fun. He seems to always have new ideas that he's eager to share and his support in all aspects of my work at Penn has been astounding. He has my sincere gratitude for all his help.

I'd also like to thank Jianbo Shi, with whom I worked for several years after joining Penn. Before starting grad school, I had no knowledge of computer vision whatsoever, and Jianbo was able to show me how interesting computer vision really is. He has been a fantastic teacher and I owe much of my success in computer vision to what he taught me.

Thanks as well to Kostas Daniilidis, with whom I worked with for a time and who has been supportive all along, and is also the head of my thesis committee. I'd also like to thank Jean Gallier, with whom I worked with on a paper during my first year at Penn and who is also a member of my thesis committee. Jean is certainly one of the most fun people I have had to chance to work with, with his unique and wonderful combination of mathematical rigour and silly jokes. Finally, thanks also to the external member of my thesis committee, Rick Szeliski, for his time and help.

I've also had the great opportunity to collaborate with Laura Balazano, now at the University of Michigan, and I have learned a huge amount working with her. I would also not even be at Penn were it not for the initial help of Lyle Ungar and my undergraduate advisor Rob Knight in my admission. There are many others at Penn as well that have my gratitude, including Charity Payne and Mike Felker

for their huge administrative help, Vijay Kumar for briefly working with me at the start of my program, and Ben Taskar and Mitch Marcus for whom I was a teaching assistant.

I also would like to thank my colleagues, including David, Katerina, Cody, Weiyu, Kosta, and many others, who made my time at Penn very enjoyable. Finally, thanks to my family for their unwavering, constant support.

ABSTRACT

DISCRETE AND CONTINUOUS OPTIMIZATION FOR MOTION ESTIMATION

Ryan Kennedy

Camillo J. Taylor

The study of motion estimation reaches back decades and has become one of the central topics of research in computer vision. Even so, there are situations where current approaches fail, such as when there are extreme lighting variations, significant occlusions, or very large motions. In this thesis, we propose several approaches to address these issues. First, we propose a novel continuous optimization framework for estimating optical flow based on a decomposition of the image domain into triangular facets. We show how this allows for occlusions to be easily and naturally handled within our optimization framework without any post-processing. We also show that a triangular decomposition enables us to use a direct Cholesky decomposition to solve the resulting linear systems by reducing its memory requirements. Second, we introduce a simple method for incorporating additional temporal information into optical flow using "inertial estimates" of the flow, which leads to a significant reduction in error. We evaluate our methods on several datasets and achieve state-of-the-art results on MPI-Sintel. Finally, we introduce a discrete optimization framework for optical flow computation. Discrete approaches have generally been avoided in optical flow because of the relatively large label space that makes them computationally expensive. In our approach, we use recent advances in image segmentation to build a tree-structured graphical model that conforms to the image content. We show how the optimal solution to these discrete optical flow problems can be computed efficiently by making use of

optimization methods from the object recognition literature, even for large images with hundreds of thousands of labels.

Table of Contents

List of Tables

List of Figures

1

Introduction

In one of the more iconic scenes from the film *Jurassic Park*, two children Lex and Tim Murphy are trapped in their jeep during a rainstorm while a *Tyrannosaurus rex* attacks them. Coming to their rescue, Dr. Grant runs to the jeep and tells them to stay still: "*Don't move. He can't see us if we don't move.*" To the viewers' relief, the plan works, and the fearsome dinosaur is unable to locate them despite being just a few feet away.

Of course, the idea that *T. Rex* was unable to separate people form the background without the aid of motion is simply untrue. In fact, the *T. Rex* likely had exceptional vision (Stevens, 2006), although this would have resulted in a much shorter movie with a less happy ending. Even so, it is certainly true that motion can provide essential information for our understanding of the visual world. A similar situation to that of *Jurassic Park* can be seen through the camouflage that animals use in nature. Consider, for example, Figure 1.1a, which depicts a lion hidden in tall grass as it stalks its prey. From static visual information alone, it may be extremely difficult to tell that the lion is there. In this case, motion can indeed prove to be an extremely valuable cue for detecting predators. Another scene where motion is important is shown in Figure 1.1b. In this street scene,

(A) A lion hiding in the grass. (Dupont, 2014)

(B) A street scene involving pedestrians. (Wang et al., 2007)

FIGURE 1.1: Scenes where motion is an important cue. In (A), the tall grass obscures a lion. Without motion, the lion may be difficult to see. In (B), the motions of the pedestrians are necessary for determining where to walk down the street.

pedestrians are walking down the street. In order for us to walk down the street ourselves, it is necessary to determine the motion of each person and plan a path to avoid them. Situations such as this that involve path planning around moving objects show up in robotics and autonomous driving applications.

Motion can play an essential role for scene understanding even in extremely simple situations, as was convincingly demonstrated in an experiment by Ullman (Ullman, 1979). In his experiment, Ullman constructed constructed two cylinders that were invisible aside from a set of random dots on their surfaces. One cylinder was placed within the other and they were rotated in opposite directions. This setup is shown in Figure 1.2. Only the orthographic projection of the cylinders from the side was made visible, and so at any point in time the image on the screen appeared to be just a random collection of dots. However, when the cylinders were rotated, the motion resulted in the visual interpretation of two rotating cylinders. As Ullman

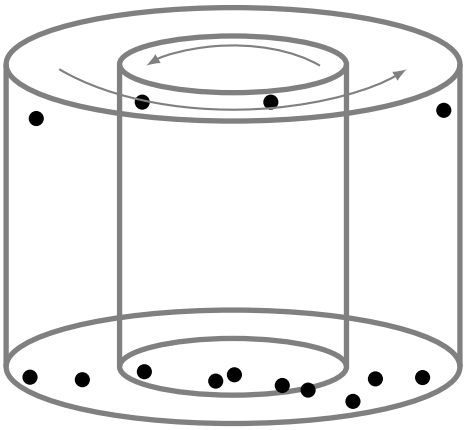


FIGURE 1.2: The setup of the cylinder used in Ullman's experiment (Ullman, 1979). Two concentric transparent cylinders rotate in opposite directions. The cylinders are depicted in gray, although they are were not visible in the experiment. The random dots on the cylinder appear as just a collection of random dots at a single frame, but as the cylinders rotate the correct object segmentation and 3D structure is perceived. Figure adapted by author from (Ullman, 1979).

put it,

> *[W]hen the changing projection was viewed, the elements in the motion across the screen were perceived as two rotating cylinders whose shapes and angles of rotation were easily determined. Both the segmentation of the scene into objects and the 3-D interpretation were based in this case on the motion alone, since each single view contained no information concerning the segmentation or the structure.*

This demonstration shows that, indeed, motion is a very important cue for visual perception. Motion aids in our understanding of the structure and semantic meaning of scenes.

Unfortunately, motion cues can also fool us. Consider the classic barber's pole illusion, as shown in Figure 1.3. As the pole spins, our perception is that the pole has a vertical component to its motion even though the pole is only spinning

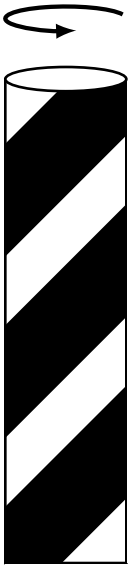
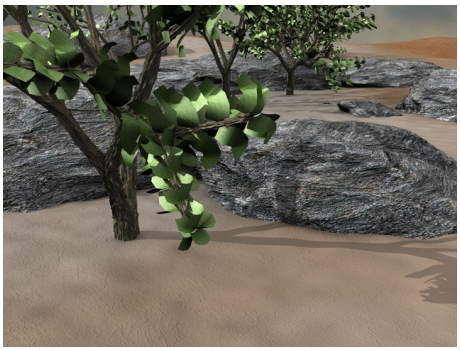


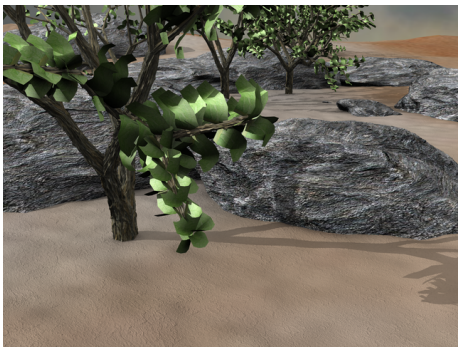
FIGURE 1.3: The barber's pole illusion. As the pole spins around its vertical axis, it appears that the motion of the pole has a vertical component when there is none. The is due to the aperture problem.

around its vertical axis. An illusion of this type demonstrates that, even with motion, a visual interpretation of a scene can be ambiguous. In this case, the illusion is caused by the fact that is only possible to estimate local motion that is perpendicular to an edge, which is known as the *aperture problem*. In reality, the barber pole could be moving one of many different directions based on our limited view of it, and our minds choose just one of these interpretations. Thus, the underlying algorithms that we use to estimate motion – and their associated biases, both implicit and explicit – play a significant role in our resulting interpretation of the world.

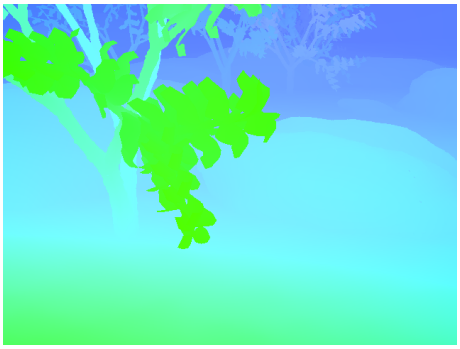
The barber's pole illusion also illustrates the difference between the true *motion field* and the *apparent motion* or *optical flow* of a scene. While the true motion of the pole is a rotation around its vertical axis, the apparent motion is ambiguous and has a vertical component. Similarly, a homogeneous textureless sphere that



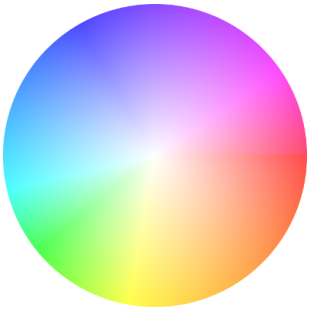
(A) Image 1 of the Grove2 sequence.



(B) Image2 of the Grove2 sequence.



(C) Groundtruth optical flow.



(D) Color key for groundtruth optical flow values. Direction is indicated by hue while magnitude is indicated by saturation of the colors.

FIGURE 1.4: An example of the optical flow problem, taken from the Middlebury dataset (Baker et al., 2011)

is spinning may have a very large motion but impart zero optical flow. Computationally, images are all that we have access to and so the optical flow of a scene is all that we can estimate. Its estimation is the topic of this thesis.

An example of optical flow estimation on a real image is shown in Figure 1.4. Given a sequence of images, the goal of optical flow estimation is to determine the offset that each pixel in the image at time t moves to at time $t + 1$. We display the estimated flow field using a color image, where the hue at each pixel indicates the direction of the motion and the saturation indicates the magnitude.

1.1 Applications of Optical Flow

We begin by outlining several concrete applications of optical flow to further motivate the issues studied in this thesis.

1.1.1 Video interpolation

While a video may be pre-recorded and fixed, in many cases it is desirable to look at the scene from a viewpoint – in either time or space – that is not captured at any of the frames of the video. In (Zitnick et al., 2004), a method was proposed that used motion estimation between a set of cameras to interpolate to novel viewpoints in the scene. In (Mahajan et al., 2009), a method closely related to optical flow was used to obtain high-quality interpolation between frames in a single video.

1.1.2 Tracking

Optical flow has an clear application to tracking: if an accurate motion field can be obtained, then tracking an object can be done just by tracking each associated point using the estimated flow. Local methods for optical flow have also been used to directly track individual points (Shi and Tomasi, 1994). Features derived from flow fields can also be used indirectly as features within a more complex tracking system, such as for tracking a human pose (Sapp et al., 2011; Fragkiadaki et al., 2013).

1.1.3 Action recognition

When attempting to estimate the actions that occur in a video, motion can be an important cue. For example, in (Wang et al., 2011; Bhattacharya, 2013; Fathi and Rehg, 2013), dense motion estimates were used to generate features for action recognition.

1.1.4 Relationship to stereo correspondence

Optical flow is also closely related to the stereo correspondence problem. In contrast to optical flow where two frames are typically of the same scene but separated in time, stereo uses two images of a scene from the same point in time but different spatial locations. This results in a slightly different problem, since the scene is completely rigid with respect to the stereo images and all motion is due to the camera. For every pixel, the set of possible correspondences in the other image are no longer the set of all pixels in the other image since the correspondence must lie on an epipolar line (Szeliski, 2010; Hartley and Zisserman, 2003). If the images are rectified first (Hartley and Zisserman, 2003), then this is equivalent to finding only the horizontal offset for each pixel, rather than both horizontal and vertical as is the case for optical flow. In this way, optical flow methods can be applied directly to the stereo problem as well. However, because the problem is somewhat different, a different set of algorithms that include the use of discrete optimization (Boykov et al., 2001; Kolmogorov and Zabih, 2001) or segmentation (Yang et al., 2009; Zitnick et al., 2004) also perform well.

1.1.5 Relationship to general image correspondence

Optical flow is also related to the general image correspondence problem. In this setup, a correspondence is estimated between two semantically similar images which may not be from the same scene. Because the images may be from different scenes, color or image intensity are not useful features, and more complex descriptors such as SIFT (Lowe, 2004; Liu et al., 2011) need to be used. The models for image correspondence may also need to account for difficulties such as scale changes (Hassner et al., 2012; Kim et al., 2013a), but in general the approach is similar to that of optical flow (Liu et al., 2011). One particularly useful application of image correspondence is for label transfer, where labels from one image can be transfered to another by aligning an image to a pre-labeled database (Liu et al., 2009). This approach can also be used in medical imaging (Kybic and Unser, 2003) to align a given medical image to a model image in order to analyze it or detect irregularities.

1.2 Modeling Optical Flow

The basic problem of optical flow can be stated simply: for each pixel in an image, determine its corresponding location in a subsequent image. To gain tractability on solving this problem, a set of simplifying assumptions are used to mathematically formalize the idea of what constitutes a "good" solution. Optical flow models generally rely on two assumptions about objects within our physical world. First, it is assumed a physical object maintains a similar appearance through time – at least over a short time span. Second, it is assumed that physical objects tend to have a finite, compact extent, which implies that the motion of nearby points are

correlated; points that are very close to each other in the image are likely to be part of the same object and have a similar motion due to the motion of that object. These two assumptions – that objects maintain a similar appearance through time and that nearby points have correlated motions – will be referred to as the *brightness constancy* assumption and the *smoothness* assumption. Although the term "brightness constancy" originally comes from models where the images have only a single, grayscale "brightness" channel, we may refer to this assumption as the idea that objects have a similar apperance through time, even if that appearance is measured using color or more complex features.

1.2.1 Brightness Constancy Assumption

The simplest form of brightness constancy states that for grayscale images, two corresponding pixels have a similar intensity value. This is easily written as a mathematical constraint, and forms the basis of all optical flow algorithms. In an ideal situation, this is the only constraint that is needed: each pixel can be corresponded with its closest match in the next image. However, simple pixel intensities are often insufficiently discriminative and the optical flow problem is underconstrained. Instead, more information can be used, such as color or even more complex features. Additionally, a smoothness assumption can be used as a regularization in order to impose more structure on the resulting motion fields.

1.2.2 Smoothness Assumption

Due to the coherence of objects in our world, two points near each other in an image will often have a correlated motion. This can be encoded mathematically

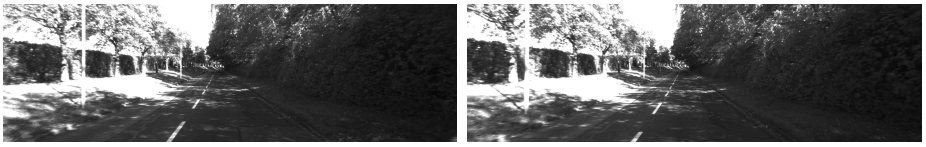
in different ways, and boradly splits algorithms into two categories: *local* and *global* methods. In local algorithms, the smoothness assumption assumes that a separate local parametric model exists to describe the motion of each pixel individually. This allows for information to be aggregated over local neighborhoods in the image. The motion of each pixel is then solved for independently. In contrast, global algorithms define a single, global cost function which encodes both the brightness constancy assumption as well as a penalty for when nearby pixels have significantly different motion estimates. A global cost function is generally more difficult to solve since it is often non-convex and cannot be as easily parallelized as local cost functions. However, incorporating information globally, over the entire image domain, can result in much more accurate solutions. Modern optimization methods have improved the speed and reliability of global methods, and the most accurate optical flow algorithms today typically use some form of global optimization.

1.3 Challenges

The basic brightness constancy and smoothness assumptions are not hard to model mathematically, and in many cases simple approaches are able to produce high-quality motion estimates relatively efficiently. However, these basic assumptions can break down.

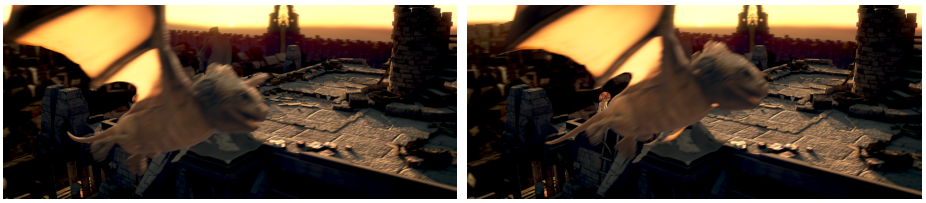
1.3.1 Lighting Variation

Lighting variations lead to violations of the brightness constancy assumption. For example, a pixel in one image might be covered by a shadow in the subsequent



(A) A pair of images from the KITTI dataset (Geiger et al., 2012). Shadows cause violations of the brighness constancy constraint and low contrast grayscale images make feature matching difficult.

(B) A pair of images from the MPI-Sintel dataset (Butler et al., 2012). Large motions result in motion blur, making matching difficult.



(C) A pair of images from the MPI-Sintel dataset (Butler et al., 2012). Large occluded regions have no match in the second frame.

FIGURE 1.5: An example of the optical flow problem.

image. This is a major problem in, for example, the KITTI dataset as shown in Figure 1.5a, where a combination of low-contrast grayscale features and outdoor scenes results in gross violations of brightness constancy. Even when color images are used, however, differences in lighting can cause significant changes in color.

1.3.2 Imaging and Atmospheric Effects

The imaging process itself is imperfect, and this can lead to violations in brightness constancy as well. Again, in the KITTI dataset as shown in Figure 1.5a, the cameras produce only low-contrast grayscale images. In the MPI-Sintel dataset (Butler et al., 2012), other imaging effects such as motion blur can be problematic,

as shown in Figure 1.5b. A related problem is caused by atmospheric effects, where fog or glare can prevent the camera from capturing the true color of the object represented at each pixel.

1.3.3 Occlusions

The brightness constancy assumption has an underlying assumption that each point in an image has a correct correspondence in the other image. When objects are occluded, this is no longer the case. Instead, points may go out of frame or be blocked by an occluding object, and brightness constancy can not possibly be used in these regions to find an accurate solution. Occlusions also cause problems for the smoothness assumption. If multiple objects are moving in the same image, then pairs of pixels on the border between objects or an object and the background may refer to two completely different positions in 3D space, even though they are adjacent in the image. The motions of these pixels are no longer correlated, and assuming that the motion field is smooth across these occlusion boundaries will give the wrong solution. Figure 1.5c shows a pair of frames exhibiting a large occluded regions for which no matches are present in the second frame.

1.3.4 Complex Motions

The simplest form of the smoothness assumption simply penalizes adjacent pixels that have different motions. More complex motions, however, will violate this. For example, articulated or non-rigid objects – such as a waving flag or a deforming piece of paper – are penalized by this smoothness constraint. Additionally, even if objects are rigid, a simple smoothness constraint assumes that objects move

parallel to the image plane. This assumption is heavily violated in the KITTI dataset (Geiger et al., 2012), where the camera is mounted to a forward-moving car and so pixels move significantly more as they are closer to the camera. This is also seen as a change in scale for objects that move towards or away from the camera. Large motions can also cause issues for local optimization methods (Brox and Malik, 2011).

1.3.5 The Aperture Problem

The aperture problem pervades all optical flow datasets, regardless of the type of image or model used. This problem is demonstrated in Figure 1.6. The problem is that any region of an image has only a limited view. Consider an image region that views the center of a textureless, white piece of paper. If the paper is shifted slightly, the image does not change! In this case, there is no possible way to determine the motion from that region alone. Similarly, for an image region centered on a line, only motions *perpendicular* to the line can be determined. The aperture problem is commonly seen in barbershop poles (Figure 1.3). The apparent motion of a barber pole is exactly the same if the pole is moving vertically or spinning. Our mind chooses a single interpretation, but the motion is ambiguous. The aperture problem is especially difficult for local methods, where information in the image is aggregated over only a small area. Global methods, in contrast, are able to propogate the motion from areas that are highly-discriminative to those that are more ambiguous.

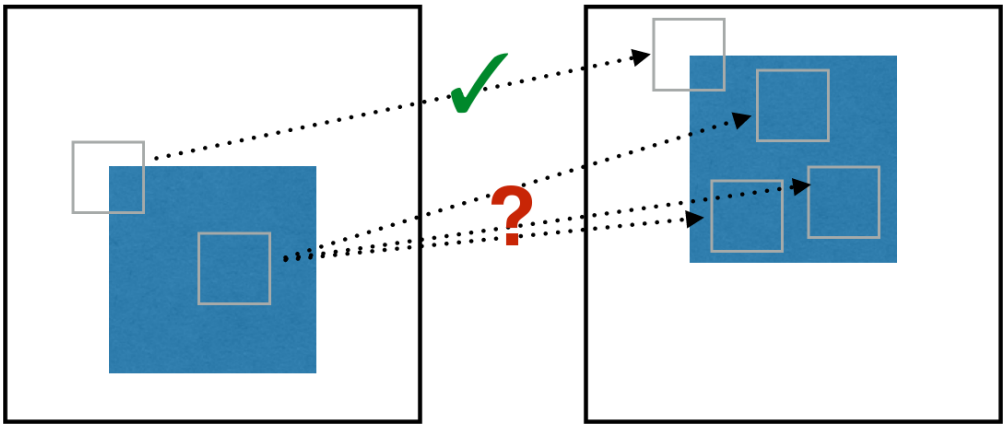


FIGURE 1.6: A depiction of the aperture problem. For a local corner region, the motion of the associated pixel can be reliably matched the second image. However, for non-corner regions, such as those within the rectangle, the motion is ambiguous.

1.3.6 Optimization

A slightly different problem is how a model should be optimized. For example, the set of possible motions that can be assigned to a pixel can be thought of as either a continuous or a discrete space, and different methods with different advantages and disadvantages can be applied depending on the choice. Within both discrete and continuous approaches, there are a large number of possible optimization methods: gradient descent, Newton's method, mean-field methods, message passing, etc. The choice of optimization methods has expanded and improved significantly in the last few years, and it depends largely on the choice of model. There is a tradeoff here as well: more complex models may be more realistic and more closely relate to the data, but they are often more difficult to optimize. Any optical flow method must decide where in this tradeoff to position themselves and how to optimize their model, depending on the application.

1.4 Contributions of this thesis

This thesis provides three main contributions. First, we present a novel optical flow model which is based on a triangulation of the image domain. The basis of our model is similar to past approaches in that it involves estimating the motion of an image by imposing both data and smoothness terms over a discretization of the image. However, in our model we view the triangles as discrete geometric pieces over which motion is estimated. We show how this triangulation allows for occlusions to be directly and naturally incorporated into the optimization problem. This approach is continuous in nature, and we use Newton's method to find a local optimum, which is typically avoided due to computational reasons. In fact, we show how a triangulated image allows for an exact Cholesky factorization to be used within Newton's method by reducing the memory requirements of Cholesky factorization.

Second, we introduce the idea of *inertial estimates* of optical flow. Inertial estimates are estimates of image motion that are taken from adjacent frames. These inertial estimates can be fused using a classifier, resulting in significant improvements in accuracy. This method is a simple, effective approach to using temporal information from a video sequence. When used together with our triangulated model, this results in state-of-the-art optical flow estimates on the difficult MPI-Sintel dataset.

Finally, we present a discrete approach to motion estimation. Discrete optimization is frequently used in related domains such as stereo correspondence, and has the advantage that a near-optimal solution can be found in many situations. However, discrete methods are rarely used in optical flow due to the much larger

label space that makes many discrete approaches impractical. We propose a novel method that combines a hierarchical Markov random field with optimization techniques from the literature on object detection. We show that this problem can be solved *optimally* even for label spaces with hundreds of thousands of labels. We also show how this discrete approach allows for inertial estimates to be easily added, which reduces the runtime requirements for their use.

The outline of this thesis proposal is as follows. In Chapter 2, we review current methods for optical flow and how they relate to our approach. Our triangulation-based model is then presented in 3. In Chapter 4, inertial estimates are proposed and we show how they can be fused using a classifier into a single motion estimate. We also evaluate our triangulation-based model with the inertial estimates, which results in a state-of-the-art optical flow algorithm. Our discrete approach to optical flow is then proposed and evaluated in Chapter 5. Finally, we conclude in Chapter 6.

2

Preliminaries and Related Work

In this chapter, we review related work in motion estimation to provide context for the contributions of this thesis. We begin by presenting a brief outline of a very common approach to optical flow estimation. We then review modern techniques and how they differ from this model with a focus on approaches that are related to the algorithms proposed in subsequent chapters.

2.1 Basic Approach to Optical Flow Estimation

A fundamental assumption made in optical flow is the *brightness constancy* assumption, which says that a pixel's appearance remains relatively constant between frames. Formally, let $I(x, y, t)$ be a time-dependent image sequence, such that our goal is to estimate the motion from $I(x, y, t)$ to $I(x, y, t+1)$. The brightness constancy assumption then says that

$$I(x, y, t) \approx I(x + u, y + v, t + 1);, \tag{2.1}$$

where u and v are the estimated horizontal and vertical displacements at each pixel. Mathematically, we can write this as a constraint

$$I(x+u, y+v, t+1) - I(x, y, t) = 0 \tag{2.2}$$

or as an ℓ_2 penalty $[I(x+u, y+v, t) - I(x, y, t)]^2$.

If we linearize this constraint by taking its first order Taylor expansion around the current estimate, we have

$$I_x u + I_y v + I_t = 0 \ , \tag{2.3}$$

where I_x, I_y and I_t are the partial derivatives of I with respect to x, y and t, respectively. To aid in its interpretation, we rewrite this constraint as

$$\frac{\begin{bmatrix} I_x & I_y \end{bmatrix}}{\sqrt{I_x^2 + I_y^2}} \begin{bmatrix} u \\ v \end{bmatrix} = \frac{-I_t}{\sqrt{I_x^2 + I_y^2}} \ . \tag{2.4}$$

This constraint is a line, and it says that the displacement vector $\begin{bmatrix} u & v \end{bmatrix}$ is perpendicular to the image gradient $\begin{bmatrix} I_x & I_y \end{bmatrix}$ with a projected magnitude of $\frac{-I_t}{\sqrt{I_x^2+I_y^2}}$ in that direction. This is depicted in Figure 2.1. From the figure, it is clear that the linearized brightness constancy equation is insufficient to determine the displacement at that pixel uniquely. Indeed, only the component of the flow orthogonal to the image gradient can be determined. This is a mathematical depiction of the aperture problem.

Because a single constraint at each pixel leads to an under-constrained system of equations, it is necessary to impose regularization in order to obtain a unique solution. This regularization is done by incorporating a *smoothness* assumption into the model. This assumption is based on the idea that the physical world is

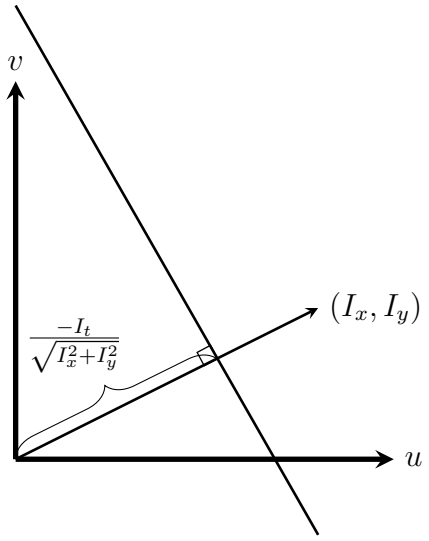


FIGURE 2.1: Graphical depiction of the aperture problem due to the linearized brightness constancy constraint. The standard linearized brightness assumption constraints the displacement vector $[u, v]$ to be on a line perpendicular the image gradient at that point. Thus, the displacement is undefined from only one such equation.

divided into distinct, compact, solid objects and thus pixels that are nearby in the image likely belong to the same object and have a similar motion. This constraint was used by Lucas and Kanade (Lucas et al., 1981), who impose the constraint that *the motion field is constant in a small region around each pixel*. Then, for each pixel, there are now multiple equation rather than just one. For example, using a 5×5 neighborhood around each pixel results in 25 equations, still with only 2 unknowns. The linear system is now over-determined, and a least-squares solution can be found using standard linear solvers. However, this solution is only valid for small displacements since the brightness constancy equation was linearized, and so the equations are re-linearized around the new solution estimate. This process is repeated until convergence.

The Lucas Kanade method works well in many cases, and the local approach has computational benefits: solving small linear systems can be done extremely efficiently and the optimization problem at each pixel can be parallelized. The

local nature of this approach makes it useful in other contexts as well, such as object tracking (Shi and Tomasi, 1994), where a dense motion field is not required and tracking discriminative points on an object can be done independently from each other. There are, however, drawbacks of this approach. First, the aperture problem (see Section 1.3.5) forces us to use neighborhoods that are sufficiently large so that they include local texture information. However, larger neighborhoods also have a greater chance of including irrelevant points that are not part of the object. This tradeoff on neighborhood size is known as the *generalized aperture problem* (Jepson and Black, 1993). One hybrid solution is to have a contextual neighborhood that varies from pixel to pixel (Baker and Matthews, 2004; Mei et al., 2011). Still, the neighborhoods may be inaccurate, and the Lucas-Kanade approach often has errors in regions without sufficient texture. An error analysis for this and similar problems can be found in (Kearney et al., 1987).

Another drawback of this approach is the sum-of-squares cost function. This ℓ_2 cost function disproportionately penalizes outliers, which can be a problem when the brightness constancy constraint is violated or a pixel is occluded. One approach to combat this is to use a robust cost function which is less susceptible to outliers, as proposed in (Cohen, 1993; Black and Anandan, 1996). This approach is commonly used in modern methods and can produce high-quality results.

Many extensions of this local method have been proposed to improve its performance (Baker and Matthews, 2004; Simoncelli et al., 1991). However, the local nature of the computation limits the accuracy of local models. Instead, it would be useful to define a single global cost function across the entire image using variational methods. This was the approach taken by Horn and Schunck (Horn and Schunck, 1981). Rather than assume a constant flow within a small neighborhood of each pixel, Horn and Schunck assumed that the gradient of the flow estimate is

smooth. The cost function is then

$$E(u,v) = \iint \left(I_x u + I_y v + I_t\right)^2 + \lambda \left(\|\nabla u\|^2 + \|\nabla v\|^2\right)\ dx\ dy\ . \tag{2.5}$$

The penalty here on the gradient of the motion estimates ensures that the flow field is locally smooth across the entire image domain. Because the brightness constancy constraint has been linearized, the Horn-Schunck functional is purely quadratic for each linearization and can be optimized efficiently using an iterative update found by solving the Euler-Lagrange equations. The advantage of this approach over the local method of Lucas-Kanade can be seen by considering what happens in image locations where the gradients are not diverse enough to overcome the aperture problem. In such locations, the smoothness constraint of Horn-Schuck will propagate motion estimates from neighboring regions with more texture and more certain flow estimates.

The Horn-Schunck method here relies on sequential linearizations of the cost function. This iteration will work well only when the image changes smoothly and the global solution can be reached using gradient descent optimization. In many cases, the image is noisy and motions are large, which introduces many local optima into the cost surface and prevent convergence to a good flow estimate. A common way to deal with this is to use a coarse-to-fine optimization. First, an image pyramid is generated (Burt and Adelson, 1983), which results in a series of images at decreasing resolutions. As the image resolution decreases, the high-frequency components of the image are removed and only larger structures remain. In addition, because the images are lower resolution, the motions become smaller between images. The optimization then proceeds by beginning at the coarsest end of the pyramid, propagating the estimated flow values as an initialization to the

next level of the pyramid, and continuing until all levels have been processed. In this way, the algorithm begins by finding large flow values and iteratively refining the estimate to incorporate more high-frequency information (Anandan, 1989).

At each level of this coarse-to-find optimization procedure, convergence is obtained by iteratively linearizing the cost function around the current solution and solving the resulting linear system. This inner linear system can be solved by standard linear techniques, such as successive over-relaxation (SOR), which is itself an iterative method. Direct methods such as Cholesky factorization have not been often used in global optical flow problems due to memory issues.

This general framework – a global cost function incorporating a brightness and smoothness constraint within a coarse-to-fine optimization – forms the basis of most modern approaches to optical flow. Recent examples of this framework are given in (Brox and Malik, 2011) and (Sun et al., 2010a). In the remainder of this chapter, we review modern techniques and how they deviate from this model, focusing on methods related to the contributions of this thesis.

2.2 Robustifying brightness constancy to lighting variations

The brightness constancy assumption is violated when there are lighting variations between the two images. This has been addressed in the literature by altering the brightness constancy assumption. In (Negahdaripour, 1998; Seitz and Baker, 2009), a multiplicative and additive term were added to the brightness constancy

equation, changing it from

$$I(x, y, t) = I(x + u, y + v, t + 1) \tag{2.6}$$

to

$$I(x, y, t)m(x, y) + a(x, y) = I(x + u, y + v, t + 1) \ . \tag{2.7}$$

In order to make the problem well-defined, it was further assumed that $m(x, y)$ and $a(x, y)$ vary smoothly over the image domain by penalizing the gradient of these terms. Thus, this allows for locally-smooth variations in image brightness.

Another approach is to use a different feature than brightness. In (Brox et al., 2004), it was assumed that brightness *gradients* remain constant, rather than just intensities. This was extended to other derivative features including Laplacian and Hessian constancy in (Papenberg et al., 2006). These various constancy constraints were simply combined using a weighted sum. However, it is better – although more complicated – to adjust the weighting based on the location in the image. For example, in well-lit image regions a standard brightness constancy might be preferable since it is less affected by noise than higher-order derivatives, while in regions that have brightness changes a gradient-based term may be best. In (Xu et al., 2012), the brightness and gradient constancy terms were weighted differently at each image location, where the weights evolved as a function of the relative data costs, allowing the algorithm to select the best-performing features at each point. A similar approach was taken in (Kim et al., 2013b), where features were combined based on the "discriminitability" of the features at a given point.

Another approach to dealing with lighting variations is to use a normalized cross-correlation (NCC) cost function. The NCC is a *patch*-based cost function, where the brightness at a pixel is normalized by the local mean and standard deviation of

brightness. NCC was shown to give good results in datasets where lighting variations are prevalent (Steinbrücker et al., 2009; Vogel et al., 2013) . Images can also be made more robust to lighting variations by altering how they are represented. In (Werlberger et al., 2009; Wedel et al., 2009b), the image is first decomposed into a "structure" and a "texture" component using denoising algorithms, and more weight is given to the texture component which is less affected by shadows and imaging irregularities. A color image can also converted into HSV or Lab color space (Zimmer et al., 2009, 2011), which allows for the lightness channel to be separately penalized from the color channels.

Violations in brightness constancy can also be overcome by changing the features that are used. For example, census transforms (Zabih and Woodfill, 1994) encode local difference information and have been successfully applied to optical flow problems, especially those involving low-contrast images (Vogel et al., 2013; Yamaguchi et al., 2013; Vogel et al., 2014; Yamaguchi et al., 2014). More complex features such as SIFT (Lowe, 2004) and HOG (Dalal and Triggs, 2005) have also been used to provide robustness while remaining discriminative. These features have mainly been used as a supplementary feature-matching term rather than a replacement of the data cost (Brox and Malik, 2011; Xu et al., 2012; Weinzaepfel et al., 2013), although in (Leordeanu et al., 2013; Revaud et al., 2014) a more extreme matching-based approach is used where sparse matches are interpolated into a dense motion field and subsequently refined using continuous optimization. One difficulty of these approaches is that they suffer from a similar generalized aperture problem as was seen in the Lucas-Kanade algorithm since they are neigborhood-based features: if the neighborhood is too small then the features are not discriminative enough, while if it is too large then the features may cover irrelevant background objects or multiple motions. Several recent methods

have attempted to overcome this issue using modified descriptors. In (Weinzaepfel et al., 2013), a SIFT-like descriptor was proposed where the distance function allows for a deformation of the descriptor itself to allow for variation in the motion in the descriptor's neighborhood. Another approach was taken by (Byrne and Shi, 2013), where the basis of the descriptor are nested circles that are all centered on the same image pixel, along with robust non-metric matching cost that allow for explicit removal of outliers.

One approach – which can be applied to any method regardless of the descriptor used – is to make the cost function *robust* to outliers, as was proposed in (Cohen, 1993; Black and Anandan, 1996). The Horn-Schunck method uses a standard least-squares ℓ_2 data cost function, which implicitly models all noise as Gaussian. Outliers can be handled by using a more robust cost function. The ℓ_1 cost function, for example, remains convex but allows to discontinuous flow estimates. Even more robust, non-convex functions can also be used. These robust methods have been shown to work very well on a variety of problems (Sun et al., 2010a; Werlberger et al., 2009; Zach et al., 2007), and this is a standard approach in modern methods.

2.3 Robustifying the smoothness constraint

An obvious drawback of the Horn-Schunck method is that the quadratic penalty on the image gradient results in a very smooth motion field, even across boundaries in the image where sharp discontinuities exist. In (Nagel and Enkelmann, 1986), Nagel and Enkelmann proposed an *oriented* smoothness constraint that reduces the smoothness constraint in directions perpendicular to the image gradient, allowing for sharper discontinuities at image edges. This approach has been used in modern algorithms such as (Alvarez et al., 2002). A variation on this is given in

(Zimmer et al., 2009, 2011; Alvarez et al., 2002), where rather than adjusting the smoothness constraint to not cross *image* edges, the constraint was made to not cross *flow* edges. This change results in a data and smoothness term that work well together and produce quite accurate results. Another approach to modifying the smoothness constraint is to use more parametric models, such as splines (Szeliski and Coughlan, 1994; Szeliski and Shum, 1996).

A related approach is to modify the weight of each smoothness term in the discretized cost function to avoid smoothing across image boundaries (Wedel et al., 2009a; Alvarez Leon et al., 1999; Xu et al., 2012). This approach was extended to non-local smoothness terms in (Werlberger et al., 2010; Bao et al., 2014; Krähenbühl and Koltun, 2012). In this case, each pixel is connected to a set of neighboring pixels where the neighborhood is defined by the similarity and distance to the center pixel. This both reduces the amount of smoothing along image edges as well as increases the smoothness between non-adjacent pixels that are nonetheless likely to have the same motion. We take a similar approach in our own algorithm in Chapter 3, where we use a triangulation that allows us to naturally have non-local smoothness constraints with no additional computation.

A very common approach in modern methods is to use a robust cost function rather than the sum of squared differences (Black and Anandan, 1996; Sun et al., 2010a). This is the same approach that was taken to make the data cost robust to lighting variations, and can be seen as a general method for making any cost function robust to outliers.

2.4 Large displacements

If displacements are not too large in optical flow problems, a standard coarse-to-fine image pyramid can be used. However, if the motion of an object between frames is significantly larger than the size of the object itself, then a coarse-to-fine approach will not work. The reason for this is that as the image is downsized in the image pyramid, large displacements will become smaller. For very large displacements with small objects, by the time the displacement is small enough the object has been completely blurred out and local optimization will fail. This was pointed out in (Brox and Malik, 2011). To overcome this issue, (Brox and Malik, 2011) proposed incorporating *global feature matching* into the variational optical flow problem. In their framework, (Brox and Malik, 2011) had a discrete matching term that encouraged features to be globally matched to similar matches, as well as a term that encouraged the flow estimate to be similar to the global feature matches. In this way, the global feature matching pushes the solution towards the true global optimum. The addition of feature matching as an additional unary data term has been used often in modern methods (Xu et al., 2012; Weinzaepfel et al., 2013; Chen et al., 2013). Another method to avoid oversmoothing small objects is to "explode" an intensity feature vector into a histogram of values so that only location information is smoothed and not feature values, as was presented in (Sevilla-Lara et al., 2014). We incorporate a feature matching term into our own algorithm using SIFT features in Chapter 3.

A different method, which uses no image pyramid, was proposed in (Steinbrucker et al., 2009). They proposed splitting the flow into two separate estimate, u, v and

u', v', using the cost function

$$E(u, v, u', v') = \iint E_{data}(u, v) + \lambda_0 E_{smooth}(u', v') + \lambda_1 \left[(u - u')^2 + (v - v')^2\right] \, dx \, dy, \tag{2.8}$$

where the last term encourages the flow estimate u, v to be similar to u', v'. Now, if $E_{data}(\cdot)$ is convex ((Steinbrucker et al., 2009) use an ℓ_1 cost), then $E(\cdot)$ can be optimized globally with respect to u', v' using standard gradient descent algorithms. If u', v' are held constant, then u, v can be solved for using a global matching procedure which can be parallelized on GPUs efficiently. The method then proceeds by alternately estimating u, v and u', v' until convergence. The parameter λ_1 begins as a small value and is gradually increased, encouraging the two flow estimates to eventually come to a compromise between the data and smoothness terms. Although this method is computationally interesting due to its unique optimization that does not involve an image pyramid, it has not seen much use since its publication. One of the algorithms presented in this thesis (Chapter 5) is related in that it also computes optical flow without the need for an image pyramid. In addition, our method can compute the *global* optimum of the full model.

Another approach to dealing with large displacements was proposed in (Barnes et al., 2009) and is known as PatchMatch. The PatchMatch algorithm is a stochastic method that involves three steps. First, patches are randomly assigned to corresponding matches in the second image. Next, good matches are propagated to their neighboring pixels. Finally, each pixel performs a local search for a better match. This process is repeated until convergence. While PatchMatch has no guarantees of optimality, it is useful in that it is extremely efficient and produces good results. This method was generalized in (Barnes et al., 2010) to deal with

changes in scale, rotations, and more general data costs. PatchMatch was shown to work well for optical flow algorithms in (Bao et al., 2014), which used PatchMatch along with non-local smoothness constraints and resulted in an algorithm with very good performance on difficult optical flow problems while taking only a fraction of a second to compute.

2.5 Occlusions

Occlusions often cause difficulties in optical flow algorithms. Most easily, they can be handled using a robust cost function and simply treated as a violation of the brightness constancy assumptions (Black and Anandan, 1996). However, for image sequences with large occluded regions, the occlusions are no longer just sparse outliers and are better handled by explicitly detecting and handling them. This is demonstrated in Figure 2.2, which shows how occlusions can be a very significant issue for some datasets.

A simple way to locate these occluded regions is using a *forward-backward consistency check*, as in (Proesmans et al., 1994). This check is based on the idea that if a pixel is occluded in the second image, the first image will not have a match while if the flow were estimated in the reverse direction a match might exist. These locations that are only matched in one direction are considered occluded. This was extended by (Alvarez et al., 2002) where a symmetrical cost function was defined within which a forward-backward check is used.

A disadvantage of the forward-backward approach is that the computation is immediately doubled since flow estimates must be computed in two directions. Another approach is to note that pixels are occluded if multiple pixels map to the

(A) First image (B) Second image

(C) Groundtruth flow (D) Groundtruth occlusions

FIGURE 2.2: Example of the difficulties that occlusion causes. For difficult image sequences, occlusions may involve a large section of the image and cannot be dismissed simply as outliers.

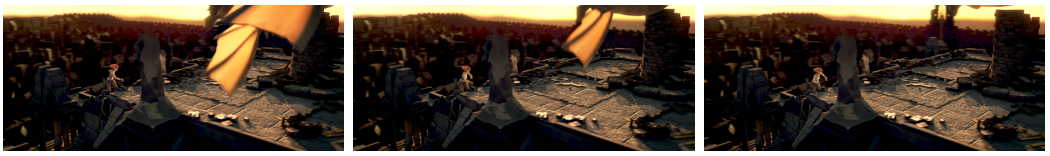
same location. This was used in (Xu et al., 2012; Kim et al., 2013b). Once occluded regions are detected, they are filled in using bilateral filtering, as was also done in (Sand and Teller, 2008).

A simpler approach simply marks pixels as occluded that have a high data cost, indicating that they are poor matches. In (Xiao et al., 2006), these regions were then filled in using bilateral filtering. In (Strecha et al., 2004), a visibility map was added to a probabilistic framework to estimate occluded regions in a more principled manner. A joint model for estimating both flow and occlusion is also proposed in (Ayvaci et al., 2012). Because occluded pixels do not participate in the data cost function, an additional term penalizes the number of occluded pixels.

Occlusions can also be estimated separately from optical flow. In (Stein and Hebert, 2009), motion and appearance features were used to identify occlusion boundaries, separate from motion estimation itself. In (Humayun et al., 2011), this was posed as a classification problem and a random forest classifier was learned

to detect occluded pixels. Improved results were obtained by (Sundberg et al., 2011), who also classify occlusion boundaries. They take the estimated motion boundaries and the optical flow map, and look at the difference of flow on both sides of the boundary to determine occlusions and figure-ground relationships.

Another approach that naturally deals with occlusion is to use *layered* models. Layered models were successfully used for modern optical flow methods in (Sun et al., 2010b). If an image can be successfully divided into separate layers, each representing primarily one motion estimate, then the flow for each layer can be estimated separately and occlusion reasoning naturally comes from reasoning about which layers occlude which others. In (Sun et al., 2010b), the layers were initialized by computing standard optical flow without considering occlusions and then clustering the motion vectors using K-means. In (Sun et al., 2012), this was extended to operate over multiple frames to improve the layer estimation. Their method also use a discrete, move-making algorithm rather than continuous optimization. The MRF used for estimating layer membership was made fully-connected in (Sun et al., 2013), and mean-field optimization within an EM framework allowed for efficient optimization, although only two layers were used. In (Sun et al., 2014), layers were used to improve the estimation of optical flow generated without considering occlusions. In particular, after estimating a flow field, the image is segmented and each segment is used as its own layer, which allows for occlusion estimation to refine the solution. In Chapter 3, we present a method for estimating occluded regions by discretizing the image into distinct triangular regions. Our approach can be considered a layered model where each triangle in our model is a separate layer.



(A) Image at $t-1$ (B) Image at t (C) Image at $t+1$

FIGURE 2.3: A situation where multi-frame optical flow is essential. The dragon's wing in the frame at time t goes out of frame at time $t+1$, making it nearly impossible to recover its motion of using two-frame optical flow. The wing, however, is visible in frame $t-1$, which provides information about its motion.

2.6 Multi-frame optical flow

Optical flow is often posed as a two-frame problem: estimate the apparent motion from the frame at time t to time $t+1$. While this works for simple situations, there is sometimes insufficient information in only two frames for a good correspondence to be found. An example of this situation is shown in Figure 2.3. In this figure, the dragon's wing at time t is no longer visible at time $t+1$, and thus no amount of smoothness constraint could propagate flow to this occluded region since the entire region is occluded. However, in the previous frame at time $t-1$ the dragon's wing is still visible and this provides information about the motion of the wing. The use of more than two frames in optical flow can thereby provide useful information and improve results.

Despite the relative lack of modern multi-frame optical flow algorithms, attempts have been made to model temporal consistency back to the work of (Murray and Buxton, 1987). The most common idea is to impose a sort of temporal regularity similar to the spatial regularity term. In particular, it is assumed that the flow estimates are smooth over time. In (Nagel, 1990), Nagel extended the idea of orienting spatial smoothness constraints along image edges into the temporal domain.

A related flow-driven temporal smoothness constraint was proposed subsequently in (Weickert and Schnörr, 2001).

However, in contrast to the spatial term where pixels can be connected with a static set of neighbors in a grid, a temporal continuity constraint works better if it is paired with *the pixel that it matches to* in the subsequent image. A primary reason why simply enforcing smoothness across the same pixel location over time does not work well is that many image sequences have a low frame-rate relative to their motions and thus temporal derivatives are meaningless. Of course, this pixel correspondence is not known beforehand, leading to a difficult optimization problem. In (Black and Anandan, 1990), the motion of a pixel was compared with an average of the same track point's previous velocities. In (Salgado and Sánchez, 2007), a model is used that penalizes deviations between velocity vectors in subsequent frames for matched pixels. A novel parameterization of temporal smoothness was presented in (Volz et al., 2011), where a constraint was imposed by parameterizing all flow values with respect to a center frame. In this model, five frames are used for motion estimation. Rather than having a separate flow estimate between all successive pairs of frames and needing to track points over time, the flow values are estimated as an *offset* such that flow from the center frame to any other frame can be obtained by summing the flow estimates at the same pixel location. This allows for temporal constraints to be naturally placed along the temporal dimension.

Temporal consistency has also been enforced in layered models. In (Sun et al., 2010b, 2012), a term of the cost function encourages pixels connected based on the current flow estimate to have similar layer assignments.

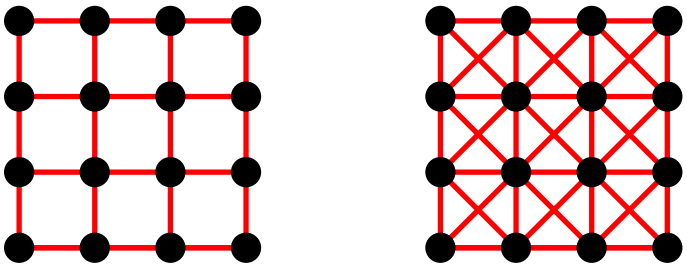
Another related method to our own multi-frame approach is that of multi-view

stereo. In multi-view stereo, more than two images are used for reconstruction and the result is a 3D surface of the imaged scene. In this case, a single point on the object is seen from multiple images. In (Newcombe et al., 2011), the data costs from multiple images are averaged, while in our method we take the minimum of several data costs to allow for occlusions. A similar approach was also taken in (Szeliski and Coughlan, 1997) and (Okutomi and Kanade, 1993). In (Szeliski and Coughlan, 1997; Sun et al., 2000) in particular, linear flow was assumed for multi-frame flow estimation. In (Kang et al., 2001), multiple frames were handled within a stereo correspondence problem by selecting a subset of frames for each pixel that were likely to be unoccluded.

In Chapter 4, we present a novel method of incorporating temporal information into any optical flow algorithm with minimal complexity, leading to very high-quality flow estimates on difficult image sequences.

2.7 Fusion methods

Fusion methods are an interesting optical flow technique, where multiple estimates of optical flow are combined in order to improve results. Most often, this is done by using multiple algorithms or parameter values to estimate the same two-frame motion. In (Lempitsky et al., 2008), flow estimates were generated using the Horn-Schunck and Lucas-Kanade algorithms with various parameter settings. The objective of the fusion step in (Lempitsky et al., 2008) was to directly minimize the energy of a global MRF model. This was done by considering two flow estimates at a time and performing optimization using a discrete move-making algorithm (Boykov et al., 2001). In (Jung et al., 2008), multiple local minima of an MRF model are combined by randomly partitioning and combining the estimates. A



(A) 4-connected neighborhood

(B) 8-connected neighborhood

FIGURE 2.4: 4- and 8-connected MRF neighborhoods. A 4-grid is more commonly used due to its simplicity.

related method in (Chen et al., 2013) produced a set of proposal motion models and then assigned each pixel to the models using an energy minimization framework.

Fusion can also be cast within a classification framework. In (Mac Aodha et al., 2013), a random forest classifier was trained to estimate how accurate the optical flow estimate is at each pixel. This confidence could potentially be used to combine several algorithms based on which is more confident. A similar approach was taken in (Mac Aodha et al., 2010), where a classifier was used to determine which algorithm from a set of them would likely give the most accurate solution at each pixel. In Chapter 4, we show how a related fusion method can be used to not only fuse multiple algorithms, but also to fuse temporal information that we call the *inertial estimates* of optical flow.

2.8 Discrete optimization

In the methods discussed so far, optical flow has been treated as a continuous optimization problem, where the offset assigned to each pixel location is potentially any real number. Locating the global optimum for such a function is extremely

difficult, especially since the data terms in optical flow problems are dependent on the image content and are likely very non-convex. Local optimization methods – especially when combined with sparse global feature matching – have been shown to achieve quite good results on a range of applications, but there is still no guarantee that the resulting solutions are not sub-optimal.

Another approach can be taken by assuming that the offset assigned to each pixel is taken from a *discrete* set of possible offsets. For example, we may allow each pixel in the first image to match to each other pixel in the second image. This approach has been used frequently in stereo correspondence problems (Boykov et al., 2001), and is often followed by an additional optimization step to achieve sub-pixel accuracy.

At first glance, this might seem like the wrong approach: why limit ourselves to discrete offsets when optical flow is naturally a continuous-valued problem? The advantage of the discrete approach is that many discrete optimization algorithms have a sense of *optimality*. For some problems, the optimal solution can be found in polynomial time, and even efficiently in practice. For other problems, it may be possible to find a bound on how far from the optimum the solution is, or even to label a subset of the pixels with their optimal offsets. This near-optimal aspect of discrete optimization may be helpful in difficult optical flow problems that involve many local minima. It also eases the requirement that all functions be continuous and differentiable so that gradients can be calculated.

The field of optimization of discrete graphical models is quite extensive and involves many situations that are not applicable here (Koller et al., 2007; Koller and Friedman, 2009). Here we only aim to provide a brief overview of the optimization of discrete Markov random fields (MRFs) as may be of interest in optical flow.

In many pixel-labeling problems – including optical flow – a graphical model can be constructed that involve a variable at each pixel in the image. The *label space* $\mathcal{L}$ is the finite set of possible labels that each pixel may be associated with. A cost function is associated with the graph that defines a *unary* data cost term that measures the quality of the label assignment at each pixel, and a *pairwise* cost that generally enforces some for of smoothness on the resulting solutions. The pairwise connections are typically placed between neighboring pixels in a 4-grid, but may be extended to more neighbors such as an 8-grid, as shown in Figure 2.4. This setup should be familiar, as it is the same setup that most continuous optical flow methods also employ. In particular, the unary cost term enforces the brightness constancy assumption and the pairwise term enforces the smoothness assumption. MRFs are generally formulated from a probabilistic perspective where a suitable distribution is defined over the network and the *maximum a posteriori* (MAP) solution is desired, but this is easily changed into an energy minimization problem by attempting to minimize the negative log-likelihood of the distribution. For more details, see (Koller and Friedman, 2009).

Optimization of discrete MRFs is quite difficult in general, and is an NP-hard problem for a large number of useful problems (Boykov et al., 2001). Because discrete MRFs are useful in many areas of vision, however, there has been a significant amount of research into optimizing them efficiently. This research has generally taken two different directions. The first line of research has tried to identify problems for which an optimal solution can be found in polynomial time. Alternatively, many approaches have focused on finding high-quality approximate solutions, with possible partial optimality or bounds on the solution quality.

2.8.1 Exact minimization of discrete MRFs

There are a number of situations for which the minimum-energy solution can be found efficiently. One such set of problems are when the graphs underlying the problem have *low treewidth*. Intuitively, one way to find the optimal solution to a discrete model is to start with one node and determine the cost of all labels for that node, then choose a neighboring node and determine the cost of labels for that node conditioned on the choices of labels for the first node, and so on, until the entire graph has been evaluated and the optimal solution can be found. This process is known as the *junction tree algorithm*. The complexity of this process depends on the order that the nodes are chosen in, and the resulting conditional probability tables can become quite large. For graphs with low treewidth, however, the size of the tables will not be too large and the optimum can be found efficiently (Pearl, 1988; Koller et al., 2007). Unfortunately, grid-graphs have tree widths dependent on the size of the grid and can be quite large. We note, however, that tree-structured graphs do have a bounded tree width and can be optimized efficiently, which we take advantage of in an algorithm presented in Chapter 5.

In (Greig et al., 1989), it was shown that binary labeling problems with submodular pairwise terms could be solved exactly and efficiently using a graph cuts algorithm. This was extended in (Ishikawa, 2003) to multi-label problems that have convex pairwise terms, under the condition that the labels have a linear ordering. For problems with submodular pairwise cost functions, it was shown in (Schlesinger and Flach, 2006) that the problems can be transformed into a binary submodular MRF, for which an exact solution can be found. Specific labeling problems, such as those involving a tiered structure (Felzenszwalb and Veksler, 2010) or where

the nodes and labels form a tree-structure metric (Felzenszwalb et al., 2010b), can also be solved exactly.

2.8.2 Approximate minimization of discrete MRFs

For many cases of interest, the optimal solution of a discrete MRF is NP-hard to compute. For these problems, a number of approximate optimization methods have been proposed that give high-quality solutions. Approximate methods generally fall into two groups: graph-cut based approaches and message-passing methods. Graph-cut based approaches are based on the idea that for certain problems, the optimal solution of a binary MRF can be computed exactly using graph cuts. In (Boykov et al., 2001), the *alpha expansion* and *alpha-beta swap* algorithms were proposed, both of which transform a multi-label problem into a series of binary problems. In alpha expansion, a single label α is chosen at time t and each pixel is allowed to choose between its current label and the label α. In alpha-beta swap, two labels are chosen and any pixel with one of the two labels is allowed to swap. While no optimality guarantee is known for these methods, they will at least converge to a local minimum. A related approach, known as Quadratic Pseudo-Boolean Optimization (QPBO), results in partially-optimal solutions.

In message-passing algorithms, "messages" are passed between nodes in the graph in order to update their solutions until convergence. While this process finds the exact solution in tree-structured graphs (Pearl, 1988), there are no convergence guarantees for graphs that contain loops. Even so, this "loopy" belief propagation algorithm can result in relatively high-quality solutions (Szeliski et al., 2006). A recent approach, known as *tree-reweighted message passing* (TRW-S) (Wainwright et al., 2002; Kolmogorov, 2006), performs message passing using a weighting based

on decomposing the graphical model into a set of spanning trees. Interestingly, this algorithm provides a lower bound on the optimal energy, and thus it is possible to have an idea of how good the resulting solution is. A related message-passing algorithm known as *dual decomposition* was proposed in (Komodakis et al., 2007). In this approach, the problem is divided into subproblems and the dual of these subproblems is optimized in an attempt to get them to agree. Like TRW-S, a lower bound on the energy of the model can give an indication of the quality of the solution. Another related approach is that of (Sapp et al., 2011), where an intractable graphical model is decomposed into multiple trees and a weaker form of agreement was enforced as compared to dual decomposition.

2.8.3 Application to optical flow

Discrete MRF optimization has been used extensively for stereo estimation problems and can achieve very accurate results (Szeliski et al., 2006). However, the application of such methods to optical flow has been very limited. A primary reason for this is that the label space is too large. In standard stereo problems, a pixel may be restricted to move only 10 or 15 pixels from its initial location. Even for optical flow problems in a similar range, the number of possible pixel matches is immediately squared because both a horizontal and a vertical offset need to be computed. Additionally, the displacement magnitude in optical flow is not limited by the baseline and resolution of the cameras as it is in stereo. Indeed, for fast-moving objects, a pixel in one image may travel completely across the entire image in the next frame. Thus, for difficult optical flow problems, the number of labels may be in the hundreds of thousands. Because most discrete

optimization methods are dependent on the size of the label space, they cannot usually be directly applied to optical flow problems.

Still, some attempts have been made to extend discrete optimization methods into the domain of optical flow. In (Glocker et al., 2008), discrete optimization techniques were used for optical flow, although the setup was somewhat different in that they used a grid of control points with interpolation using splines. In (Goldluecke and Cremers, 2010), a convex relaxation for multi-label problems was proposed that reduced the computational dependence on the size of the label space, allowing for optimization of MRFs for optical flow problems. However, the label spaces were still restricted to around 50×50. This framework was extended in (Strekalovskiy et al., 2011) to give a tighter relaxation. A similar approach in (Goldstein et al., 2012; Pock et al., 2008; Cremers et al., 2011) extended previous results that required a linear ordering of variables (Ishikawa, 2003) to vector-valued labels, allowing for its application to optical flow. Another approach taken by (Wu et al., 2010) involves the use of discrete optimization within a hierarchy, but their model contains loops and the results are therefore still approximate.

In all these cases, the application to optical flow requires a high computational complexity that would not allow it to be applied to problems with very large displacements. The method we present in Chapter 5 takes a different approach: we take advantage of recent advances in segmentation algorithms in computer vision to change the structure of the MRF such that it is a tree and thus is solvable in polynomial time. Related approaches have been applied to scene labeling problems (Feng et al., 2002; Reynolds and Murphy, 2007; Zhu et al., 2012), but these problems generally involve many fewer labels. Our method is able to find an optimal solution to a global optical flow problem involving hundreds of thousands of labels.

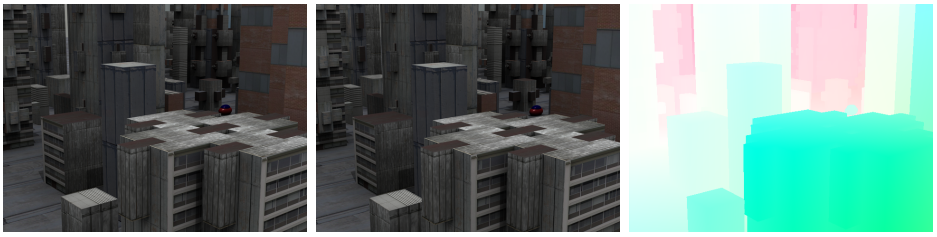
2.9 Datasets and evaluation

The methods that we present in this thesis are primarily evaluated on three datasets: Middlebury (Baker et al., 2011), MPI-Sintel (Butler et al., 2012), and KITTI (Geiger et al., 2012). Examples of optical flow estimates on the training set for each of these datasets are shown in Figure 2.5.

The Middlebury dataset (Figure 2.5a) has become a standard benchmark in optical flow. The dataset is relatively small with only 8 images containing groundtruth data in both the training and testing sets. The motions of Middlebury are also very small – on the order of a few pixels – and all motion is due to camera motion rather than from the objects themselves. Because of its simplicity, modern methods have achieved nearly error-free results and over 100 methods have been submitted for evaluation.

The MPI-Sintel dataset (Figure 2.5b) is more recent and has significantly more complexity. The MPI-Sintel dataset was generated from an open-source, computer-generated short film. The images include difficulties such as lighting variation, large motions, significant occlusions, motion blur, and fog. The dataset is also quite large, with over 1000 training and testing images.

The KITTI dataset (Figure 2.5c) is based on the goal of developing autonomous driving systems. The images were captured from a camera mounted to a driving car. Thus, the motion between images often has very sigfnificant planar effects and the motion is primarily generated from the motion of the car. The images are also only grayscale and have very low constrast.



(A) Middlebury dataset (Baker et al., 2011)

(B) MPI-Sintel dataset (Butler et al., 2012)

(C) KITTI dataset (Geiger et al., 2012)

FIGURE 2.5: Examples from optical flow datasets. From left to right, we show the first image, second image, and groundtruth optical flow.

The evaluation of optical flow can take many forms. The most common error measure is the *endpoint error* (EPE), which is defined as the Euclidean distance between the estimated values u, v and actual flow values u^*, v^*:

$$EPE(u, v; u^*, v^*) = \sqrt{(u - u^*)^2 + (v - v^*)^2} \, . \tag{2.9}$$

This error measure is used in the evaluation for Middlebury and is also the primary error measure for MPI-Sintel. This will also be the primary error measurement used in this thesis.

Another error measure is the *angular error*, which is defined as the angle between the vectors $(u, v, 1)$ and $(u^*, v^*, 1)$:

$$AE(u, v; u^*, v^*) = \cos^{-1}\left[\frac{1 + u \times u^* + v \times v^*}{\sqrt{1 + u^2 + v^2}\sqrt{1 + (u^*)^2 + (v^*)^2}}\right] \, . \tag{2.10}$$

The Middlebury dataset also uses this angular measure in its evaluation, although they note that endpoint error is probably better for most applications (Baker et al., 2011).

The KITTI dataset calculates the error by specifying a threshold and reporting the number of pixels which have an endpoint error in excess of the threshold.

3

Triangulation-Based Optical Flow

In this chapter, we present a framework for optical flow based on a triangulation of the image domain, over which we compute optical flow. We employ a tectonic model where the triangular facets are allowed to move relative to each other and a numerical quadrature scheme is used to handle the resulting occlusion effects. This allows for occlusions to be directly incorporated into the optimization procedure without the need for arbitrary regularization terms. The triangulation also allows us to easily impose a non-local smoothness term within our model.

The proposed triangular decomposition also has computational benefits. A common approach in many of the top algorithms that perform continuous optimization (Brox et al., 2004; Brox and Malik, 2011; Weinzaepfel et al., 2013; Xu et al., 2012) is to use a variational method by calculating the Euler-Lagrange equations for the cost function, linearizing them, solving the linear system using an iterative solver, and repeating this process within a coarse-to-fine image pyramid. Other possibilities, however, exist. For example, gradient descent could be used to directly minimize the cost function (Black and Anandan, 1996), but convergence may be slow and it requires a choice of step size. Newton's method could be used instead, which would result in much faster convergence, but to the best of our knowledge

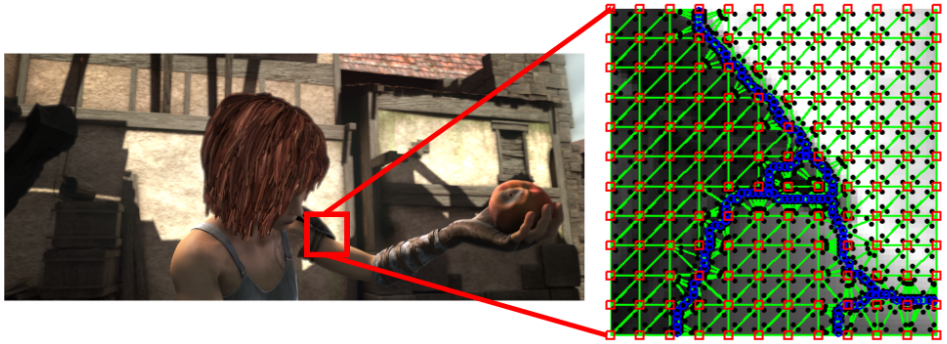


FIGURE 3.1: A triangulated section of an image. Blue circles denote edge points and red squares denote points generated on a uniform grid with a spacing of 5 pixels. The Delaunay triangulation given by the green lines tessellates the image into regions which form the basis of our algorithm. In practice, the data cost functions are evaluated at a set of quadrature points within each triangle, shown here as black dots.

this has not been attempted for global optical flow methods for computational reasons. In fact, some researchers have suggested that Newton-type methods could not be applied to optical flow problems because of the computational difficulties associated with solving for the Newton step (Baker et al., 2011). In this chapter we show how Newton's method can be implemented efficiently using a triangulation, even for large images, using an exact Cholesky factorization.

In Section 3.1, we describe how our triangulation is set up. We subsequently define the model used for optical flow in Section 3.2 and show how a non-local smoothness term is easily included. In Section 3.4, we show how our triangulation allows for the use of an efficient Cholesky factorization within Newton's method. Section 3.3 describes how the triangulation is used for direct occlusion estimation within our model.

3.1 Problem setup

Let $I_1, I_2 : (\Omega \subseteq \mathbb{R}^2) \to \mathbb{R}^d$ be two d-dimensional images. We consider both images to be color images in the CIELab color space such that $d = 3$. Channels are denoted using a superscript, such as $I_1^{(c)}$. We attempt to estimate the motion of each point from I_1 to I_2. The estimated motions in the horizontal and vertical directions are denoted by the functions $u, v : \Omega \to \mathbb{R}$ which map a point in the image domain Ω to its estimated motion. For simplicity, let $\mathbf{x} = (x, y)$ be a point in Ω, and let $f : \Omega \to \mathbb{R}^2$ be a function such that $f(\mathbf{x}) = (u(\mathbf{x}), v(\mathbf{x}))$. In addition, we estimate a function $m : \Omega \to \mathbb{R}$ which is a multiplicative factor that measures changes in lightness between frames, as we will define in Section 3.2. This "generalized brightness constancy" model has been previously used (Negahdaripour, 1998), and we found that it improved our results.

A key aspect of our approach is that we consider the image to be a continuous 2D function of the image domain, rather than a set of sampled pixel locations. We do so by extending the sampled pixel values to intermediate locations in the image plane using bicubic interpolation. More specifically, at any continuous-valued location $(x, y) \in \Omega$, the value of the image at channel c is computed using a quadratic form

$$I_1^{(c)}(x, y) = \begin{bmatrix} x^3 & x^2 & x & 1 \end{bmatrix} K_c \begin{bmatrix} y^3 & y^2 & y & 1 \end{bmatrix}^T , \tag{3.1}$$

where K_c is the matrix of coefficients based on the values of nearby pixels. Note that spatial image derivatives at any point are easily computed using derivatives of this quadratic form. Given this representation of the image as a continuous

function, our goal is to compute a corresponding continuous function $f(\cdot)$ that specifies the motion of each point in Ω.

We discretize the problem by tessellating the image I_1 into discrete triangular regions (Figure 3.1), and then seek to estimate a constant motion vector for each triangle. We note that it is also possible to use more complex models for the motion of each triangle. For example, each triangle could be assigned an affine flow rather than a constant value, although the resulting model has more parameters and we found that using a denser triangulation with constant flow per triangle resulted in better solutions than using fewer triangles with an affine flow. Another possibility is to allow for an affine flow parameterized by the three corners of each triangle where corners that are shared between triangles are restricted to have the same value. While this model then has fewer parameters, it would also require estimating occlusion boundaries explicitly as latent variables, making the model more complex. For these reasons, we use the triangles as a discretization of the image and estimate a constant flow for each triangle.

Because we assume the motion to be constant within each triangle, the triangles should be made to conform to the content of the image in order to find an accurate solution. This approach is similar to that of (Glocker et al., 2010), where a triangulation of the image domain was also used. We use the following procedure. First, we extract edges from the image I_1 by using the method of (Donoser and Schmalstieg, 2014) and thresholding the given ultrametric contour map at 0.2. Each edge pixel in the image is then used as a vertex in our triangulation. In addition to these points, we also use a set of grid points that are evenly spaced throughout the 2D image, which serve to limit the maximum dimension of the resulting triangles. The grid points and edge pixels are combined and a Delaunay

triangulation is constructed. An example of a tessellated image is shown in Figure 3.1.

The cost function that we optimize will be defined as an integral over the entire continuous image domain. In order to numerically evaluate this integral, we use a method based on *quadrature points*, wherein the integral is approximated as a weighted sum of function values at specific points within each triangle. For this, we use the scheme described in (Cowper, 1973) which provides the locations of quadrature points and their associated weights for integrating functions over triangular domains. The integral is thereby approximated by forming a weighted sum of the cost function evaluated at these points. We used 3 quadrature points per triangle, as shown in Figure 3.1.

3.2 Cost function

Our cost function consists of data terms and smoothness terms. The data terms penalize incorrectly-matched pixels based on image data, while the smoothness terms encourage solutions that are smooth over the image domain. Our cost function takes the form

$$\mathcal{E}(f,m) = \mathcal{D}(f,m) + \tau_0\mathcal{F}(f) + \tau_1\mathcal{S}_1(f) + \tau_2\mathcal{S}_2(f) + \tau_3\mathcal{S}_3(m)\ , \tag{3.2}$$

where $\mathcal{D}(\cdot)$ is a data cost term based on image data, $\mathcal{F}(\cdot)$ is a feature matching term, and $\mathcal{S}_1(\cdot), \mathcal{S}_2(\cdot)$ and $\mathcal{S}_3(\cdot)$ are smoothness terms. The parameters τ_0, τ_1, τ_2 and τ_3 control the tradeoff between these terms.

3.2.1 Data term

Our data term is given by the equation

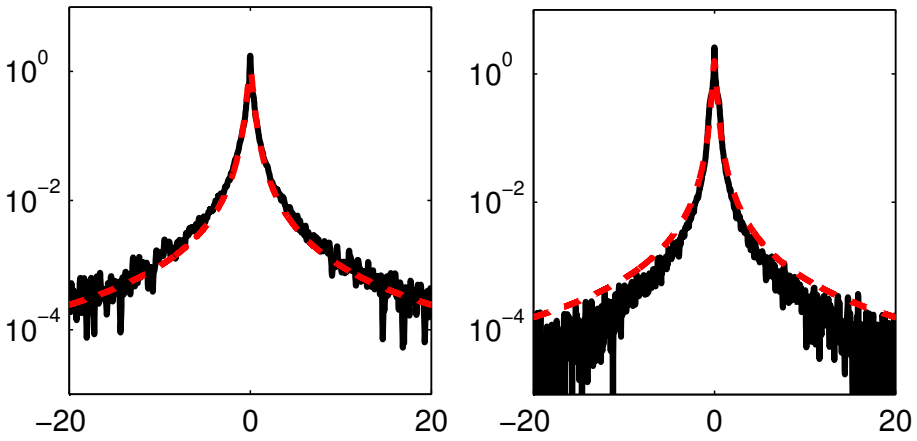
$$\mathcal{D}(f,m) = \int_{\Omega} \Phi_{\gamma} \left(I_2(\mathbf{x}+f(\mathbf{x})) - \begin{bmatrix} m(\mathbf{x}) & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} I_1(\mathbf{x}) \right) d\mathbf{x} \,, \tag{3.3}$$

where $\Phi_{\gamma}(\cdot)$ is a robust error function with parameter vector γ. Because of the large amount of data made available in the MPI-Sintel dataset, we chose our robust cost function through a fitting procedure. In particular, the difference values, $I_2^{(c)}(\mathbf{x}+f(\mathbf{x})) - I_1^{(c)}(\mathbf{x})$, are well-modeled by a Cauchy distribution, as has been previously observed (Sun et al., 2008). In Figure 3.2 we show histograms of these values and the corresponding Cauchy distributions for the lightness channel L and the combined color channels a and b.

The robust function $\Phi_{\gamma}(\cdot)$ is then the negative log-likelihood of the Cauchy density function, summed over all channels:

$$\Phi_{\gamma}(\delta) = \sum_{c=1}^{d} \log \left[\pi(\delta_c^2 + \gamma_c^2)/\gamma_c \right] \,. \tag{3.4}$$

A separate distribution was fit to the lightness and to the combined color channels, giving values of $\gamma_1 = 0.3044$ for lightness and $\gamma_2 = \gamma_3 = 0.2012$ for the color channels.



(A) L channel; $\gamma = 0.3044$ (B) a, b channels; $\gamma = 0.2012$

FIGURE 3.2: Matching errors are well-fit by a Cauchy distribution. Here we show the log-histograms of the difference values between the two images using the groundtruth flow estimates in black and the fitted Cauchy distributions in red. We fit a fit a separate Cauchy distribution **(a)** to the lightness channel and **(b)** to the combined color channels of the CIELab images of the MPI-Sintel dataset.

3.2.2 Feature matching term

Feature matching has been shown to be effective at improving optical flow results, especially for large motions (Brox and Malik, 2011; Xu et al., 2012). We use HOG features (Dalal and Triggs, 2005), computed *densely* at every pixel. These descriptors are then matched to their nearest neighbor in the opposite image using the approximate nearest neighbors library FLANN (Muja and Lowe, 2009). The matches from I_1 to I_2 generate motion estimates for each of the pixels, which we denote as $f_{HOG} : \Omega \to \mathbb{R}^2$.

If the HOG match is correct, then it's desirable to have $f(\mathbf{x})$ be close to $f_{HOG}(\mathbf{x})$. Thus, our feature matching term is given by

$$\mathcal{F}(f) = \int_\Omega s(\mathbf{x}) \Psi_\alpha \left(\| f(\mathbf{x}) - f_{HOG}(\mathbf{x}) \|_2 \right) \, d\mathbf{x} \, , \tag{3.5}$$

where

$$\Psi_\alpha(\delta) = (\delta^2 + \epsilon)^\alpha \tag{3.6}$$

is a robust cost function with parameter α and small constant epsilon (i.e. $\epsilon = 0.001$) (Sun et al., 2010a). For $\alpha = 1$, this is a pseudo-ℓ_2 penalty. As α decreases, it becomes less convex with it becoming a pseudo-ℓ_1 penalty for $\alpha = 0.5$. For our feature matching term, we set $\alpha = 0.5$.

The function $s : \Omega \to \mathbb{R}$ is a weighting function which measures the confidence in each HOG match, and is defined as follows. First, we enforce forward-backward consistency by setting $s(\mathbf{x}) = 0$ if a match is not a mutual nearest-neighbor. Otherwise, we let $s(\mathbf{x}) = ((d_2 - d_1)/d_1)^{0.2}$, where d_i is the ℓ_1 distance between the HOG feature vector in I_1 at location $\mathbf{x}$ and its i^{th}-closest match in I_2. This is similar to the weight used in (Brox and Malik, 2011) and provides a measure of confidence for each HOG match.

When evaluating this term on a triangulation, each triangle is assigned a HOG flow estimate by taking the mean of all flow values within the triangle t weighted by their confidence scores, $\sum_{\mathbf{x}\in t} \frac{s(\mathbf{x})}{\sum_{\mathbf{x}\in t} s(\mathbf{x})} f_{HOG}(\mathbf{x})$. This flow value is then used for all quadrature points within the triangle when evaluating the cost function.

While we used HOG features due to their speed and simplicity, more complex feature matching could be used here as well, such as (Weinzaepfel et al., 2013) or (Byrne and Shi, 2013).

3.2.3 Smoothness terms

We use two different smoothness terms in our cost function: a first-order term that penalizes non-constant flow fields, and a second-order term that penalizes

non-affine flow fields.

3.2.3.1 First-order smoothness

A first-order smoothness term penalizes non-constant motion estimates. In our cost function, all pairs of neighboring triangles are considered. The cost is defined as

$$\mathcal{S}_1(f) = \sum_{t_i,t_j \in N} |t_i||t_j| \Psi_\alpha \left(\frac{\|f(\bar{t_i}) - f(\bar{t_j})\|_2}{\|\bar{t_i} - \bar{t_j}\|_2} \right) , \tag{3.7}$$

where $N \subseteq T \times T$ is the set of all neighboring triangles T in the tessellation, $\bar{t_i}$ is the centroid of triangle $t_i \in T$, and $|t_i|$ is its area. The function $\Psi_\alpha(\cdot)$ is a robust cost function, which was defined in Equation (3.6).

This cost function penalizes differences in the flows between neighboring triangles, modulated by the distance between their centroids. Note that we also multiply by the area of the two triangles (rather than by the edge length), which effectively connects all points within one triangle to all points in the other triangle. Now, recall that our triangulation is constructed using both edges points and a set of uniform grid points (Figure 3.1). The triangles along edges will therefore tend to have a smaller area, resulting in a weaker smoothness constraint. In this way, our triangulation naturally allows for a non-local smoothness cost (Werlberger et al., 2010).

We also apply this smoothness cost to the multiplicative term m to encourage only locally-consistent changes in image brightness. For this, we use $\alpha = 0.5$.

3.2.3.2 Second-order smoothness

While a first-order smoothness term penalizes non-constant flows, a second-order smoothness term penalizes non-planar flows. This allows for motion fields with a constant gradient, which is important for datasets where such motions are common, such as KITTI (Section 4.3.3).

Intuitively, our second order smoothness term says that the flow of each triangle is encouraged to be near the plane that is formed from the flow values of its three neighbors. Formally, the cost function is written as a sum of costs over all triangles $t \in T$:

$$\mathcal{S}_2(f) = \sum_{t \in T} |t_i||t_j||t_k| \Psi_\alpha \left(\frac{\| f(\bar{t}) - \left[\lambda_i f(\bar{t_i}) + \lambda_j f(\bar{t_j}) + \lambda_k f(\bar{t_k}) \right] \|_2}{|\Delta_{ijk}|} \right) . \tag{3.8}$$

Here, t_i, t_j and t_k are the three neighboring triangles to t. The values λ_i, λ_j and λ_k are the barycentric coordinates of the centroid of t with respect to the centroids of t_i, t_j and t_k. In other words, the numerator is zero exactly when the value of triangle t lies on the plane passing through the values of its neighboring triangles' centroids. This is then normalized by $|\Delta_{ijk}|$, the area of the triangle formed by connecting the centroids of t_i, t_j and t_k. Similar to the first-order smoothness term, the function $\Psi_\alpha(\cdot)$ is a robust cost function and each term is multiplied by the areas of the three neighboring triangles, which imparts a non-local character to the cost.

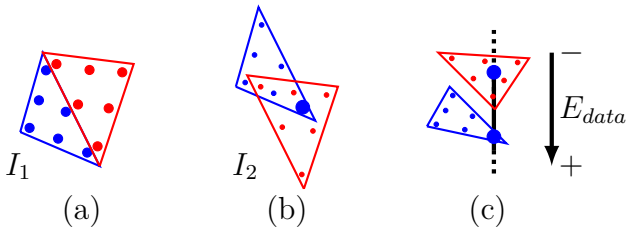


FIGURE 3.3: Depiction of our occlusion term. **(a)** Two triangles and their quadrature points in the tessellation of I_1. **(b)** The triangles are moved to their estimated locations in I_2, where they now overlap. Each quadrature point is processed separately and we have highlighted one quadrature point as an example. **(c)** The data cost is compared for all overlapping triangles at the quadrature point. The quadrature point here has a lower data cost at the same location in the red triangle, and so we mark the quadrature point as occluded.

3.3 Occlusion reasoning

Because we model an image as a set of triangular pieces that can move independently, we can directly reason about occlusions. A depiction of this process is shown in Figure 3.3. At each iteration of our algorithm, for each quadrature point in each triangle of I_1, we compute where it appears in the other image I_2. We then determine whether any other triangles overlap it in I_2. For each of these overlapping triangles, we determine whether that triangle offers a better explanation for that location as measured by the data cost (Equation (3.3)). If a better solution exists, then the quadrature point in question is labeled as occluded. The occluded quadrature points are not included in the evaluation of the data cost. In this way, the cost function only includes points which are estimated to be unoccluded. Note that these occlusion estimates are generated *directly* from the geometry and from the data cost term; no additional regularization parameters are needed to avoid the trivial solution of labeling all points occluded. An example of our occlusion estimation is show in Figure 3.4.

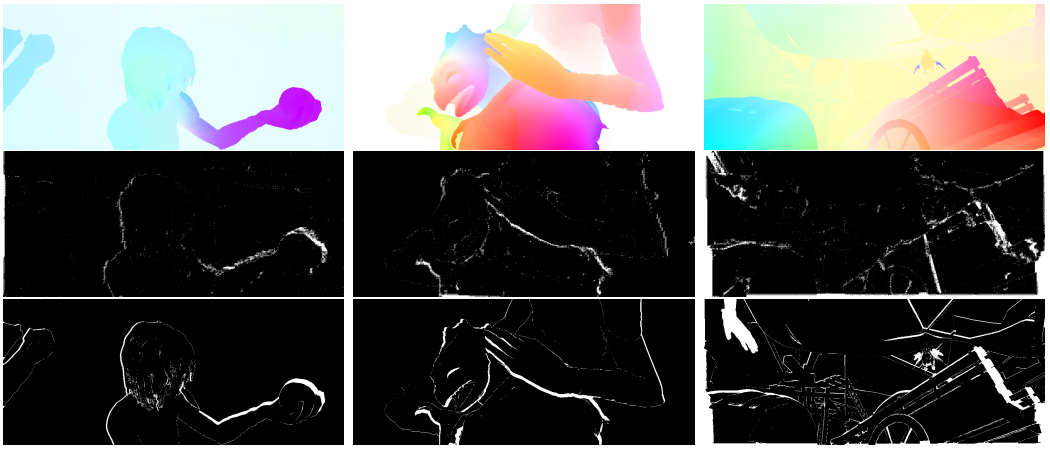


FIGURE 3.4: Examples of our occlusion estimation on MPI-Sintel. During optimization, the occlusion status of each quadrature point in each triangle is directly estimated. For visualization, we label each triangle a value in $[0, 1]$ as the proportion of its quadrature points that are labeled occluded, and then each pixel is labeled based on the triangles that it overlaps. **Top:** Groundtruth flow. **Middle:** Estimated occlusions. **Bottom:** Groundtruth occlusions.

Occlusions can be calculated efficiently by rasterizing all of the triangles to determine which pixels they overlap in I_2. When evaluating the occlusion term for a quadrature point, only triangles rasterized to the same pixel need to be considered as potential occluders.

3.4 Optimization

As is standard, local optimization is carried out within a coarse-to-fine image pyramid (Brox et al., 2004). We begin with a zero-valued flow and at each level the flow estimate from the previous level (appropriately scaled) is used as an initialization. At each level, a new triangulation is calculated as described in Section 3.1.

Rather than linearizing the Euler-Lagrange equations (Brox et al., 2004), we use Newton's method, which provides flexibility to our framework since any suitably-differentiable function can be substituted for our cost function without changing the optimization scheme. Newton's method is a second-order optimization algorithm that fits a quadratic approximation to a function, takes a step to the minimum of the approximation, and iterates until convergence. More specifically, let $g(f)$ and $H(f)$ be the gradient and Hessian of our cost function at the current estimate f. The Newton update is computed by solving the linear system $H\hat{f} = -g$ and updating $f \leftarrow f + \hat{f}$. Our cost function is differentiable, so the gradient and Hessian can readily be computed.

In order to successfully implement Newton's method, there are two issues that must be addressed. First, the Hessian matrix H needs to be symmetric positive-definite in order for the minimum of the quadratic approximation to be well-defined. Second, a method for solving the linear system must be chosen. We address these two issues in the following sections.

3.4.1 Ensuring that H is symmetric positive-definite

In practice, it is necessary to approximate the Hessian matrix, H, to ensure that it is positive definite and that the quadratic approximation has a well-defined minimum. We do so using an iteratively-reweighted least-squares (IRLS) approach, which exploits the structure of our objective function, following (Zhang, 1997). We note that other options for optimization also exist, such as the methods based on functional lifting which are shown in (Zach, 2014) to outperform IRLS in certain settings, but we do not explore them here.

Let $r = Ay - b$ be the residuals to be minimized, which are an affine function of the variables y. For example, for the unary term $\mathcal{F}(\cdot)$, we have A being the identity matrix, $A = I$, and b being the HOG flow values from HOG matching; for pairwise terms we may have that A is the signed incidence matrix that takes the difference $y_i - y_j$ for each pair of neighboring triangles i and j, and $b = 0$.

The cost function is then some (possibly non-convex) function of r:

$$C(y) = \sum_i \rho(r_i) = \sum_i \rho(A_{i:}y - b_i) \ . \tag{3.9}$$

Setting the derivative of this function to zero for y_j yields

$$\sum_i \frac{\partial \rho}{\partial r_i} \frac{\partial r_i}{\partial y_j} = 0 \tag{3.10}$$

Define the weight function

$$w(x) = \frac{1}{x} \frac{\partial \rho}{\partial x} \ , \tag{3.11}$$

and we can equivalently write

$$\sum_i w(r_i) r_i \frac{\partial r_i}{\partial y_j} = 0 \ . \tag{3.12}$$

This is the same set of equations that would occur if we had been solving

$$\min \sum_i w(r_i) r_i^2 \ , \tag{3.13}$$

and had considered $w(r_i)$ as a fixed variable at each iteration. Thus, we attempt to minimize the cost function in Equation (3.9) by computing the weights w and subsequently solving the weighted least-squares problem in Equation (3.13).

Because Equation (3.13) is a weighted quadratic function, it can be written as the quadratic form

$$\min r^T W r = y^T \left[A^T W A\right] y - 2y^T \left[A^T W b\right] + b^T W b \,, \tag{3.14}$$

where $W = \mathrm{diag}\,(w)$ is a diagonal matrix of weights. The gradient and Hessian matrices are straightforward to calculate:

$$\begin{aligned} g &= 2\left[A^T W A\right] y - 2\left[A^T W b\right] \\ H &= 2\left[A^T W A\right] \,. \end{aligned} \tag{3.15}$$

Importantly, *the Hessian in Equation* (3.15) *is positive definite if and only if the weights* w *are all positive.* Most cost functions $\rho(x)$ have the properties of being symmetric about the origin and monotonically increasing with $|x|$. For such functions, $\frac{\partial \rho}{\partial x}$ always has the same sign as x and therefore $w(x) = \frac{1}{x}\frac{\partial \rho}{\partial x}$ will never be negative. This allows us to safely use Newton's method even for non-convex cost functions.

3.4.2 Cholesky-based optimization

To find the Newton step at each iteration, a sparse linear system $H\hat{f} = -g$, must be solved for $\hat{f}$. This is most commonly done with an iterative method such as Successive Over-Relaxation (SOR). An alternative approach is to directly decompose the Hessian matrix into its Cholesky factorization, after which the linear system can be solved directly.

If H is symmetric and positive-definite, then it can be uniquely decomposed into its Cholesky factorization, $H = LL^T$, where L is a lower-triangular matrix. After

this factorization, the solution to the linear system can be obtained easily in the following way. First, the system can be rewritten as $LL^T\hat{f} = -g$ or $Ly = -g$, where $y = L^T\hat{f}$. Since L is lower-triangular, the solution for y can be found by simple forward-substitution, after which the solution to $L^T\hat{f} = y$ can be found using backward-substitution.

The computation of the Cholesky factorization itself can be done efficiently and has been implemented in readily-available optimized software packages such as CHOLMOD (Chen et al., 2008). For completeness, we briefly review one method that can be used to compute a Cholesky decomposition. First, we write the matrices as

$$H = \begin{bmatrix} \alpha & v^T \\ v & B \end{bmatrix}, \qquad L = \begin{bmatrix} \beta & 0 \\ \ell & M \end{bmatrix}, \tag{3.16}$$

where scalars are denoted using Greek letters, vectors are lowercase letters and matrices are uppercase. Then, $H = LL^T$ can be written as

$$\begin{bmatrix} \alpha & v^T \\ v & B \end{bmatrix} = \begin{bmatrix} \beta & 0 \\ \ell & M \end{bmatrix} \begin{bmatrix} \beta & \ell^T \\ 0 & M^T \end{bmatrix} = \begin{bmatrix} \beta^2 & \beta\ell^T \\ \beta\ell & \ell\ell^T + MM^T \end{bmatrix}. \tag{3.17}$$

By equating values, we find that

$$\beta = \sqrt{\alpha} \tag{3.18}$$

and

$$\ell = v/\beta = v/\sqrt{\alpha}\,. \tag{3.19}$$

We also find that

$$MM^T = B - \ell\ell^T = B - vv^T/\alpha\,. \tag{3.20}$$

Notice that if M is a lower-triangular matrix, then LL^T is indeed a Cholesky decomposition of H. Thus, this process has reduced the dimension by 1 and it can be applied recursively to find a Cholesky decomposition of the matrix $C = B - vv^T/\alpha$.

In our framework, the matrix H is very sparse since it encodes only local interactions in the image. However, directly computing a Cholesky factorization of even a sparse matrix can use a significant amount of memory. The reason for this is that even though H may be sparse, the L matrix may be much more dense. This can be seen in the algorithm we have described by considering that after the first iteration, the Cholesky algorithm is applied recursively to the matrix $C = B - vv^T/\alpha$. While the matrix B is a direct submatrix of H and thus has the same sparsity pattern, the matrix vv^T may add additional "fill-in" elements. In particular, vv^T will be nonzero at all locations i, j for which i and j are nonzero in the vector v. In words, this means that all neighbors of the node represented by vector v are connected into a clique. Thus, this indicates that (1) the *ordering* of when each node is processed in the Cholesky algorithm has an effect on the density of the resulting matrix L, and (2) graphs that have fewer neighbors per node will potentially lead to a lower fill-in rate.

In optical flow, the Cholesky factorization algorithm can be particularly problematic since the matrix H may be extremely large: for an image with n pixels, H will be of size $2n \times 2n$. If H is sparse, then storing H in memory is generally feasible, but the Cholesky factorization can be difficult since the factor L may be much more dense than H.

It is often the case in optical flow problems that H represents a planar or near-planar graph, such as when the cost function has binary potentials between a pixel

and each of its 4 or 8 pixel neighbors. For such graphs, it is known that an ordering exists such that there is only a $O(\log n)$ factor fill-in (George, 1973; Lipton et al., 1979), indicating that the L will not be *too* dense. However, the constant in front depends upon the number of connections that each node has. This is where using a triangulation is advantageous: a triangulation of the image plane results in a graph with only 3 neighbors – rather than 4 or 8 – and so it will have a Cholesky factorization with less fill-in. When a second-order smoothness term is used, neighbors-of-neighbors are included in the graphical model and the number of connections are correspondingly larger but is still smaller for triangulations than for grid graphs. In practice, we found that a triangulation resulted in about 25% less memory used than for a 4-grid with the same number of variables. Furthermore, a triangulated image generally has fewer triangles than pixels, resulting in a smaller linear system. Taken together, this allows us to use a Cholesky-based linearly solver without encountering memory issues.

All of these results, however, consider fill-in when the *optimal* variable ordering is used. Unfortunately, determining the optimal ordering is NP-hard for general graphs. However, good approximation algorithms exist. We focus on three such algorithms. The first is the Approximate Minimum Degree (AMD) algorithm (Amestoy et al., 2004), which tries to minimize the number of off-diagonal values in the pivot row or column at each iteration within the Cholesky algorithm. We also consider Nested Dissection (NESDIS) (George, 1973; Lipton et al., 1979), which finds a variable ordering by recursively partitioning the set of variables. Finally, we also consider METIS (Karypis and Kumar, 1998), another reordering algorithm.

We also note that the issues explored in this section are closely related to the

problem of inference in graphical models (Koller and Friedman, 2009). In particular, exact inference can be computed in a graphical model if the associated graph structure is *chordal*, meaning that every cycle of more than three nodes has a link between two non-adjacent nodes in the cycle. If a graph is not chordal, then it can be made chordal by adding "fill-in" edges. These fill-in edges are equivalent to the fill-in edges produced during Cholesky factorization. Indeed, if a positive-definite matrix is chordal, then there exists a Cholesky factorization for which there is no fill-in. Thus, rather than computing a Cholesky factorization that generates fill-in, we could equivalently make the graph chordal and then compute a zero-fill Cholesky factorization. The number of fill-in edges required for both approaches is correspondingly the same: in (Lipton et al., 1979) it was shown that planar graphs have an ordering with a $O(\log n)$ factor fill-in, while (Chung and Mumford, 1994) showed that planar graphs can be made chordal using $O(\log n)$ edges, after which a Cholesky factorization causes no additional fill in.

3.5 Experiments

We postpone quantitative results of our optical flow method until Section 4.3, so that results can be presented with the multiframe methods presented in Chapter 4. Here, we evaluate the effect of Cholesky-based optimization.

Time (s)						
		Number of variables				
Reordering	Graph	20^2	50^2	100^2	200^2	500^2
AMD	8-grid	0.01	0.06	0.30	1.80	21.90
	4-grid	0.00	0.03	0.17	0.99	11.32
	triang.	0.00	0.03	0.14	0.79	7.17
METIS	8-grid	0.01	0.09	0.46	2.26	19.86
	4-grid	0.01	0.06	0.33	1.62	14.05
	triang.	0.01	0.05	0.26	1.23	9.51
NESDIS	8-grid	0.01	0.09	0.50	2.57	22.72
	4-grid	0.01	0.07	0.37	1.91	16.56
	triang.	0.01	0.06	0.31	1.51	11.74
Memory (MB)						
AMD	8-grid	1.0	8.2	40.1	202.5	1659.2
	4-grid	0.7	5.5	27.1	132.8	1090.2
	triang.	0.6	4.3	18.9	84.9	622.3
METIS	8-grid	1.0	7.6	36.1	166.4	1256.6
	4-grid	0.7	5.3	25.9	117.0	861.9
	triang.	0.6	4.1	17.8	77.0	534.7
NESDIS	8-grid	0.9	7.4	35.1	164.2	1233.2
	4-grid	0.7	5.6	25.9	118.1	860.6
	triang.	0.6	4.1	18.0	77.8	537.7

TABLE 3.1: Effect of graph and re-ordering on Cholesky solver. Results are averaged over 10 iterations.

3.5.1 Evaluation of Cholesky-based Newton's method

3.5.1.1 Computational complexity on synthetic linear systems

We begin with synthetic linear systems of equations in order to assess the effect of different graph structures and reordering methods on the computational complexity. We use three different graph structures: a 4-connected grid, an 8-connected

grid, and a graph generated from a random triangulation of the same image domain. For square images of different sizes, an adjacency matrix for each graph structure was generated with the same number of variables (i.e., triangles were not used as superpixels to reduce the number of variables). We also consider three different ordering methods: approximate minimum degree (Amestoy et al., 2004) (AMD; build into matlab), nested dissection (Lipton et al., 1979) (NESDIS; in the CHOLMOD package (Chen et al., 2008)), and METIS (Karypis and Kumar, 1998). Results are shown in Table 3.1 and are averaged over 10 evaluations.

First, we find that using a triangulation is significantly more efficient than using a pixel grid. When compared to a standard 4-grid on a 500×500 image with AMD reordering, a triangulation uses 622MB of memory versus 1090MB for the grid. This improvement in memory is important: by using a triangulation, in addition to using triangles as superpixels, allows for Cholesky factorization to be used even for large images. Similarly, it takes less than 10 seconds, versus 14 for a pixel grid. We also note that both METIS and NESDIS result in orderings which use less memory, but because they are slower to compute, their overall runtime is higher than that of AMD.

3.5.1.2 Computational complexity on optical flow problems

To assess the effect that different solvers have in real optical flow problems, we replaced the linear solver in our system with different algorithms. We use Cholesky factorization with different re-ordering methods, in addition to several iterative linear solvers, which are often used in other optical flow algorithms. The linear

solvers we use are pre-conditioned conjugate gradients, symmetric LQ factorization, and stabilized biconjugate gradients, all of which are native MATLAB functions. To make these solvers as efficient as possible, each one is preconditioned with an incomplete Cholesky factorization. The use of other preconditioners here may also improve results (Krishnan et al., 2013), although the cost to compute any preconditioner must be taken into account since it needs to be recomputed at each level of the image pyramid. An incomplete Cholesky was used due to its speed and similarity to Cholesky factorization for comparison. We also show two different tolerances: one which is near-exact at 1e-8, and one which is less exact at 1e-3. Note that Cholesky is always exact (to machine precision).

Results are shown in Figure 3.5. The sizes of the linear system varies between images, and so we evaluated results on frame 15 of the cave_2 sequence from the MPI-Sintel training dataset. Note that these timings include the time to compute the ordering, in addition to the solver itself. Athough it is possible to compute the ordering only once per level and reuse it for all iterations, we use a large number of levels and found that the time to compute the ordering was an important factor for the overall runtime of the algorithm, and so included the time in this computation. We find that Cholesky factorization using AMD reordering is at least as fast as other methods, including iterative methods with an in-exact tolerance. When a near-exact tolerance is used, Cholesky factorization is significantly faster than iterative solvers at a similar tolerance. Cholesky decomposition also has the advantage of being invariant to the conditioning of the linear system. In particular, we found that the convergence rate of iterative solvers is dependent on the image and particular linear system involved. For ill-conditioned systems, Cholesky factorization thus has even more of an advantage.

3.5.1.3 Effect of linear solvers of endpoint error

From the results presented thus far, we conclude that Cholesky factorization on triangular grids is superior to iterative solvers on 4- or 8-grids when near-exact results are required. However, when computing optical flow a more pertinent question is: does this result in lower error? We explore this question by running our optical flow method on 100 randomly-chosen images from the MPI-Sintel dataset. In all cases, the parameters are the same (see Section 5.4.2 for more details on the parameters that were used), with the only difference being that different linear solvers are used. We compare Cholesky factorization using AMD reordering to conjugate gradient that is preconditioned with an incomplete Cholesky factorization (PCG). The tolerance for PCG is varied from a near-exact value of 1×10^{-10} up to 1. The results are shown in Figure 3.6.

As before, we find the Cholesky is faster than a near-exact iterative method. However, raising the tolerance does not necessarily correspond to higher error. In fact, the tolerance can be raise up to around 5×10^{-2}, at which point the error remains nearly the same while only 20 seconds were used for solving linear systems compared to 177 for the Cholesky solver. Thus, we conclude that Cholesky-based factorization provides little benefit for computing optical flow, and PCG using a high tolerance will achieve similar results in less time.

The reason that a more exact Cholesky solver does not result in lower error is that the linear system is only very approximately related to endpoint error, the quantity we wish to minimize. Note the number of approximations: first, the MRF model itself is an approximation. The image pyramid then smooths this cost function, so it is an approximation to the finest-level cost function. Within this, we approximate the cost locally using a quadratic function, and further approximate

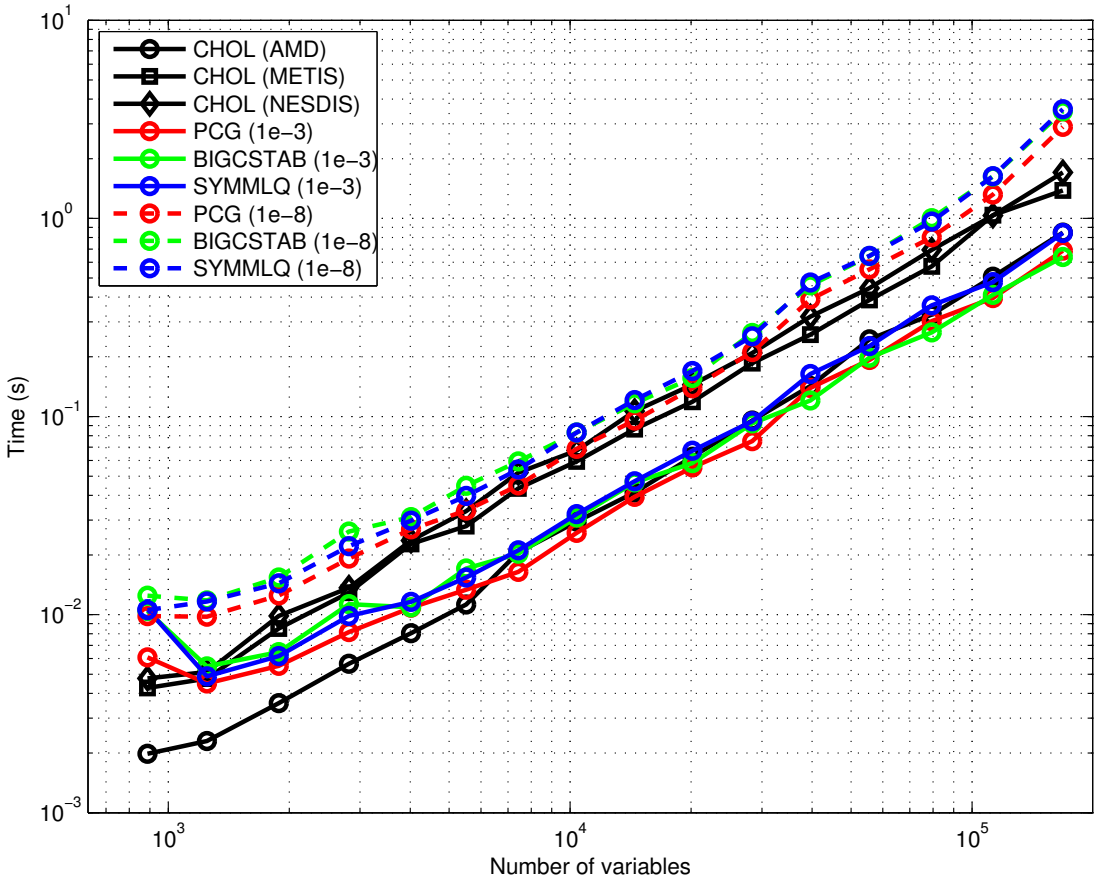


FIGURE 3.5: Comparison of linear solvers at each iteration of optical flow, evaluated on frame 15 of the cave_2 sequence from MPI-Sintel. The system size increases in the coarse-to-fine optimization. Values are averaged over the 10 iterations performed at each level of the image pyramid. Iterative solvers were pre-conditioned with an incomplete Cholesky factorization. Times include computation of the incomplete Cholesky factorization or matrix reordering when used. For iterative algorithms, tolerances of 1e-8 and 1e-3 are used. Cholesky factorization is significantly faster than any of the iterative solvers at a similar tolerance.

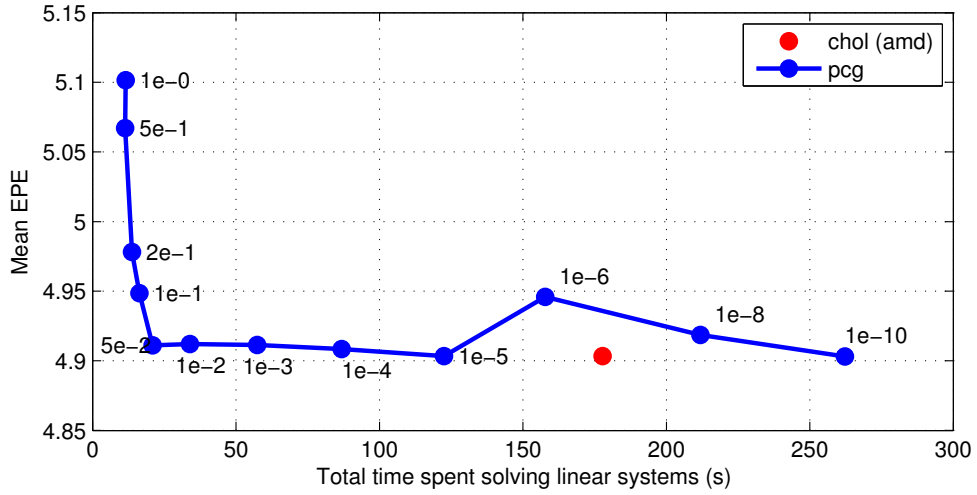


FIGURE 3.6: Effect of linear solvers on mean endpoint error (EPE) for 100 random images from the MPI-Sintel Final training dataset. We compare an exact Cholesky-based solver using AMD reordering to conjugate gradient preconditioned with an incomplete Cholesky factorization. For each point, the tolerance used for PCG is shown. For near-exact tolerance values, Cholesky is faster (and more exact). However, this does not translate to lower EPE, which remains low even for tolerances as high as 5×10^{-2}. These linear solvers also take significantly less time than an exact Cholesky factorization.

the Hessian to ensure that it is positive definite. With all these approximations, it is not surprising that solving the internal linear systems more exactly may not correspond to lower error. Even so, because Cholesky factorization is immune to ill-conditioned systems, it may be useful in cases where this is an issue.

3.6 Summary

In this chapter we have presented a novel optimization framework for the estimation of optical flow. Our method uses a triangulation of the image domain that conforms to the image content. This triangular discretization allows for efficient and straightforward occlusion estimation within the optimization scheme, in addition to allowing for a simple way to impose a non-local smoothness cost. The

triangulation also allows us to use Cholesky decomposition as a method for solving linear systems by both reducing the size of the linear systems and reducing the computational requirements of the factorization itself. Although this approach did not result in direct improvements in endpoint error over iterative linear solvers, it may prove useful in situations where the linear systems are ill-conditioned.

4

Multi-Frame Fusion of Inertial Estimates

A significant challenge for modern optical flow algorithms is when objects move large distances between frames. This is especially true when objects move either into or out of frame, for which there are no matches. In this case, it is often not possible to directly estimate the motion of these pixels from two-frame optical flow. Temporal approaches to optical flow have been presented many times (Murray and Buxton, 1987; Nagel, 1990; Black and Anandan, 1990; Volz et al., 2011), but have nonetheless been used infrequently in modern top-performing algorithms. One explanation for this is that enforcing temporal smoothness is more difficult than enforcing spatial smoothness: temporal derivatives are not always meaningful and so points must be tracked (Black and Anandan, 1990) or more complex parameterizations need to be used (Volz et al., 2011).

In this chapter, we address this problem by proposing a simple method of incorporating temporal information from adjacent frames. We introduce the idea of "inertial estimates" of optical flow and show how this allows for temporal information to be easily added to any optical flow algorithm. By combining this approach

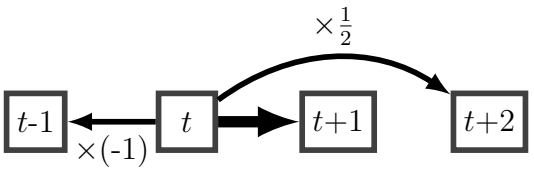


FIGURE 4.1: Inertial flow estimates used in multi-frame fusion. In addition to using the two-frame estimate $[t \to t+1]$ directly, we also estimate the flow from $[t \to t-1]$ and from $[t \to t+2]$. These two estimates are then multiplied by the factors -1 and $\frac{1}{2}$, respectively, to give an estimate of the desired flow $[t \to t+1]$. All three flow estimates are then fused using a classifier.

with the continuous optimization framework presented in Chapter 3, we achieve state-of-the-art results on the difficult MPI-Sintel dataset.

4.1 Inertial estimates

Let $[t \to (t+1)]$ denote the estimated flow between frames t and $t+1$, and suppose that we also have access to frames $t-1$ and $t+2$. If it is assumed that objects move at a constant velocity (i.e., they are carried by inertia) and move parallel to the image plane, then an estimate of the motion from $[t \to (t+1)]$ is given by $-[t \to (t-1)]$, which is found by computing the flow from t to $t-1$ and negating it, as shown in Figure 4.1. Similarly, another estimate can be found using frame $t+2$ as $\frac{1}{2}[t \to (t+2)]$. We call these "inertial estimates" since they provide an estimate of the flow by assuming that inertia moves all objects at a constant velocity.

Of course, these estimates will, on average, be inferior to using $[t \to (t+1)]$ directly. However, if an object is visible in frame t and moves out of frame in $t+1$, then it may still be visible in $t-1$ and so $-[t \to (t-1)]$ will likely give a better estimate for that part of the image. Similarly, using the estimate $\frac{1}{2}[t \to (t+2)]$ will provide an additional source of information.

4.2 Classifier-based fusion

These three optical flow estimates must then be fused together. We do so by training a random forest classifier whose output tells us which estimate to use for each pixel in order to minimize endpoint error. We use the following features:

- The tail probability for the Cauchy distribution used in the match cost $\mathcal{D}(\cdot)$. This value varies from 0 to 1 with larger values indicating a better match. The index of the flow estimate with the best score, and its associated score, are also used.
- Each pixel in frame t is projected forward via the flow estimate and then projected back using the backward flow. The Euclidean norm of this discrepancy vector is used as a feature for each flow estimate. The index of the flow estimate with the smallest discrepancy and its corresponding value are also included as features.
- The flow estimates u and v, and the magnitudes $\sqrt{u^2 + v^2}$.
- The multiplicative offset $m(\mathbf{x})$ at each pixel.
- For every pixel, an indicator of whether the pixel is estimated to be occluded.
- The (x, y) location of each pixel.

This results in a total of 27 features. We sampled a number of points uniformly at random from each training image such that the resulting dataset had $\sim 10^6$ observations.

In this classification problem, not all data points should be counted equally. In particular, a misclassification is more costly when the three inertial estimates

FIGURE 4.2: Examples of our multi-frame fusion on the MPI-Sintel Final training set. **Top row:** Frame at time t. **Rows 2-4:** Inertial estimates of the flow $[t \to t+1]$, $-[t \to t-1]$, and $\frac{1}{2}[t \to t+2]$. **Row 5:** Fusion classification for each pixel. Color indicates the estimate used at each pixel. Colors correspond to the border colors of the inertial estimates. **Row 6:** Fused flow estimate. **Bottom row:** Groundtruth flow. For all flow estimates, the endpoint error is printed in the image.

have very different errors. To take this into account, each data point was weighted by the difference between the lowest endpoint error of all three flow estimates and the mean of the other two. This weighting indicates how important each datapoint is. We then trained a random forest classifier with 500 trees using Matlab's TreeBagger class. An example of our fusion is shown in Figure 4.2 on the MPI-Sintel dataset (Butler et al., 2012).

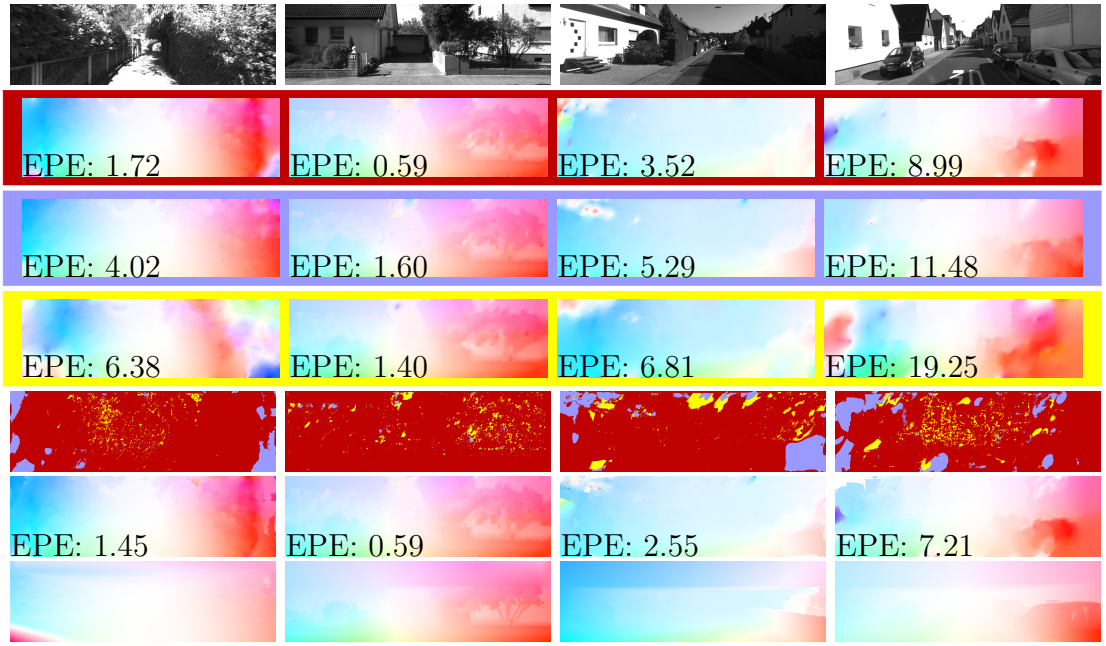


FIGURE 4.3: Examples of our multi-frame fusion on the KITTI training dataset. **Top row:** Frame at time t. **Rows 2-4:** Inertial estimates of the flow $[t \rightarrow t+1]$, $-[t \rightarrow t-1]$, and $\frac{1}{2}[t \rightarrow t+2]$. **Row 5:** Fusion classification for each pixel. Color indicates the estimate used at each pixel. Colors correspond to the border colors of the inertial estimates. **Row 6:** Fused flow estimate. **Bottom row:** Groundtruth flow, interpolated using linear interpolation. For all flow estimates, the endpoint error is printed in the image.

As a final step in our procedure, a median filter was applied to the fused flow estimates.

4.3 Experiments

We evaluate our algorithm quantitatively on three datasets. The free parameters of our method are $\tau_0, \tau_1, \tau_2, \tau_3$, the value of α used in the smoothness terms $\mathcal{S}_1(\cdot)$ and $\mathcal{S}_2(\cdot)$, and the spacing of the uniform grid used in the triangulation. For $\mathcal{S}_3(\cdot)$,

α was set to 0.5. Parameters were chosen for each dataset using a small-scale grid search on the training data.

Our basic method is denoted as TF (TriFlow), and when occlusion estimation, multi-frame fusion and median filtering are used, they are denoted as "O" (occlusion), "F" (fusion) and "M" (median filtering), respectively. Our final method with all components is thereby denoted as TF+OFM.

4.3.1 Middlebury

We begin with the Middlebury dataset (Baker et al., 2011) since it is a standard benchmark for optical flow, although it has only small and simple motions. For this dataset, the parameters were set to $\tau_0 = 0, \tau_1 = 3.5, \tau_2 = 0, \tau_3 = 25, \alpha = 0.36$, and a small grid spacing of 2 pixels was used in order to capture the small details in this dataset.

Results on the test dataset are given in Table 4.1. Note that we did not evaluate our multi-frame fusion since the dataset was too small for a reliable classifier to be trained. Our results are comparable to other similar coarse-to-fine methods such as DeepFlow (Weinzaepfel et al., 2013). Our occlusion estimation provides little benefit in this case, since the dataset has very small occlusion regions.

4.3.2 MPI-Sintel

The MPI-Sintel dataset (Butler et al., 2012) is a large, difficult dataset that includes difficulties such as large displacements, significant occlusions and atmospheric effects. Parameters were set to $\tau_0 = 0.5$, $\tau_1 = 2.0, \tau_2 = 0, \tau_3 = 100, \alpha = 0.6$, and the grid spacing was set to 5 pixels.

	Army	Meq.	Sch.	Wood.	Grove	Urban	Yos.	Teddy	mean
TF+OM	0.10	0.22	0.36	0.20	0.98	0.56	0.16	0.76	0.42
Layers++ (Sun et al., 2010b)	0.08	0.19	0.20	0.13	0.48	0.47	0.15	0.46	0.27
MDP-Flow2 (Xu et al., 2012)	0.09	0.19	0.24	0.16	0.74	0.46	0.12	0.78	0.35
DeepFlow (Weinzaepfel et al., 2013)	0.12	0.28	0.44	0.26	0.81	0.38	0.11	0.93	0.42
LDOF (Brox and Malik, 2011)	0.12	0.32	0.43	0.45	1.01	0.10	0.12	0.94	0.56

TABLE 4.1: Endpoint error on the Middlebury test dataset. Our results are comparable with similar coarse-to-fine methods.

Results on the MPI-Sintel test dataset are given in Table 4.2. Our method is currently rank second on this dataset, behind only EpicFlow (Revaud et al., 2014) in terms of endpoint error. Our results are especially good for unmatched pixels which are helped by our occlusion term and multi-frame fusion. In particular, on the Final dataset the occlusion term improves the error on unmatched pixels by 6.4% and the fusion improves it by an additional 7.2%.

Several examples of results from our multi-frame fusion for the Final version of the training dataset are shown in Figure 4.2, with additional examples in Appendix A. As we would expect, the inertial estimates that the classifier selects are spatially localized around the edges of objects where occlusions occur. In all cases, the multi-frame fusion significantly reduces the endpoint error.

Figure 3.4 shows several examples of the occlusion estimates on images from MPI-Sintel. During optimization, the occlusion status of each quadrature point in each triangle is estimated. For visualization, we label each triangle a value in $[0, 1]$ as the proportion of its quadrature points that are labeled as occluded, and then each

	Final			Clean		
	EPE	matched	unmatched	EPE	matched	unmatched
TF+OFM	6.727	3.388	33.929	4.917	1.874	29.735
TF+OF	6.780	3.436	34.029	4.986	1.937	29.857
TF+O	7.164	3.547	36.657	5.357	2.033	32.474
TF	7.493	3.609	39.170	5.723	2.077	35.471
EpicFlow (Revaud et al., 2014)	**6.285**	**3.060**	**32.564**	**4.115**	**1.360**	**26.595**
DeepFlow (Weinzaepfel et al., 2013)	7.212	3.336	38.781	5.377	1.771	34.751
FC-2Layers-FF (Sun et al., 2013)	8.137	4.261	39.723	6.781	3.053	37.144
MDP-Flow2 (Xu et al., 2012)	8.445	4.150	43.430	5.837	1.869	38.158
LDOF (Brox and Malik, 2011)	9.116	5.037	42.344	7.563	3.432	41.170

TABLE 4.2: Results on the MPI-Sintel test set. Our algorithm is currently ranked second on this dataset. The largest improvement over other methods is on unmatched pixels due to our occlusion estimation and multi-frame fusion.

pixel is labeled based on the triangles that it overlaps. The occlusion term is able to estimate the occlusions accurately, which results in reduced error.

4.3.2.1 Violation of inertial estimate assumptions

The inertial estimates of optical flow are based on the assumption that the motion of objects is constant and parallel to the image plane, which results in a constant displacement between subsequent pairs of frames. We can estimate how much this assumption is violated by measuring the Euclidean distance between the offsets in subsequent pairs of frames for unoccluded pixels. This is shown in Figure 4.4. We find that 70% of all pixels on the MPI-Sintel training set have a deviation between their displacements in subsequent pairs of frames of less than a pixel, and the displacement changes by less than 10 pixels for 90% of unoccluded pixels.

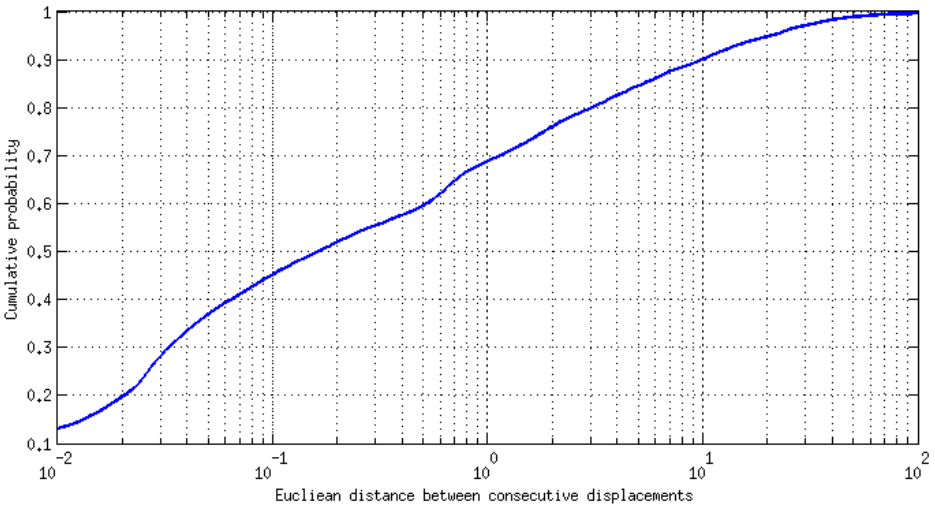


FIGURE 4.4: Violation of the assumption of inertial estimates on MPI-Sintel. For each unoccluded pixel, we plot the Euclidean distance between two successive motion estimates. Seventy percent of pixels have a deviation of less than one pixel and 90% deviate less than 10 pixels.

This indicates that the assumptions made by the inertial estimates hold for a large number of points in the dataset, which is also corroborated by the improved results on the optical flow evaluation.

4.3.3 KITTI

The KITTI dataset (Geiger et al., 2012) consists of grayscale images taken from a moving vehicle. We used the parameter settings $\tau_0 = 0.05$, $\tau_1 = 0.02, \tau_2 = 7, \tau_3 = 125, \alpha = 0.6$, and the grid spacing was set to 5 pixels.

On the KITTI test dataset, error is measured as the percentage of pixels with an endpoint error greater than 3, in addition to the standard endpoint error. Our results on this dataset are given in Table 4.3. This dataset is quite different than MPI-Sintel: the images are grayscale and have low contrast and the motions are often dominated by that of the camera. Top-performing methods on this

	EPE (all)	EPE (not occ.)	% > 3 (all)	% > 3 (not occ.)
TF+OFM	5.0	2.0	18.46%	10.22%
PCBP-Flow (Yamaguchi et al., 2013)	2.2	0.9	8.28%	3.64%
DeepFlow (Weinzaepfel et al., 2013)	5.8	1.5	17.79%	7.22%
LDOF (Brox and Malik, 2011)	12.4	5.6	31.39%	21.93%
DB-TV-L1 (Zach et al., 2007)	14.6	7.9	39.25%	30.87 %

TABLE 4.3: Results on the KITTI test set. We show both the endpoint error (EPE) and ther percentage of pixels with an EPE more than 3, for all pixels as well as non-occluded pixels.

dataset take advantage of these properties by using better features such as census transforms and more information such as stereo and epipolar information (Yamaguchi et al., 2013). However, our results are comparable to similar coarse-to-fine approaches such as DeepFlow (Weinzaepfel et al., 2013), especially for endpoint error (which the fusion classifier was trained to minimize). Examples of our fusion are shown in Figure 4.3, with additional examples in Appendix A.

We also evaluate the effect of our occlusion and fusion terms on a validation set from the training images. For this, 100 training images were used to train a fusion classifier and evaluation was done on remaining 94 images. These results are shown in Table 4.4. Both the occlusion and multi-frame fusion terms significantly improve results, as measured by either endpoint error or the percentage of pixels with and endpoint error more than 3.

	EPE	% > 3
TF+OFM	4.23	16.43%
TF+OF	4.32	16.62%
TF+O	5.29	16.91%
TF	6.89	19.96%

TABLE 4.4: Results on a validation set from the KITTI training dataset. The occlusion estimation term and multi-frame fusion significantly improve results.

4.3.4 Timing

Timing was evaluated on a laptop with a 1.80 GHz Intel Core i5 processor and 4 GB of RAM. The typical time taken for two-frame flow estimation on a 1024×436 image from MPI-Sintel, (excluding multi-frame fusion), was 500 seconds. About half of this time is spent evaluating the cost function, and another 20% is spent solving linear systems in the Newton-based optimization. Much of our approach can be sped up through parallelization. For example, the cost function evaluation, running the algorithm on all three inertial estimates, and the random forest fusion are all trivially-parallelizable.

4.3.5 Tears of Steel

We also qualitatively assess our algorithm on *Tears of Steel* (Roosendaal, 2012), a short film, for which ground truth is not available. Tears of Steel was produced by the Blender Foundation and is freely available online. In contrast to Sintel, it is not fully synthetic but rather has visual effects added to live action. We chose three sequences of 10 frames from the film and ran our algorithm using the same settings as for MPI-Sintel. The results for several frames are show in Fig. 4.5.

FIGURE 4.5: Qualitative results on Tears of Steel. Parameters are the same as for MPI-Sintel.

The flow estimates are all quite plausible, indicating that our method is able to generalize well to other similar sequences.

4.4 Summary

In this chapter we have presented a method for incorporating temporal information into optical flow estimation through the use of inertial estimates. While the assumptions made by inertial estimates are often violated, we have found that they are valid on a large enough subset of pixels that fusing them using a classifier can yield a substantial reduction in error. This approach is quite simple and does not involve complex temporal MRFs or point tracking over multiple frames.

5

Hierarchically-Constrained Optical Flow

The space of possible offsets in optical flow problems can be modeled as either continuous or discrete. Discrete labeling approaches have been applied successfully to many image correspondence problems (Kolmogorov, 2006), but they can be computationally demanding which limits their use to problems with small label spaces. For optical flow, the set of possible labels is extremely large and so these problems are often solved using a continuous label space (Brox et al., 2004), which allows for both efficient local optimization methods and sub-pixel offsets. However, these methods are prone to finding suboptimal local minima. In order to better avoid such minima, coarse-to-fine optimization is often employed (Brox et al., 2004) along with a global matching of sparse, discriminative features (Brox and Malik, 2011). Even so, these methods have no guarantee of being near a global optimum.

In this chapter, we propose the use of a tree-based graphical model that leverages a hierarchical segmentation of the image. We show how the global optimum of this discrete optimization problem can be found using efficient optimization methods,

borrowing ideas from the literature on deformable-parts models for object recognition (Felzenszwalb et al., 2010a). This allows us to optimally solve large image correspondence problems with a global smoothness model.

Additionally, we describe a simple method of incorporating information from multiple frames in optical flow. We use the idea of *inertial estimates* from Chapter 4, where several estimates of the optical flow are computed using nearby frames and subsequently fused using a classifier. We show how the inertial estimates can instead be directly taken into account in our cost function.

In Section 5.4.4, we show that state-of-the-art motion estimation schemes based on local optimization can have difficulties even on relatively simple motion analysis problems that contain large displacements. We illustrate this issue with both synthetic datasets and real images and show how our proposed global method can significantly improve performance in these situations. We also evaluate the proposed method on the challenging MPI-Sintel dataset (Butler et al., 2012) as well as the Middlebury dataset (Baker et al., 2011) and compare its performance to other recent methods.

5.1 Problem setup

Let $I_1, I_2 : (\Omega \subseteq \mathbb{R}^2) \rightarrow \mathbb{R}^d$ be two d-dimensional images, where d is typically 3 for color images and 1 for grayscale images. The image domain is denoted by Ω, which we consider to be a discrete set of pixel locations.

Our algorithm proceeds by constructing a tree-structured Markov Random Field (MRF) model based on a hierarchical segmentation of the image I_1 as depicted in Figure 5.1. The goal of the motion estimation procedure is to assign an integral

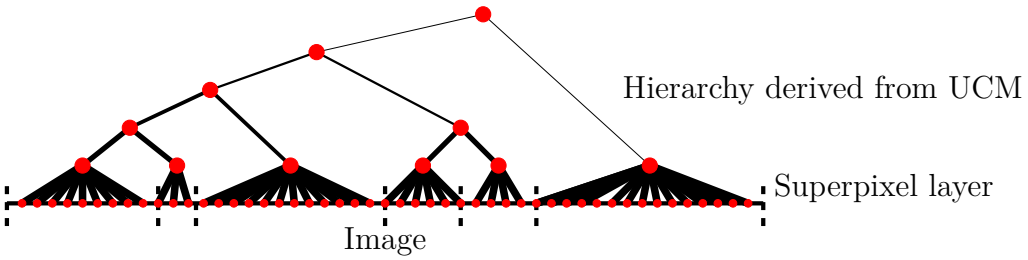
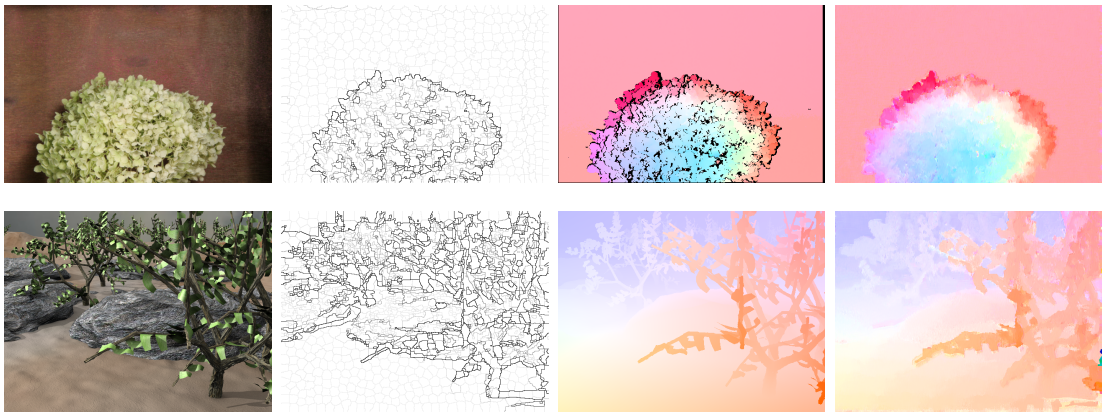


FIGURE 5.1: Depiction of our segmentation-based hierarchical model, shown as 1D for simplicity. The image is segmented into a hierarchical segmentation, represented using an ultrametric contour map (UCM). At the base of the hierarchy, the image is divided into superpixels, denoted by dashed black lines. A node in the graph is associated with each superpixel, which are connected to their associated pixel variables. Each segment in the segmentation hierarchy has an associated variable and is connected to its children in the hierarchy, with the root of the tree representing the entire image. The weights of our graphical model's edges are a function of the weights in the UCM, denoted here by the thickness of black edges. Variables in the graphical model are denoted by red circles. After optimization, the final motion estimate consists of the labels that are applied to the pixel variables at the base of the hierarchy.

offset to each vertex that maps it onto its correspondent in I_2. This is done by defining a cost function that is small when pixels have a good correspondence and when each node has a similar offset to its parent. The goal of the optimization procedure is to find a solution which minimizes this cost function. The steps in this process are described in more detail in the following subsections.

5.1.1 Hierarchical segmentation

We base our hierarchical segmentation of I_1 on the framework of (Donoser and Schmalstieg, 2014), which is in turn based on the ultrametric contour map (UCM) hierarchy of (Arbelaez, 2006; Arbelaez et al., 2011). Rather than using a watershed segmentation as the the base of the hierarchy, we begin with a segmentation using SLIC superpixels (Achanta et al., 2012). By using SLIC, we are able to explicitly control the size and complexity of the base superpixel layer. In our experiments,



(E) First image (F) Segmentation (G) Groundtruth (H) HCOF

FIGURE 5.2: Estimated optical flow using our method HCOF on two images from the Middlebury (Baker et al., 2011) dataset. The values of the segmentation are the weights used in the pairwise terms of the MRF, using the trained logistic curve given in Equation (5.1).

we set the region size and regularization parameters to 50, which we found resulted in superpixels that are small enough to contain primarily a single motion but are large enough for an efficient algorithm.

Examples of segmentations obtained with this procedure are shown in Figure 5.2. The segmentation divides the image into a set of small superpixels that are then successively merged with their most similar neighbors until the entire image is a single segment, forming a tree structure. Distances in the hierarchy encode the similarity of pixels and the likelihood that they belong to the same object (Arbelaez et al., 2011).

5.1.2 Graphical model

We construct a MRF model which mirrors the structure of the tree produced by the hierarchical segmentation (Figure 5.1). Let the graph be denoted by $G = (\mathcal{V}, \mathcal{E})$, for vertex set $\mathcal{V}$ and edge set $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. The vertex set $\mathcal{V}$ corresponds to the

nodes in the segmentation tree; the leaf nodes correspond to the individual pixels and the interior nodes correspond to merged regions produced by the hierarchical segmentation. Each vertex has an associated *location*, *area* and *similarity weight*.

The location of a vertex in I_1 is denoted by the function $\mathbf{x} : \mathcal{V} \to \Omega$. For vertices corresponding to pixels, this is defined simply as the location of the pixel. For internal nodes, we arbitrarily define it as the location nearest to the region's centroid. The area of each vertex is denoted by $\mathbf{a} : \mathcal{V} \to \mathbb{R}_+$. This area is 1 for the vertices corresponding to individual pixels, and for internal nodes the area is the sum of the areas of its children. The similarity weight of each interior vertex is denoted by $\mathbf{s} : \mathcal{V} \to \mathbb{R}_+$, and is defined as the similarity of v's two children. Specifically, our chosen segmentation algorithm (Donoser and Schmalstieg, 2014) specifies an ultrametric distance $\mathbf{d}(v)$, which is the level at which the regions corresponding to v's children are merged in the segmentation process. The similarity weight is then set using a logistic function:

$$\mathbf{s}(v) = 1/\left(1 + e^{\eta_1(\mathbf{d}(v)+\eta_0)}\right) . \tag{5.1}$$

Edges in the graph are placed between all nodes sharing a parent-child relationship in the image segmentation hierarchy:

$$\mathcal{E} = \{(v_p \in \mathcal{V}, v_c \in \mathcal{V}) \mid v_p \text{ is the parent of } v_c\}. \tag{5.2}$$

Each edge $(v_p, v_c) \in \mathcal{E}$ is assigned a weight that determines how similar the offsets of a node v_p and its child v_c should be, which will be used in the smoothness term of our cost function. This weight is denoted by $\mathbf{w} : \mathcal{V} \times \mathcal{V} \to \mathbb{R}$, and is defined as

$$\mathbf{w}(v_p, v_c) = \mathbf{a}(v_c)\left[\mathbf{s}(v_p) + (1 - \mathbf{s}(v_p))\, e^{-\tau \mathbf{a}(v_c)}\right] . \tag{5.3}$$

This weight does the following: (1) it weights the edge by the area of the child node so that the smoothness term operates on all regions equally, (2) it weights the edge using the region similarity $\mathbf{s}(v_p)$ and (3) it up-weights the edge for small regions to encourage them to be better connected to their parents, modulated by the parameter τ.

5.1.3 Cost function

We denote the image motions using a function $\mathbf{u} : \mathcal{V} \to \mathbb{Z}^2$, and we say that vertex v is matched to location $\mathbf{x}(v) + \mathbf{u}(v)$ in I_2. The cost function is defined over the graph structure for this displacement function $\mathbf{u}$:

$$\mathcal{C}(\mathbf{u}) = \lambda_0 \sum_{v \in \mathcal{V}} \mathcal{P}_v(\mathbf{u}(v)) + \lambda_1 \sum_{v \in \mathcal{V}} \mathcal{D}_v(\mathbf{u}(v)) + \sum_{(v_p, v_c) \in \mathcal{E}} \mathcal{S}_{v_p, v_c}(\mathbf{u}(v_p), \mathbf{u}(v_c)) \; . \quad (5.4)$$

Here, $\mathcal{P}_v(\mathbf{u}(v))$ is a prior term that encourages each node to have a small offset. The term $\mathcal{D}_v(\mathbf{u}(v))$ is a unary matching term that measures how well vertex v is matched, and $\mathcal{S}_{v_p, v_c}(\mathbf{u}(v_p), \mathbf{u}(v_c))$ is a smoothness term defined over the set of edges. The parameters λ_0 and λ_1 control the tradeoff of these terms.

We set $\mathcal{P}_v(\mathbf{u}(v)) = \mathcal{D}_v(\mathbf{u}(v)) = 0$ for all nodes that do not correspond to pixels at the base of the hierarchy. In other words, our data term affects only the leaves. The hierarchy thus is used only for enforcing smoothness. It is possible to add unary costs to internal nodes to incorporate region information, but we do not consider this here.

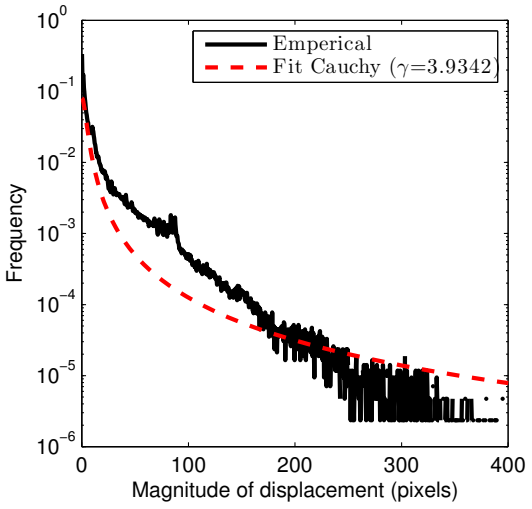


FIGURE 5.3: Distribution of offsets in MPI-Sintel. A Cauchy distribution was fit to the data, for which $\gamma = 3.9342$. This Cauchy distribution is used as our spatial prior term.

5.1.3.1 Spatial prior term

The spatial prior term encourages pixels to have small displacements. For this term, we fit a Cauchy distribution to the set of groundtruth offsets in the MPI-Sintel dataset, as shown in Figure 5.3. The spatial prior term is then the negative log-likelihood of the associated Cauchy distribution with $\gamma_{prior} = 3.3942$:

$$\mathcal{P}_v(\mathbf{u}(v)) = \log\left[\pi\left(\mathbf{u}(v)^2 + \gamma_{prior}^2\right)/\gamma_{prior}\right] \ . \tag{5.5}$$

5.1.3.2 Matching term

We use a data matching term that is composed of both a color and a gradient component. Let I_1 and I_2 be two images in the Lab color space. For color features, we construct a 5×5 array of cells, centered at a given pixel. The size of the cells

can be varied in order to accumulate a larger or smaller contextual area. All pixel values within each cell are averaged, and this is applied to all three channels of the Lab color images, resulting in a 75-dimensional feature vector. Distances between feature vectors are computed using an L1 distance function. Let $[\cdot]_C$ be a function that maps an image to its 75-dimensional color feature vector. The cost function is then given by:

$$\mathcal{D}_v^{LAB}(\mathbf{u}(v)) = \|[I_1]_C(\mathbf{x}(v)) - [I_2]_C(\mathbf{x}(v) + \mathbf{u}(v))\|_1 \ . \tag{5.6}$$

The gradient component of our data cost function uses SIFT features (Lowe, 2004) computed densely at every pixel. Let $[\cdot]_S$ be a function that maps an image to its 128-dimensional SIFT features computed densely at each pixel location in Ω. We use an L1 cost function:

$$\mathcal{D}_v^{SIFT}(\mathbf{u}(v)) = \|[I_1]_S(\mathbf{x}(v)) - [I_2]_S(\mathbf{x}(v) + \mathbf{u}(v)))\|_1 \ . \tag{5.7}$$

The final cost function is a weighted linear combination of the color and SIFT costs:

$$\mathcal{D}_v(\mathbf{u}(v)) = \alpha\mathcal{D}_v^{LAB}(\mathbf{u}(v)) + (1-\alpha)\mathcal{D}_v^{SIFT}(\mathbf{u}(v)) \ . \tag{5.8}$$

Note that the only desideratum of the distance function is that it be computationally efficient; it need not be a metric, differentiable, convex, or even continuous. This allows us to use SIFT features in dense optical flow without computing matches beforehand, and opens the door to more complex descriptors and distances.

5.1.3.3 Smoothness term

The smoothness term encourages vertices to have a similar offset to their parent. We use a weighted $L1$ penalty,

$$\mathcal{S}_{v_p,v_c}(\mathbf{u}(v_p),\mathbf{u}(v_c)) = \mathbf{w}(v_p,v_c)\|\mathbf{u}(v_p)-\mathbf{u}(v_c)\|_1 \ , \tag{5.9}$$

for v_p the parent of v_c, where $\mathbf{w}(v_p,v_c)$ is the associated edge weight as defined in Equation (5.3).

Note that this tree based smoothness term is a significant departure from more traditional smoothing functions which are usually formulated on a graph that connects each image region to all of its neighbors. It has been previously observed that people's perceptual organization of objects in an image is hierarchical in nature (Martin et al., 2001); we likewise argue that this tree-based smoothing term captures most of the salient features of the true motion field, while yielding a tractable optimization problem.

5.2 Optimization

Our model can be though of as a large "deformable parts" model (DPM) (Felzenszwalb et al., 2010a), which is a widely-used framework in object recognition. In a DPM, an object is represented by a tree that models how the parts of the object are connected to each other. Each node has a unary cost function that reflects its preference for various matches, and each edge in the tree is associated with a smoothness function which constrains the relative motion of the two parts. Although our model is more complex – our tree describes an entire image rather than

a single object – the optimization problem is similar: in both cases the model is a tree-structured MRF and the label space captures the set of possible displacements.

The relationship between our model and a DPM allows us to leverage optimization techniques first developed within the DPM literature. First, because our graph is a tree, the minimum-energy solution can be found in polynomial time using a generalization of the Viterbi algorithm (Felzenszwalb and Huttenlocher, 2005). Also, because we use an $L1$ distance function, the cost matrices at each node can be computed very efficiently using a linear-time distance transform (Felzenszwalb and Huttenlocher, 2004).

To briefly summarize this procedure, the cost for each node is computed as a function of its children. This process begins at the leaf (pixel) nodes, and the cost matrices of each internal node are computed by applying distance transforms to the cost matrices of its children and summing them. We also compute and store a Voronoi array that specifies the optimal label for each vertex given a label for its parent. When the root of the tree is reached, the optimal cost for the entire model is found, and the labels of each vertex are set by backtracing back down the tree using the stored Voronoi arrays.

To more formally specify this optimization procedure, we first note that the cost function given in Equation (5.4) can be rewritten recursively with respect to each subtree:

$$\mathcal{C}_v(\mathbf{u}) = \lambda_0 \mathcal{P}_v(\mathbf{u}(v)) + \lambda_1 \mathcal{D}_v(\mathbf{u}(v)) + \sum_{(v,v_c)\in\mathcal{E}} \{\mathcal{S}_{v,v_c}(\mathbf{u}(v), \mathbf{u}) + \mathcal{C}_{v_c}(\mathbf{u}(v_c))\} \ . \quad (5.10)$$

In words, this says that the cost of the subtree corresponding to vertex v is equal to the sum of its unary costs (the prior and data terms), the smoothness costs to

its children, and then the recursively-computed costs of each of its children. The cost of the full model is then $\mathcal{C}_{v_0}(\mathbf{u})$, where v_0 is the root vertex of the tree. Note that in our setup, the unary cost terms are zero for all nodes other than the leaves of the tree, but we include it in the optimization procedure here for completeness.

The goal of the optimization is to compute the minimum-cost solution: $\min_{\mathbf{u}} \mathcal{C}_{v_0}(\mathbf{u})$. We do so using dynamic programming, by recursively defining the cost of each vertex's label as a function of the best placement of its children,

$$\mathcal{C}_v(\mathbf{u}(v)) = \lambda_0 \mathcal{P}_v(\mathbf{u}(v)) + \lambda_1 \mathcal{D}_v(\mathbf{u}(v)) + \sum_{v_c:(v,v_c)\in\mathcal{E}} \min_{\mathbf{u}(v_c)} \{\mathcal{S}_{v,v_c}(\mathbf{u}(v), \mathbf{u}(v_c)) + \mathcal{C}_{v_c}(\mathbf{u}(v_c))\} \ . \tag{5.11}$$

The optimal cost is then given by $\min_{\mathbf{u}(v_0)} \mathcal{C}_{v_0}(\mathbf{u}(v_0))$ for root node v_0. Note that since the leaves of the tree have no children, $\mathcal{C}_v(\mathbf{u}(v)) = \lambda_0 \mathcal{P}_v(\mathbf{u}(v)) + \lambda_1 \mathcal{D}_v(\mathbf{u}(v))$ for all leaf nodes v. Thus, at the leaf nodes only the unary costs need to be computed. For all other nodes, the cost matrices are recursively computed using Equation (5.11). At each node v, we store the cost matrix $\mathcal{C}_v(\mathbf{u}(v))$ that specifies the optimal cost of the subtree model for each position of the node v. In addition to the costs, we also store matrices that specify the optimal *location* of the node for a given position of its parent: these location matrices are the Voronoi matrices associated with the given cost function.

The optimization thus begins at the leaves of the tree and proceeds to the root where the optimal location of the root is selected, yielding the optimal cost of the full model. The locations of all other nodes are then determined by recursively tracing back down the tree using the stored Voronoi matrices.

5.2.0.4 Computation using generalized distance transforms

In order to have an efficient algorithm, the computation of the right side of Equation (5.11) should be done as efficiently as possible. Consider this equation for isolated for a single vertex v' given a vertex v:

$$f_v(v') = \min_{\mathbf{u}(v')} \{\mathcal{S}_{v,v'}(\mathbf{u}(v), \mathbf{u}(v')) + \mathcal{C}_{v'}(\mathbf{u}(v'))\} \ . \tag{5.12}$$

The function $\mathcal{S}_{v,v'}(\cdot)$ is a distance measure (in our case, a weighted ℓ_1 norm), and $\mathcal{C}_{v'}(\cdot)$ is a unary cost. Intuitively, this is trying to find a value of $\mathbf{u}(v')$ which is close to $\mathbf{u}(v)$ and also has a low unary cost. This is exactly the form of the *generalized distance transform* as given in (Felzenszwalb and Huttenlocher, 2004). In (Felzenszwalb and Huttenlocher, 2004), a linear-time algorithm is given for computing generalized distance transforms for several distance measures, including the squared Euclidean distance and the ℓ_1 norm. The algorithm is based on the observation that generalized distance transforms are equivalent to min-convolutions, which is a convolution where multiplication and summation are replaced with addition and taking the minimum, respectively.

The overall runtime of our algorithm is $O(nk)$ where n is the number of pixels and k is the number of MRF variables. Details on this optimization procedure as applied to deformable parts models for object recognition can be found in (Felzenszwalb and Huttenlocher, 2005; Felzenszwalb et al., 2010a).

5.2.1 Implementation details

Even though our tree-based MRF model can be optimally solved in polynomial time, a naive implementation would still be computationally demanding because of the number of nodes and the size of the label space. We make the following changes to the algorithm so that it can be computed on a standard computer in a reasonable amount of time with high accuracy. Some of these details introduce approximations, and although the final solution may no longer be exactly the minimum-energy solution for our cost function, we show in Section 5.4.8 that the resulting solutions are virtually indistinguishable from the true global optima.

Subsampling of superpixels We assume that the superpixels at the base of the segmentation hierarchy are sufficiently small that the optimal offsets of their constituent pixels will be very similar. This implies that their cost matrices will also be very similar, and so rather than computing a separate cost matrix for every pixel within a superpixel, we only compute the cost matrix for a small set of randomly-sampled constituent pixels. In practice, we use only 10 pixels per superpixel, and since each superpixel may contain hundreds or thousands of pixels, this significantly reduces the computational complexlity.

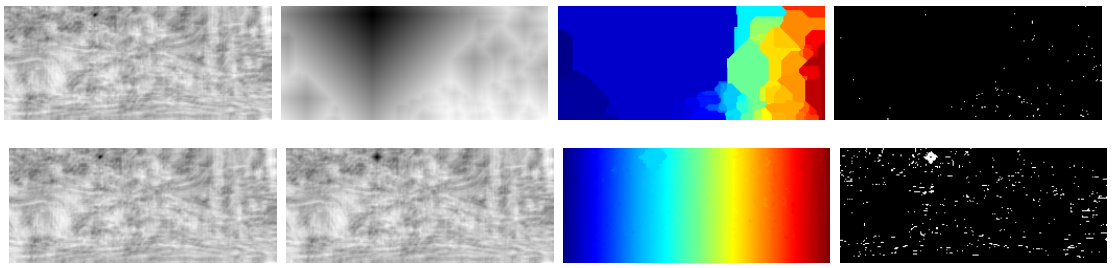
However, this creates a problem when backtracing down the tree to label each pixel. Because a cost matrix is not created for all pixels at the base of the hierarchy, they cannot all be directly labeled. We could circumvent this problem by computing the full cost matrix for each pixel only when it needs to be labeled at the end of the backtracing step, but this is still time consuming. Instead, we assume that a pixel's offset will be close to its parent's offset and only compute the cost in a small window around the parent's offset. In practice, we use a window that has a radius that is either 2 pixels or 20% of the magnitude of the parent's offset, whichever is

larger. Because a large majority of pixels in many datasets have small motions, this is significantly more efficient than computing the full cost matrix for each pixel when backtracing.

Subsampled cost matrices When performing the upward pass of optimization, we do not compute a full cost matrix for each node. Instead, we only compute every k^{th} pixel in a grid for some value of k. Because neighboring pixels will have similar costs, this results in a minimal decrease in accuracy while significantly speeding up the optimization. This strategy effectively reduces the number of labels on the upwards pass by only allowing each node to take every k^{th} label. However, at the final level of the downward pass, we compute a dense cost matrix for each pixel in the vicinity of its parent's optimal displacement and use this to determine its label. Thus, the size of the label space for the pixels is not reduced.

Sub-pixel localization Although our method is discrete, sub-pixel estimates may improve results. We do so in a similar way to (Fortun et al., 2014). At the pixel-level of the downward pass of the optimization, a cost matrix for each pixel is computed as mentioned previously. Let (x, y) be the location of the minimum value in the pixel-level cost matrix. We fit a parabola to locations $(x-1, y), (x, y), (x+1, y)$ and analytically locate the minimum. The y-location is treated similarly by finding the minimum of a parabola fit to the values $(x, y-1), (x, y), (x, y+1)$. Note that because (x, y) is the minimum location, the two quadratics will have a well-defined minimum near the location (x, y). Because the minimum of the two quadratics can be found analytically, this results in a negligible increase in runtime.

Compressing the cost matrices The values within the cost matrices need



(A) Cost matrix (B) Distance transform (C) Voronoi diagram (D) Indices stored

FIGURE 5.4: The Voronoi diagram encodes the optimal label of a node given its parent's location. It can be compressed two ways. The last column shows the entries that need to be stored of the cost matrix (first row) or Voronoi diagram (second row). **Top row:** When the smoothness cost is small, the Voronoi diagram is dominated by a few entries. Only a few locations of the cost matrix are needed to reproduce the Voronoi diagram. **Bottom row:** When the smoothness cost is large, most indices of the Voronoi diagram are just the index of the location itself, and we only store the indices which differ.

to be known only approximately. Rather than store the full matrix as double-precision floating point values, we store the minimum and maximum values in each matrix, scale the cost matrices to be between 0 and $2^8 - 1$, and store them as unsigned 8-bit integers.

Compressing the Voronoi matrices The Vornoi diagrams – which encode where every node should be placed as a function of its parent's offset – need to be computed and stored for each of the nodes in the graph and this can consume a significant amount of memory. In practice we have observed that these diagrams often conform to one of two patterns (Figure 5.4). If the smoothness cost is small, the Voronoi diagram will be dominated by only a few entries corresponding to a few dominant local minima of the cost matrix and all other cost matrix values can be discarded without altering the resulting Voronoi diagram. In this case, we store only these entries of the cost matrix and the Voronoi diagram can then be recomputed when it is needed.

Alternatively, if the smoothness constraint is relatively high then the best motion

of a node will usually be similair to that of its parent. In this case we store a sparse matrix encoding the entries where the child's displacement differs from that of its parent. The full, non-sparse Voronoi diagram can then be constructed when it is needed.

At runtime the system determines which compression scheme is most effective for each node and applies it.

5.3 Multi-frame optical flow

For two-frame optical flow, large displacements can make it difficult or even impossible to find correct correspondences, such as when an object moves out of frame. In Chapter 4, we proposed a simple method to incorporate information from multiple frames without changing the optical flow algorithm itself. This is done by assuming that objects move linearly, from inertia, over a short time span. Multiple "inertial" estimates of the flow are formed from nearby frames under this assumption and subsequently fused using a classifier. However, this requires running the flow computation multiple times, in addition to training a classifier and using it to fuse the estimates.

In our global optimization framework, inertial estimates can be directly incorporated into the data cost term and the optimization does not change at all as shown in Figure 5.5. Let $\mathcal{D}_v^{t+1}(\mathbf{u}(v))$ be the data term for pixel node v with respect to frame $t+1$, which is just the standard two-frame data cost function. Similarly, let $\mathcal{D}_v^{t-1}(-\mathbf{u}(v))$ be the cost as projected onto frame $t-1$ and let $\mathcal{D}_v^{t+2}(2\mathbf{u}(v))$ be the cost with respect to frame $t+2$. If a vertex is occluded, it may have a high cost with respect to frame $t+1$, but might match better to one of the other frames.

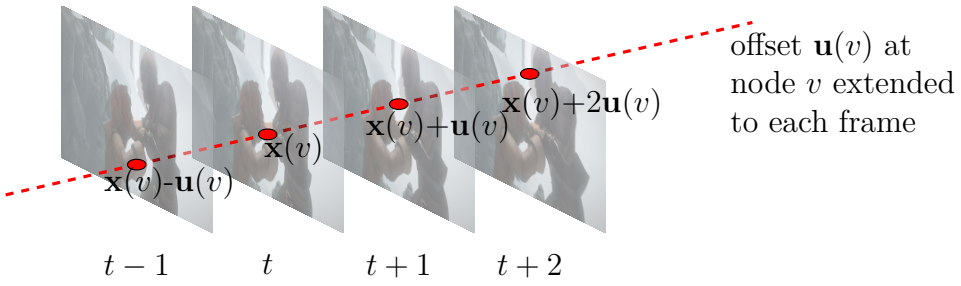


FIGURE 5.5: Depiction of our multi-frame term. Rather than only computing the data cost from frame t to $t+1$, the offset is extended to other adjacent frames, and the data terms are combined into a single cost matrix that is used during optimization. The intuition is that if a node is occluded, it may match better to another adjacent frame, assuming linear motion.

In other words, we want vertex v to match to *one* of the three frames, but not necessarily all. We then define

$$\mathcal{D}_v(\mathbf{u}(v)) = \min\left\{\mathcal{D}_v^{t+1}(\mathbf{u}(v)), \quad \mathcal{D}_v^{t-1}(-\mathbf{u}(v)) + \beta, \quad \mathcal{D}_v^{t+2}(2\mathbf{u}(v)) + \beta\right\}, \quad (5.13)$$

where $\beta > 0$ biases the cost towards matching to frame $t+1$. This data cost is then used directly within the optimization. This approach results in minimal increase in runtime especially when the distance function can be computed efficiently. It would be difficult to use this method within a continuous optimization framework because this approach introduces many extra local minima and results in some points not being differentiable, but neither of these issues is a problem in our discrete framework. We evaluate this approach on the MPI-Sintel dataset in Section 5.4.2, and find that it significantly decreases the error over two-frame optical flow, especially for unmatched pixels.

5.4 Experiments

In the sequel our algorithm is denoted by HCOF, while the HCOF+multi variant uses multiple frames as described in Section 5.3.

Parameters Parameters were determined for the MPI-Sintel and Middlebury datasets using a small-scale grid search on each training dataset. For MPI-Sintel, we set $\alpha = 0.15$, $\lambda_0 = 1$, $\lambda_1 = 2$, $\eta_0 = 50$, $\eta_1 = -0.05$, $\tau = 0.01$, and $\beta = 40$ when multiple frames were used. For Middlebury, we set $\alpha = 0.05$, $\lambda_0 = 0$, $\lambda_1 = 0.2$, $\eta_0 = 50$, $\eta_1 = -0.05$, $\tau = 0.01$, and $\beta = 0$. For the Synthetic and Tracking datasets evaluated in Sections 5.4.4 and 5.4.5, the parameters from the MPI-Sintel dataset were used.

Ten pixels were sampled per superpixel to compute the cost matrices on the upward pass. When computing the cost matrices for each individual pixel on the downward pass we used a window around the parent's offset that had a width of the greater of 5 and 20% of the magnitude of the parent's offset. The maximum offset was set to 200 pixels for all datasets except for Middlebury where it was set to 40 pixels. Note that a maximum offset of 200 pixels accounts accounts for 99.9% of all pixels in MPI-Sintel and results in a label space with 160,000 labels. On the upward pass, the data cost matrices were sampled every 3 pixels. The SIFT features had a cell size of 5 pixels and the Lab color features had a cell size of 3 pixels. For SLIC superpixels, the region size parameter was set to 50 and the regularization parameter was also set to 50.

	Army	Meq.	Scheff.	Wood.	Grove	Urban	Yos.	Teddy	mean
Layers++ (Sun et al., 2010b)	0.08	0.19	0.20	0.13	0.48	0.47	0.15	0.46	0.27
MDP-Flow2 (Xu et al., 2012)	0.09	0.19	0.24	0.16	0.74	0.46	0.12	0.78	0.35
DeepFlow (Weinzaepfel et al., 2013)	0.12	0.28	0.44	0.26	0.81	0.38	0.11	0.93	0.42
LDOF (Brox and Malik, 2011)	0.12	0.32	0.43	0.45	1.01	0.10	0.12	0.94	0.56
HCOF	0.36	1.74	0.92	1.07	1.09	1.39	0.71	1.16	1.05

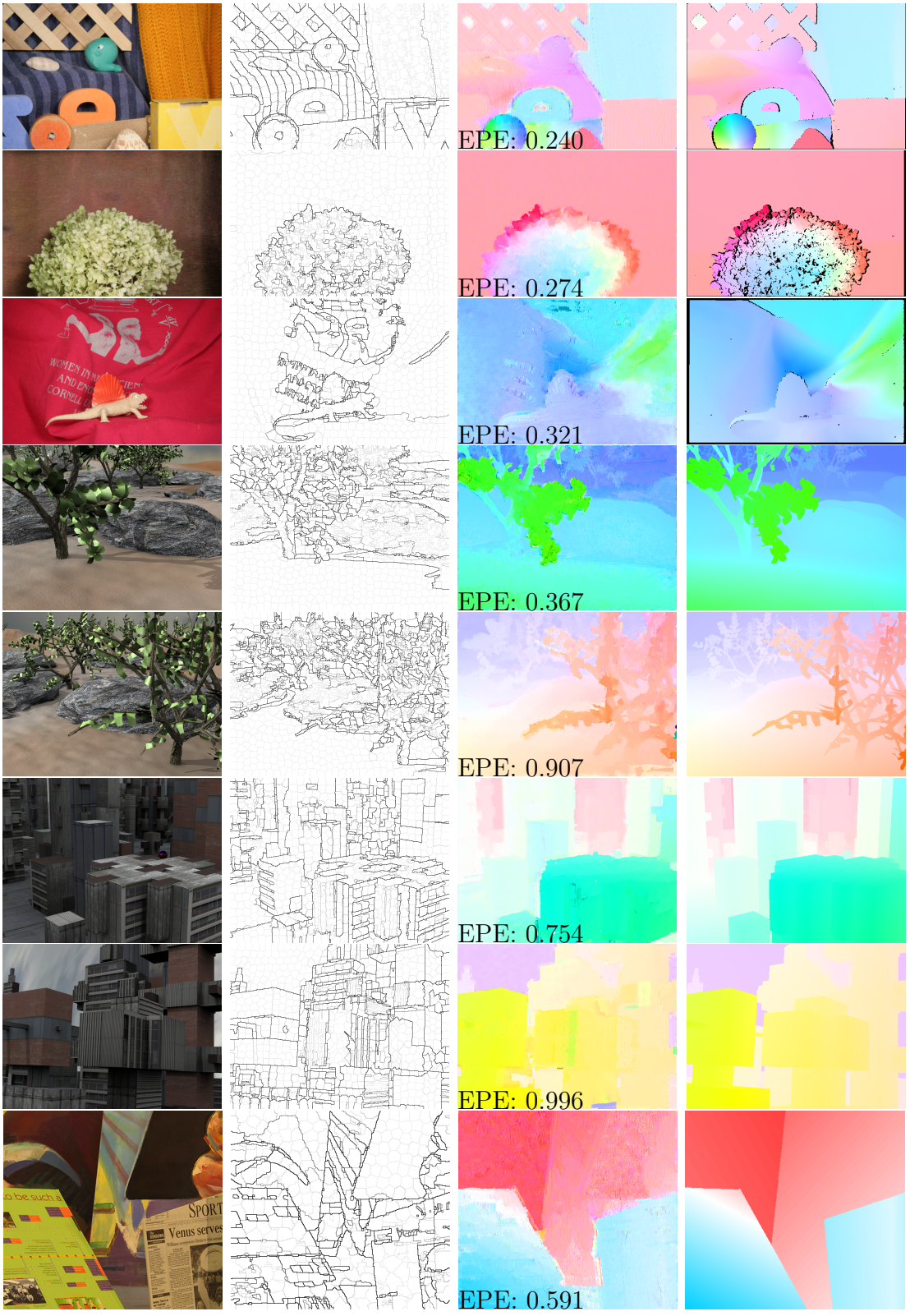
TABLE 5.1: Endpoint error on the Middlebury test dataset.

5.4.1 Middlebury

The Middlebury dataset is a commonly-used optical flow dataset (Baker et al., 2011). The motions are quite simple and there are minimal occlusions. Results on the training dataset are shown in Figure 5.6. The results are fairly accurate, and most of the large errors can be traced to an incorrect segmentation. We note that this discrete, global-optimization algorithm is not ideal for the Middlebury dataset: the motions on Middlebury are small enough that local optimization performs very well. Results on the test dataset are given in Table 5.1. As expected, the results are also somewhat worse than algorithms that use continuous optimization methods due to the simplicity of this dataset.

5.4.2 MPI-Sintel

The MPI-Sintel dataset (Butler et al., 2012) is a difficult optical flow dataset created using an open-source 3D computer-generated film. The dataset consists of over 1000 images and contains difficulties such as large displacements, significant occlusions, lighting variation, and motion blur.



(A) First image (B) Segmentation (C) Estimated flow (D) Groundtruth flow

FIGURE 5.6: Results on the Middlebury training set. Colors for the estimated flow values are scaled based on the maximum offset of the groundtruth flow.

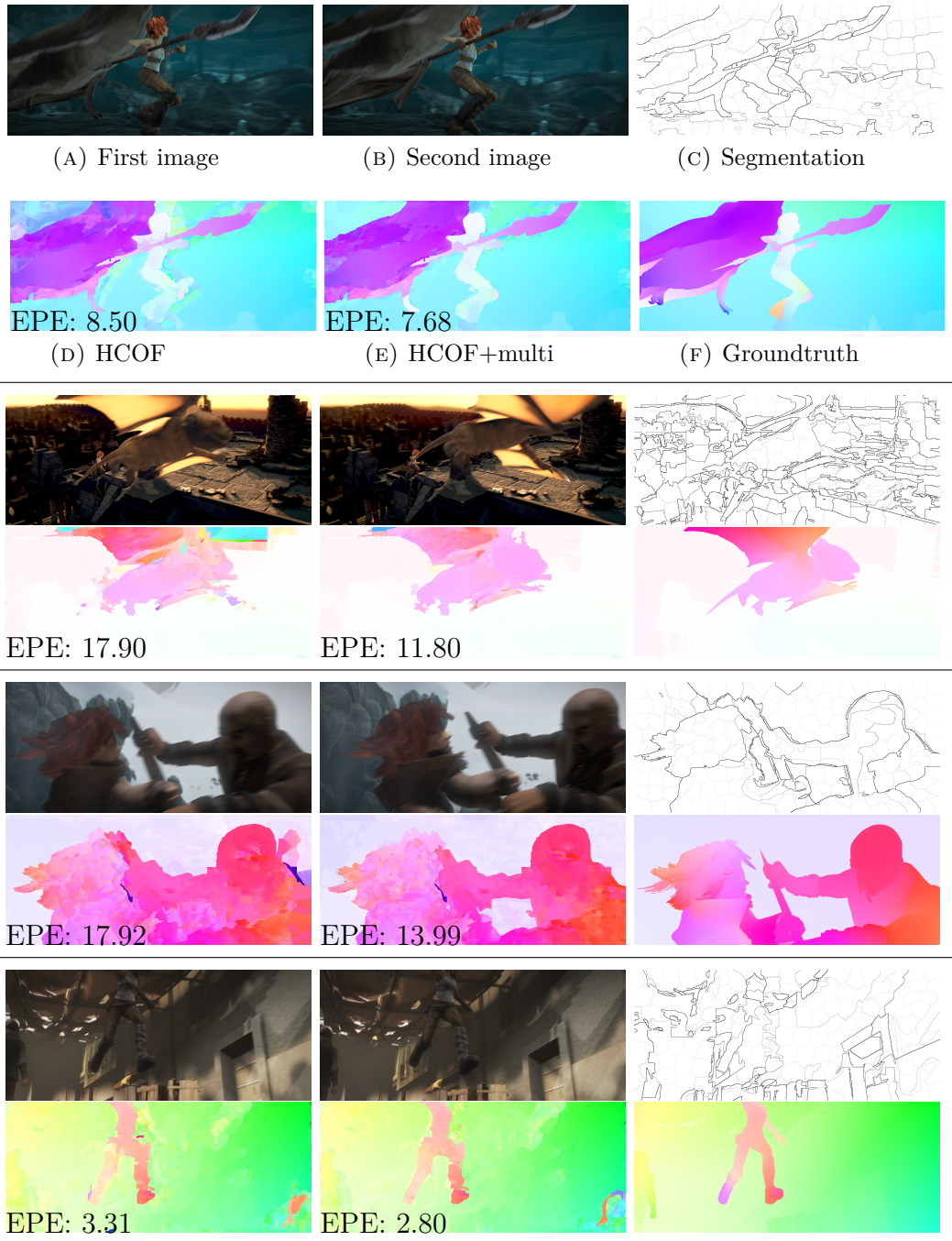
	EPE	s0-10	s10-40	s40+
HCOF	5.882	3.163	6.786	17.596
HCOF+multi	5.265	2.800	5.928	15.892

TABLE 5.2: Evaluation on the Final training dataset of MPI-Sintel. The use if inertial estimates improves results.

Results for several images chosen from the Final training set of MPI-Sintel are show in Figure 5.7. The global nature of the optimization coupled with the underlying superpixel segmentation results in flow estimates with crisp motion boundaries as opposed to the oversmoothed motions produced by many schemes based on continuous optimization.

We have found that the use of multiple inertial estimates of optical flow in the data cost can significantly improve results, especially in occluded regions. This is illustrated quantitatively in Table 5.2 which summarizes the results obtained with both HCOF and HCOF+multi on the Final training dataset.

Quantitative results on the test set for the method HCOF+multi are shown in Table 5.3. Our method is competitive with many modern methods, although the errors are still below the state-of-the-art. A significant fraction of the error can be attributed to segmentation errors, as we show in Section 5.4.3. When we evaluated the method on the training dataset using a segmentation derived from the ground truth motion, the EPE error results improved by 15% on the training set. This suggests that future work could consider strategies that refine the segmentation tree as the optimization proceeds.



(A) First image (B) Second image (C) Segmentation

(D) HCOF (E) HCOF+multi (F) Groundtruth

FIGURE 5.7: Results on several images from the Final dataset of MPI-Sintel. The flow images are colored with respect to the maximum groundtruth displacement.

5.4.3 Effect of segmentation error

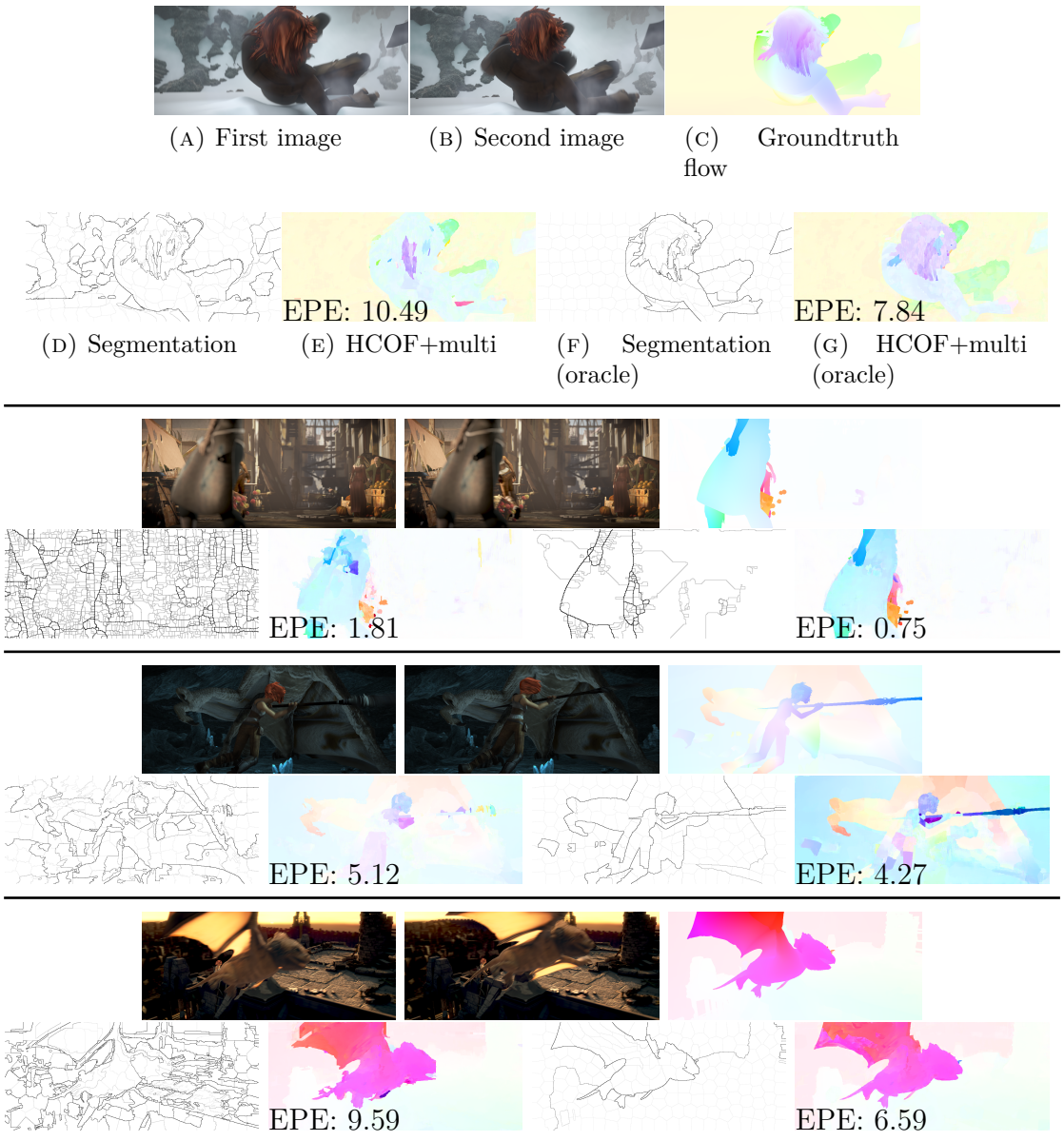
Because our graphical model is set up based on a hierarchical image segmentation, it is sensitive to segmentation errors. In order to get a sense of how this affects

	EPE	s0-10	s10-40	s40+
EpicFlow (Revaud et al., 2014)	6.285	1.135	3.727	38.021
TF+OFM (Kennedy and Taylor, 2015)	6.727	1.512	3.765	39.761
DeepFlow (Weinzaepfel et al., 2013)	7.212	1.284	4.107	44.118
AggregFlow (Fortun et al., 2014)	7.329	1.241	4.296	44.858
ChannelFlow (Sevilla-Lara et al., 2014)	8.835	1.292	5.349	54.648
LDOF (Brox and Malik, 2011)	9.116	1.485	4.839	57.296
AnisoHuber.L1 (Werlberger et al., 2009)	11.927	1.155	7.966	74.796
HCOF+multi	8.799	1.682	5.786	51.363

TABLE 5.3: Evaluation on the Final test dataset of MPI-Sintel. We report endpoint error for all pixels, and also based on the groundtruth speed.

the results, we perform the following experiment. For the training set of MPI-Sintel, rather than using the image itself to generate a segmentation, we run the segmentation algorithm on the colorized flow image of the groundtruth optical flow. The resulting segmentation should divide the image into regions based explicitly on the true motion field, rather than on color differences in the original image.

Quantitative results are given in Table 5.4, and several example images are shown in Figure 5.8. By using an "oracle" segmentation of the flow image, we obtain a 15.138% improvement on the training dataset. This indicates that segmentation error is a significant source of error in our approach, and subsequent research might get improved results by focusing on this issue. Observe that if we obtained the same percentage improvement on the test set, the resulting endpoint error would be 7.446, putting us much close to the current state-of-the-art.



(A) First image (B) Second image (C) Groundtruth flow

(D) Segmentation (E) HCOF+multi (F) Segmentation (oracle) (G) HCOF+multi (oracle)

FIGURE 5.8: Example images showing the effect of segmentation error on optical flow results on MPI-Sintel. We show the optical flow results using a segmentation of the image, as well as when segmentation is performed on the colorized groundtruth flow image, denoted as the "oracle". The improved segmentation often results in improved flow estimates.

	EPE	s0-10	s10-40	s40+
HCOF+multi	5.265	2.800	5.928	15.892
HCOF+multi (oracle)	4.468	2.225	5.076	13.609
% improvement	15.138%	20.536%	14.372%	14.366%

TABLE 5.4: Effect of segmentation error on MPI-Sintel. Rather than segmenting the image, we apply the segmentation algorithm to the colorized groundtruth flow image on the training dataset. This results in an improved segmentation. Otherwise, all parameters are kept constant.

5.4.4 Synthetic Dataset

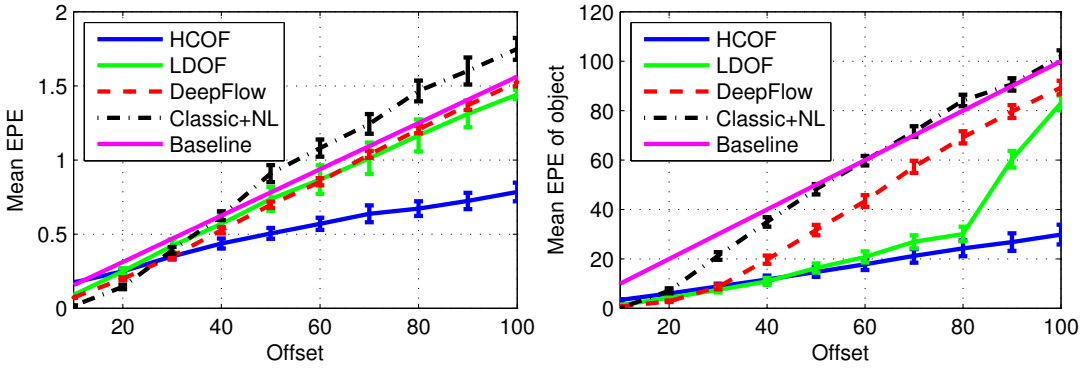
In order to illustrate the issues that often occur in the face of large motions, we constructed a simple synthetic dataset using the imagery from the Final dataset of MPI-Sintel. An example of an image pair from this dataset is shown in Figure 5.9. For each such pair a background is generated by choosing an image uniformly at random and selecting a random 256×256 square patch from that image. We then generate an "object" by selecting a random 32×32 patch from another image. The object patch is placed at a random location on the background and is then translated in a random direction by a specified offset distance. For a given offset distance, we evaluate each algorithm averaged over 100 such random images.

We deliberately chose a relatively simple motion stimulus to illuminate how various schemes handle layered motions with large displacements. Adding more layers, more motions and more objects would not fundamentally alter the results.

We compare our own algorithm, HCOF, to several other modern approaches for which code was readily available: Classic+NL (Sun et al., 2010a), LDOF (Brox and Malik, 2011), and DeepFlow (Weinzaepfel et al., 2013). Classic+NL is a modern implementation of a coarse-to-fine variational approach. Large Displacement Optical Flow (LDOF) incorporates sparse feature matching into the optimization

(A) First image (B) Second image (C) Groundtruth

FIGURE 5.9: An example of an image from our synthetic dataset. A random 256×256 patch is used as a background while another 32×32 "object" patch is moved a given offset between images. In this example, the offset is 50 pixels.



(A) EPE for entire image (B) EPE for object region

FIGURE 5.10: Endpoint error on a synthetic dataset where an object is translated a varying amount. The Baseline method is a zero-flow estimate. The endpoint error is averaged over 100 images and we show the standard error as error bars. In (a), endpoint error is averaged over the entire image, while in (b) we plot the error for only the object itself.

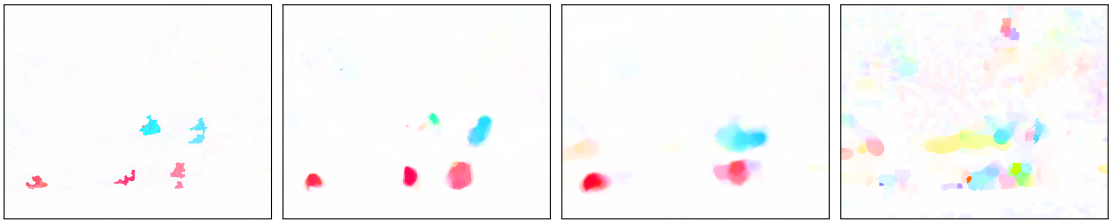
in order to better deal with large displacements. Finally, DeepFlow is a top-performing algorithm on MPI-Sintel that uses a more advanced feature matching term to account for deformation. For all approaches, we used their default parameters, except for DeepFlow where we used the "improved settings" as documented in their code. We also compare to the baseline method of predicting a zero-valued flow.

Results on this dataset are given in Figure 5.10. We show the mean endpoint error for each method as averaged over 100 random images, and the offset is varied from a minimum of 10 pixels to a maximum of 100 pixels. The error is evaluated for both the full image as well as only the object patch itself.

In all cases, the error increases roughly linearly as the offset is increased. For offsets of less than 20 pixels, HCOF is outperformed by the other coarse-to-fine continuous methods which are able to achieve better sub-pixel accuracy and are not affected by segmentation error. However, HCOF is significantly more robust to large displacements. For offsets of 100 pixels, our method achieves errors several times lower than other approaches. The reason for this is that the magnitude of the offset has little effect on the accuracy of our method because it uses a discrete, global optimization.

It is interesting to note that all other methods based on continuous optimization fail for very large displacements on this simple dataset even though they often achieve state-of-the-art results on the more realistic MPI-Sintel dataset (Secion 5.4.2). This is due to the motions and images in MPI-Sintel being more complex: as the offsets get larger, they are often made even more difficult by deformations and rotations of the object, motion blur, and large lighting changes. These effects also cause an image segmentation to have errors which leads to an incorrect hierarchy in our model. Continuous methods are able to handle these deformations better and are not subject to errors in segmentation. However, HCOF might then outperform other methods in situations that involve large translational motions when there is minimal segmentation ambiguity. We explore this further in the next section.

(A) First image (B) Second image (C) Estimated background (D) HCOF



(E) HCOF+multi (F) LDOF (G) DeepFlow (H) Classic+NL

FIGURE 5.11: An example of a result from the Tracking dataset. In (a), the black boxes indicate the people for which annotations are provided and the extent of the boxes is the area over which we look for the lowest endpoint error for each person. In (c), we show the estimated background pixels, for which we assume the flow is zero in our evaluation. The results of different methods are given in (d)-(h). The results of our method HCOF are much more localized.

5.4.5 Tracking Dataset

To demonstrate that the issues illustrated in Section 5.4.4 do in fact occur in real images we evaluated our algorithm on the Hotel sequence from the BIWI Walking Pedestrians dataset (Pellegrini et al., 2009), which is a video of a city sidewalk taken from an overhead camera. The video was taken at 25 frames per second and annotations are provided for every 10th frame indicating the positions of people visible in the scene. We further subsample every other annotated frame so that the dataset contains both large and small motions, resulting in a total of 572 annotated frames each with a resolution of 720×576 pixels. Two frames from this dataset are shown in Figure 5.11.

We evaluate flow methods on this dataset in two ways. First, we have sparse annotations of the locations of people. For these points we calculate the groundtruth offsets and compare it to the results from each method. However, this may penalize methods that successfully track most of a person but miss the one annotated pixel, and so instead we find the best estimated flow value within a 41×41 box around each annotated point, as shown in Figure 5.11. This evaluation is done independently on all 2623 pedestrian annotations in the dataset

We also evaluate methods by noting that the camera is stationary and that much of the image will have zero optical flow. We determine the stationary pixels in each frame by thresholding the magnitude of the intensity difference between frames at 0.05. We evaluate the flow on these background pixels separately.

Qualitative results on a pair of frames from this dataset are shown in Figure 5.11, and quantitative results are given in Table 5.5. For the annotated tracks, the results are presented using average EPE and are also divided into bins based on the magnitude of the groundtruth motion, similar to the results for MPI-Sintel (Section 5.4.2). The last column of the table also shows the mean EPE over all the background pixels.

LDOF performs the best of all continuous methods, but for large offsets we find that HCOF+multi has a significantly lower error than other methods. DeepFlow may underperform on this dataset as compared to MPI-Sintel because the objects that are moving are relatively small and DeepFlow does not compute a fully-dense feature matching due to memory constraints. Overall, HCOF+multi outperforms all other methods on this dataset. We also find that HCOF and HCOF+multi are significantly more accurate on the background pixels that have zero flow than all methods except for Classic+NL which has a low error only because it fails on

	Annotated Tracks				Background
	EPE	s0-10	s10-40	s40+	EPE
LDOF (Brox and Malik, 2011)	3.031	0.654	**1.954**	4.333	0.738
DeepFlow (Weinzaepfel et al., 2013)	13.048	0.862	4.751	20.265	0.730
Classic+NL (Sun et al., 2010a)	35.220	0.821	16.705	54.631	**0.385**
HCOF	2.896	**0.603**	2.321	4.060	0.580
HCOF+multi	**2.561**	0.678	2.948	**3.349**	0.430

TABLE 5.5: Evaluation on the tracking dataset. The left section of the table shows the endpoint error over all annotated tracks, both overall and divided based on the magnitude of the groundtruth motion vectors. The last column measures the endpoint error over estimated background pixels.

correctly estimating the motion of the people. All other methods tend to pull part of the background along with the objects and are less able to localize the objects themselves. This can be seen visually in the results in Figure 5.11. Note, for example, that all methods other than HCOF are unable to separate the two people walking next to each other.

5.4.6 Computational Requirements

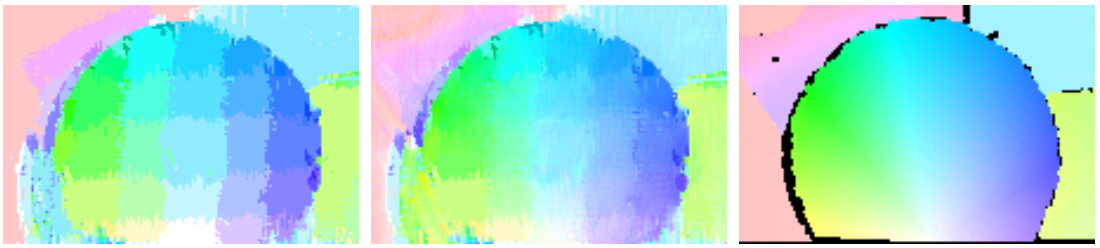
There are two parts of our algorithm that require the most computation time: the distance transforms that are computed at each node in the hierarchy, and the initial feature computation at the leaves of the tree. The computational complexity is thus dependent on the size of the image and of the image hierarchy – which determine the number of variables in the MRF and the associated number of distance transforms to be computed – and on the type of features used. We evaluate these factors on the Ambush5 image sequence from the MPI-Sintel dataset, which consists of 49 image pairs of size 1024×436. We evaluate the time for computing optical flow on a desktop with a 2.0 GHz Intel Xeon processor and 4 GB of memory. We evaluate with and without using multi-frame inertial estimates.

The mean time taken to evaluate two-frame optical flow without the use of inertial estimates was 15.3 minutes. When inertial estimates were used, the mean time was 40.9 minutes. The large increase in computation time occurs because the features we use are quite complex, including both Lab and SIFT features. The computation of these image features takes a relatively large percentage of the computation time, and by using inertial estimates the feature computation is tripled. Still, fewer distance transforms need to be done and no classification is needed to fuse the inertial estimates as would be the case if they were computed separately.

We also note that much of this algorithm could be sped up through parallelization. The computation of the data cost can be done independently for each pixel. Also, the computation performed at each tree node can be done independently for each node in one layer of the tree on both the upwards and downwards pass of the algorithm.

5.4.7 Effect of Sub-Pixel Localization

Although our MRF is discrete by nature, in Section 5.2.1 we described a method for obtaining sub-pixel localization at the last step of backtracing by fitting a quadratic function separately for the x and y offsets. The effect of this is shown in Figure 5.12. Without sub-pixel localization, the results are very blocky and a grid-like pattern is clearly visible. Although our sub-pixel localization does not completely eliminate this pattern, the results are nonetheless significantly smoother while there is negligible additional computation time.



(A) Without sub-pixel localization

(B) With sub-pixel localization

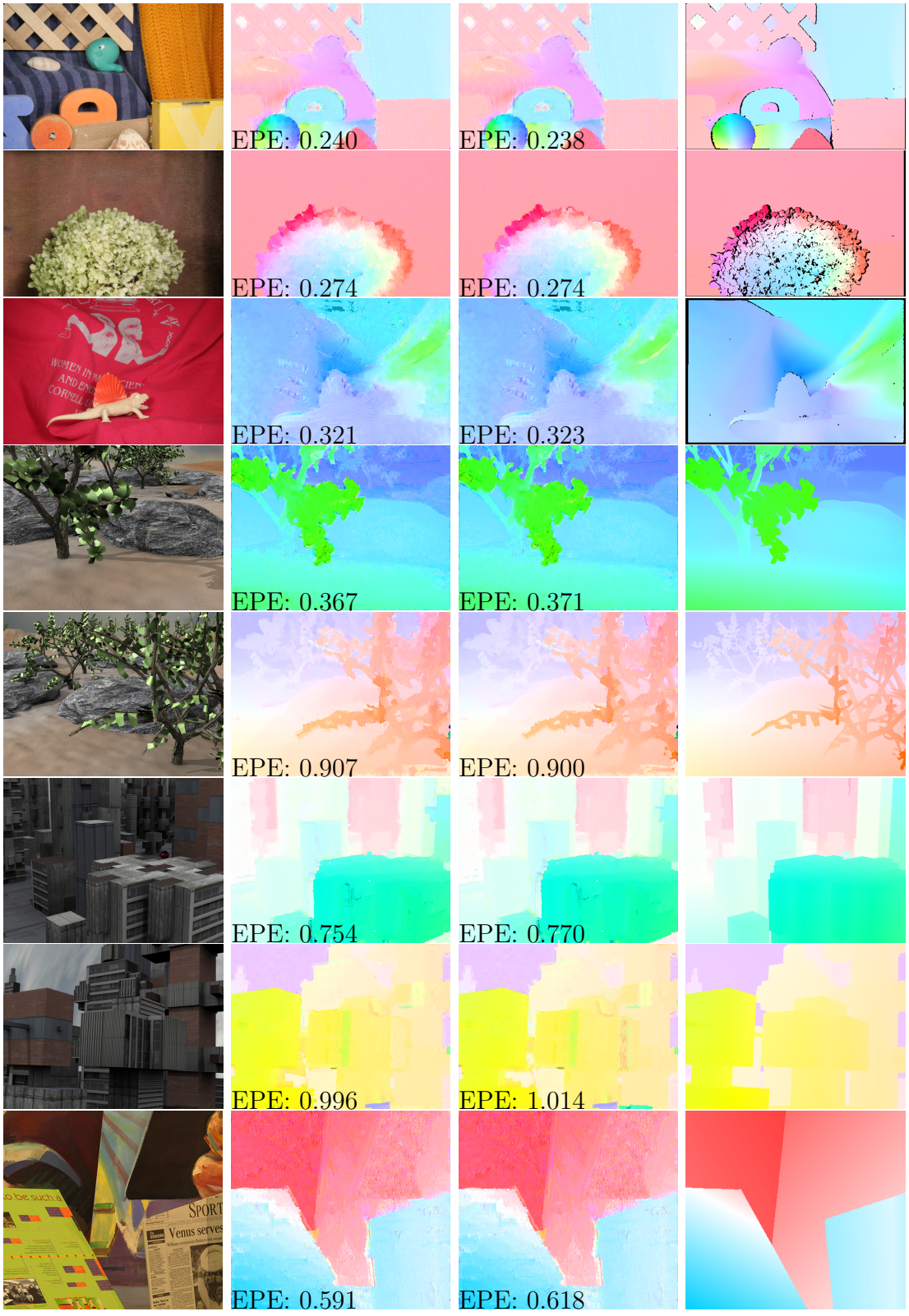
(C) Groundtruth flow

FIGURE 5.12: Effect of sub-pixel localization on a portion of the RubberWhale image from the Middlebury dataset.

5.4.8 Effect of approximations

Although our tree-structured MRF can be solved optimally in polynomial time, in Section 5.2.1 we outlined several approximations used in our actual implementation in order to speed up the computation and lessen the memory requirements. Still, it is possible to compute an exact solution at the cost of additional computation. We examine the effect of our approximations by evaluating our algorithm on the Middlebury dataset without such approximations. In particular, we do not subsample the pixels when computing data cost matrices on the upward pass of the optimization and compute a full data cost matrix for each pixel on the downward pass when labeling each pixel.

A comparison between the exact solutions and the solutions obtained when using our proposed approximations is given in Figure 5.13 (in both cases, sub-pixel localization is also used). The approximate and exact solutions to the MRF are virtually indistinguishable and have nearly identical error; the mean EPE for the approximate and exact solutions is 0.556 and 0.564, respectively. However, the computation time is very different: while the approximate solution is computed in about 5 minutes the exact solution takes nearly 45 minutes.



(A) First image (B) Approx. solution (C) Exact solution (D) Groundtruth flow

FIGURE 5.13: Results on the Middlebury training set. Colors for the estimated flow values are scaled based on the maximum offset of the groundtruth flow.

5.5 Limitations

Although our method finds the (near) global minimum of our optimization problem, the results are still limited by the following:

Occlusions Our model does not explicitly handle occlusions, although errors are somewhat reduced by using a robust cost function (Black and Anandan, 1996). However, when an entire region which is not well-connected to any other region is occluded, it may match to a location very far from its original location, resulting in very high endpoint error. This can be somewhat mitigated, however, by using the inertial estimate from nearby frames.

Segmentation errors The smoothness terms in our model depends on an accurate segmentation. If the segmentation is inaccurate, regions which should be grouped together may be separated or regions with different motions may be linked. Even when using state-of-the-art segmentation methods, we still found that this was a source of error in our results.

Smoothness model Our hierarchical smoothness model allows for deformation, but does not explicitly model large changes such as rotations and scale changes. It also assumes a fronto-parallel motions, and would penalize other affine motions.

5.6 Summary

In this chapter, we have shown that image correspondence problems with very large label spaces can be solved not only efficiently, but optimally, by using a hierarchical segmentation. Our results are already competitive with many modern

approaches, and we expect that the use of more informative features (Byrne and Shi, 2013; Weinzaepfel et al., 2013), as well as prior and contextual information, will lead to even better algorithms.

The proposed method is sensitive to segmentation quality. It may be possible to reduce this error by re-weighting the connections between nodes, or even by dynamically generating the segmentation as the optimization proceeds. Our model is also biased towards fronto-parallel motions and will have difficulty on datasets that have excessive rotations, scale changes, or affine motions. We plan on exploring many of these issues in future work.

Finally, we point out that this framework could be used for other problems, including semantic segmentation. In this case, rather than using distance transforms, a cost matrix would be kept at each vertex that records the cost of each label for each choice of its parent's label. While similar approaches have been used before for this task (Feng et al., 2002; Kumar and Hebert, 2005), we have shown that this can be done efficiently for very large label spaces, which opens the door to large-scale semantic segmentation involving thousands of object categories.

6

Conclusion

Decades of research into optical flow and motion estimation have resulted in a huge number of algorithms that use a variety of different models and computational approaches. These algorithms may have different strengths and weaknesses, but there is no doubt that the state-of-the-art in optical flow has steadily improved over time. This progress has been quantified by a number of datasets, including Middlebury, KITTI and now MPI-Sintel. While the simple motions present in the Middlebury dataset have been largely dealt with and efficient algorithms can easily find very accurate solutions, there is still a significant amount of error in the complex MPI-Sintel dataset. These errors are caused by the much more realistic and complex motions present in the dataset, which include large occlusions, complex motions, lighting changes, and atmospheric effects.

In this thesis we have presented two quite disparate algorithms: one algorithm performs continuous optimization using a triangular decomposition of the image while the other performs discrete optimization based on a hierarchical segmentation. We have also presented a method of incorporating temporal information into the optimization. Together, these approaches have resulted in improvements on difficult optical flow problems. In particular, the triangulation-based method

allows for natural occlusion estimation, the fusion of multiple frames adds additional temporal information to improve results, and the hierarchical approach uses discrete optimization avoids getting stuck in local optima.

Although the approaches presented in this thesis have resulted in improved motion estimates for many problems, there remain many additional avenues of research. As optical flow datasets have become more difficult with large motions and occlusions, algorithms have necessarily shifted from a gradient-descent "flow" calculation to more of a global matching since objects may have a different appearance and move to a different part of the image. In some cases, this may make the problem difficult enough that higher-level information becomes necessary in the flow calculation. For example, if we recognize that a person is in an image and can identify their pose, then the motion of the corresponding pixels should be constrained to make sense in terms of the kinematics of the human body. The use of such information may allow us to overcome significant occlusions and changes in visual appearance that will cause matching to fail.

Even without the use of high-level information about objects, many improvements can also be made. For example, our hierarchical discrete approach has significant errors on images where the segmentation fails, as often happens on the MPI-Sintel datset. Reducing the segmentation error here would potentially result in much better motion estimates. This approach is also the first globally-optimal discrete optimization to be used for large-displacement optical flow, and adding additional terms to the model to improve its accuracy would be useful. In general, there is still room for the development of optical flow algorithms that are more efficient and better model the motions that are present in real images.

Appendix A

Additional examples of multi-frame fusion

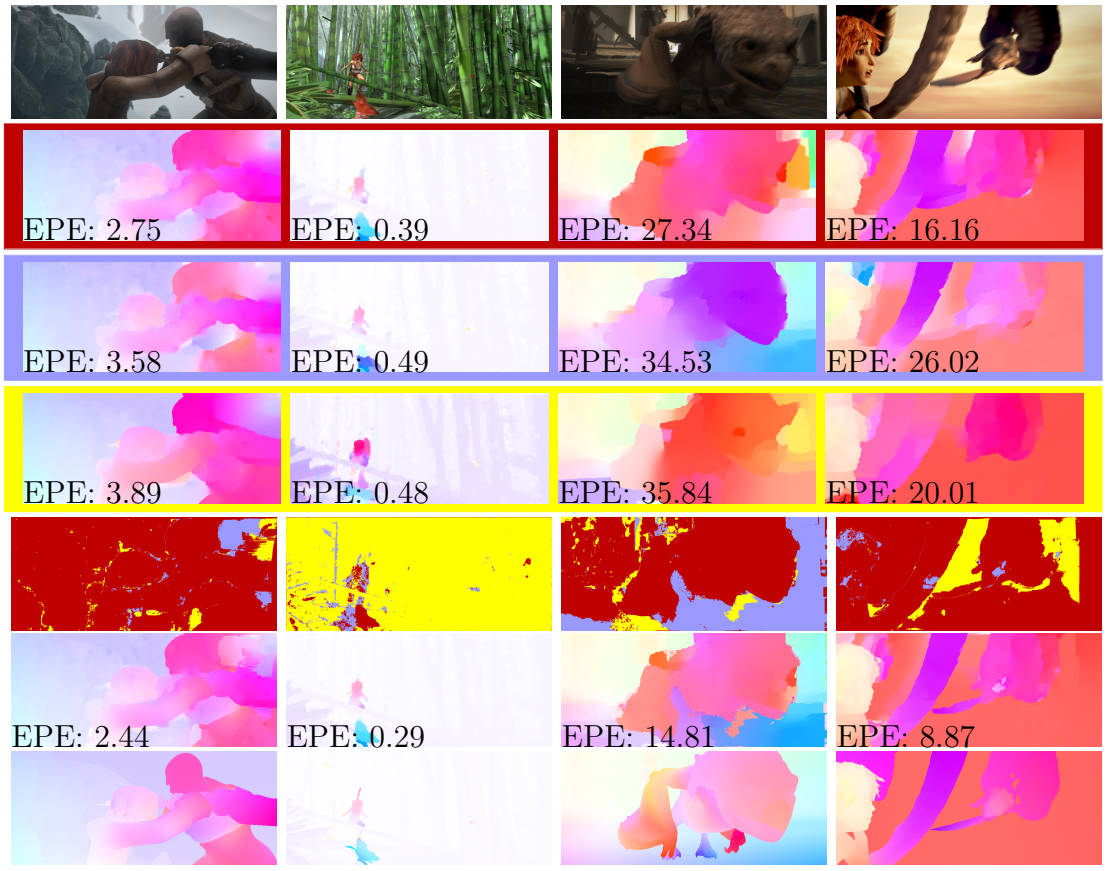


FIGURE A.1: Examples of our multi-frame fusion on the MPI-Sintel Final training set. **Top row:** Frame at time t. **Rows 2-4:** Inertial estimates of the flow $[t \to t+1]$, $-[t \to t-1]$, and $\frac{1}{2}[t \to t+2]$. **Row 5:** Fusion classification for each pixel. Color indicates the estimate used at each pixel. Colors correspond to the border colors of the inertial estimates. **Row 6:** Fused flow estimate. **Bottom row:** Groundtruth flow. For all flow estimates, the endpoint error is printed in the image.

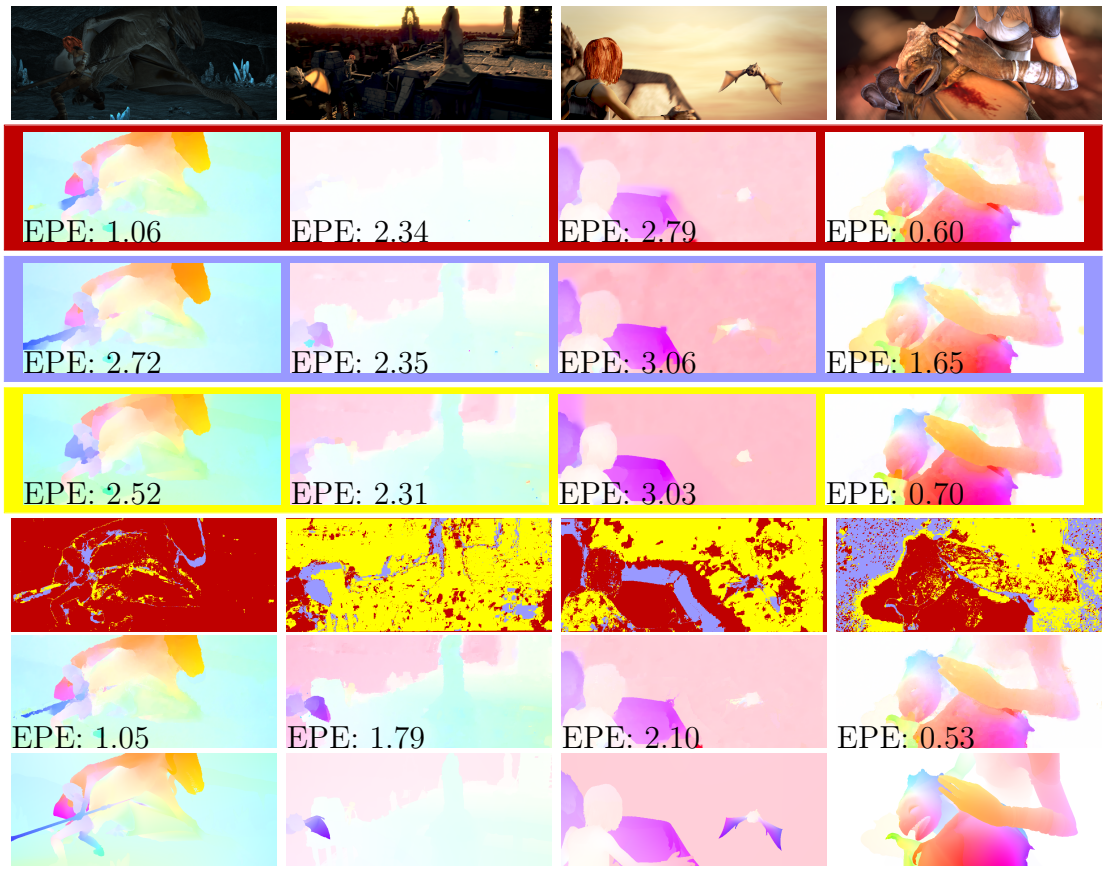


FIGURE A.2: Examples of our multi-frame fusion on the MPI-Sintel Final training set. **Top row:** Frame at time t. **Rows 2-4:** Inertial estimates of the flow $[t \to t+1]$, $-[t \to t-1]$, and $\frac{1}{2}[t \to t+2]$. **Row 5:** Fusion classification for each pixel. Color indicates the estimate used at each pixel. Colors correspond to the border colors of the inertial estimates. **Row 6:** Fused flow estimate. **Bottom row:** Groundtruth flow. For all flow estimates, the endpoint error is printed in the image.

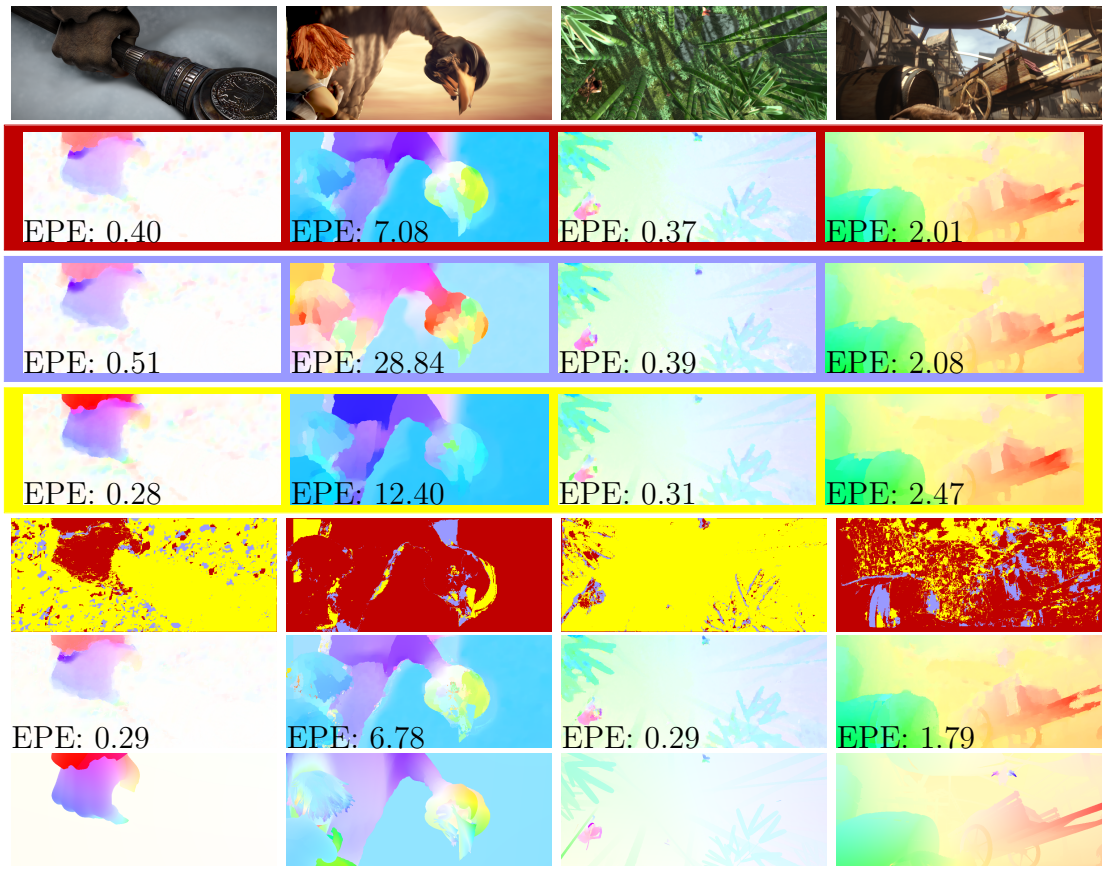


FIGURE A.3: Examples of our multi-frame fusion on the MPI-Sintel Final training set. **Top row:** Frame at time t. **Rows 2-4:** Inertial estimates of the flow $[t \to t+1]$, $-[t \to t-1]$, and $\frac{1}{2}[t \to t+2]$. **Row 5:** Fusion classification for each pixel. Color indicates the estimate used at each pixel. Colors correspond to the border colors of the inertial estimates. **Row 6:** Fused flow estimate. **Bottom row:** Groundtruth flow. For all flow estimates, the endpoint error is printed in the image.

FIGURE A.4: Examples of our multi-frame fusion on the MPI-Sintel Final training set. **Top row:** Frame at time t. **Rows 2-4:** Inertial estimates of the flow $[t \to t+1]$, $-[t \to t-1]$, and $\frac{1}{2}[t \to t+2]$. **Row 5:** Fusion classification for each pixel. Color indicates the estimate used at each pixel. Colors correspond to the border colors of the inertial estimates. **Row 6:** Fused flow estimate. **Bottom row:** Groundtruth flow. For all flow estimates, the endpoint error is printed in the image.

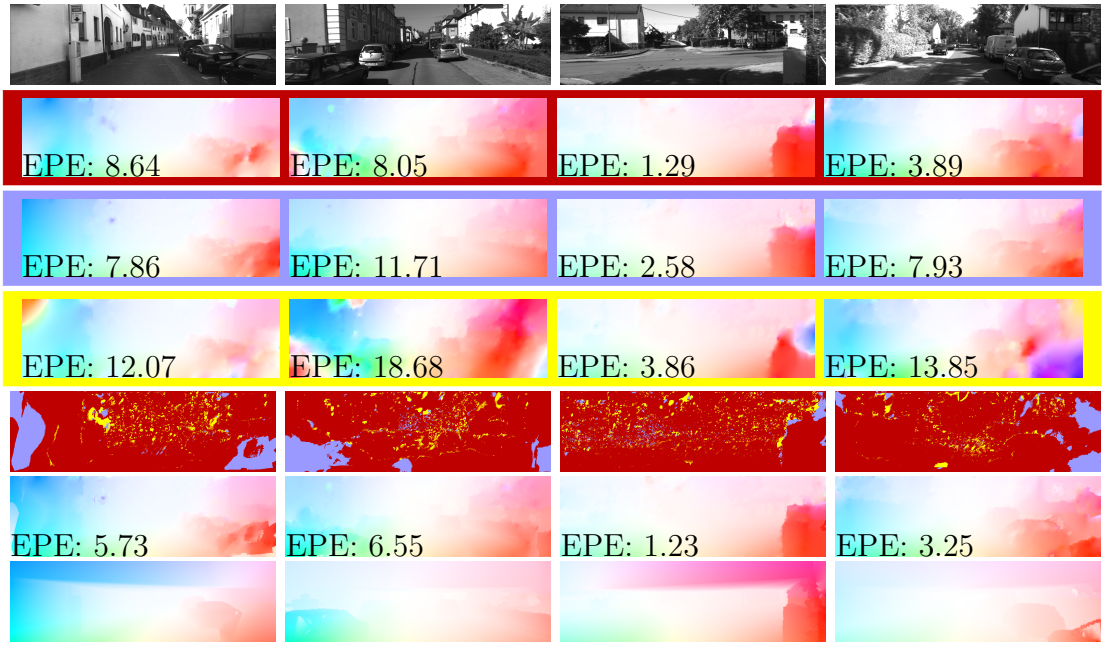


FIGURE A.5: Examples of our multi-frame fusion on the KITTI training dataset. **Top row:** Frame at time t. **Rows 2-4:** Inertial estimates of the flow $[t \to t+1]$, $-[t \to t-1]$, and $\frac{1}{2}[t \to t+2]$. **Row 5:** Fusion classification for each pixel. Color indicates the estimate used at each pixel. Colors correspond to the border colors of the inertial estimates. **Row 6:** Fused flow estimate. **Bottom row:** Groundtruth flow, interpolated using linear interpolation. For all flow estimates, the endpoint error is printed in the image.

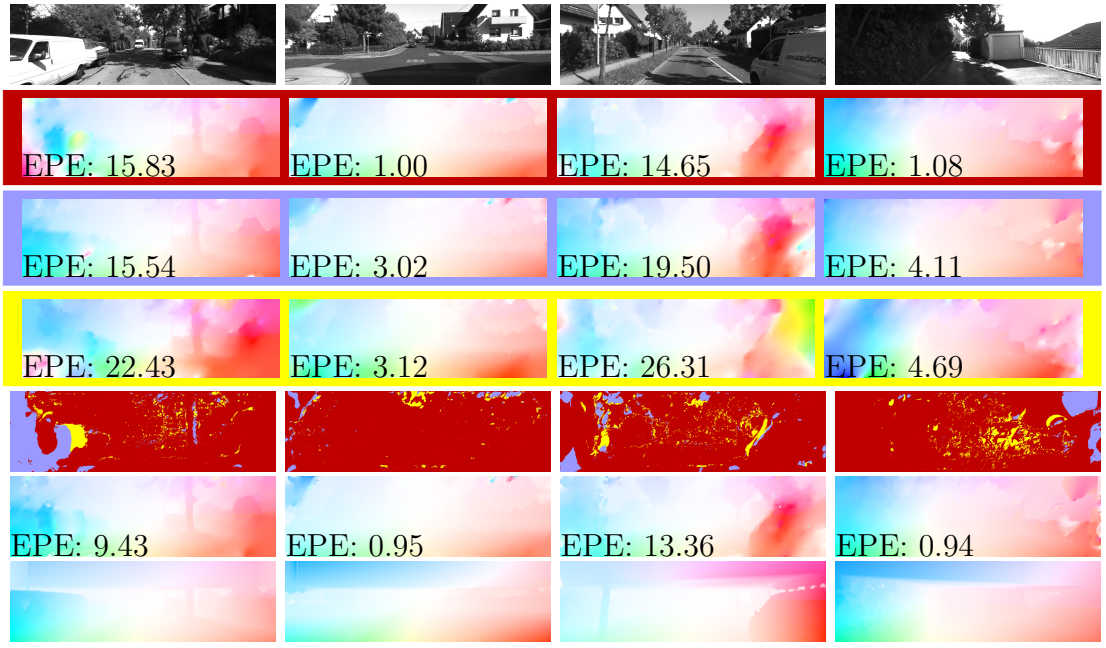


FIGURE A.6: Examples of our multi-frame fusion on the KITTI training dataset. **Top row:** Frame at time t. **Rows 2-4:** Inertial estimates of the flow $[t \to t+1]$, $-[t \to t-1]$, and $\frac{1}{2}[t \to t+2]$. **Row 5:** Fusion classification for each pixel. Color indicates the estimate used at each pixel. Colors correspond to the border colors of the inertial estimates. **Row 6:** Fused flow estimate. **Bottom row:** Groundtruth flow, interpolated using linear interpolation. For all flow estimates, the endpoint error is printed in the image.

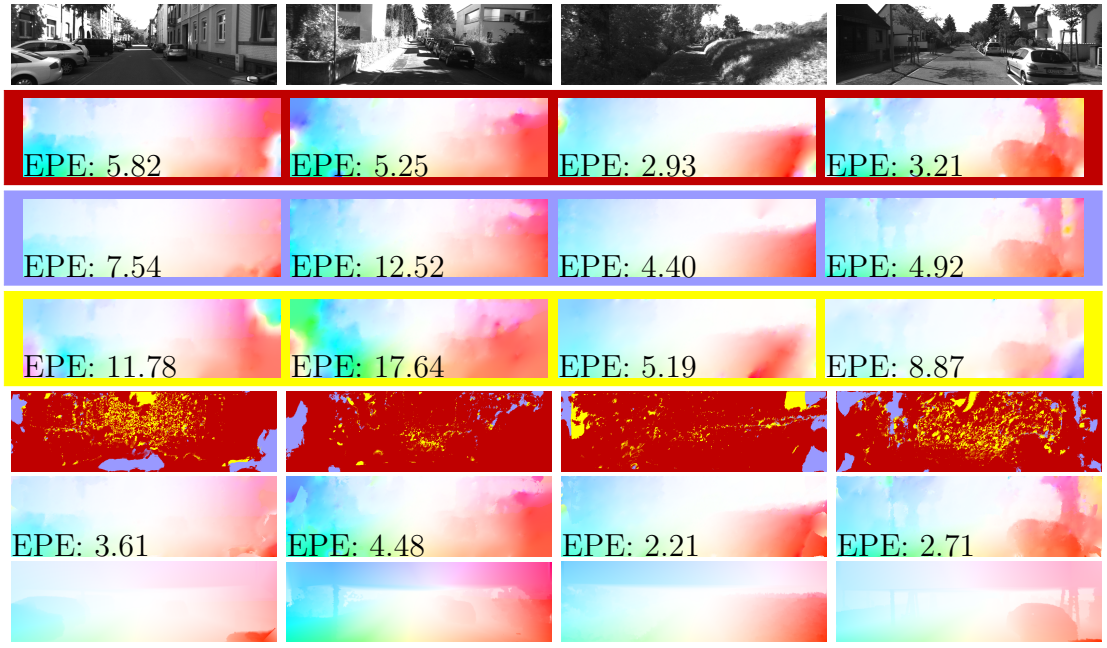


FIGURE A.7: Examples of our multi-frame fusion on the KITTI training dataset. **Top row:** Frame at time t. **Rows 2-4:** Inertial estimates of the flow $[t \to t+1]$, $-[t \to t-1]$, and $\frac{1}{2}[t \to t+2]$. **Row 5:** Fusion classification for each pixel. Color indicates the estimate used at each pixel. Colors correspond to the border colors of the inertial estimates. **Row 6:** Fused flow estimate. **Bottom row:** Groundtruth flow, interpolated using linear interpolation. For all flow estimates, the endpoint error is printed in the image.

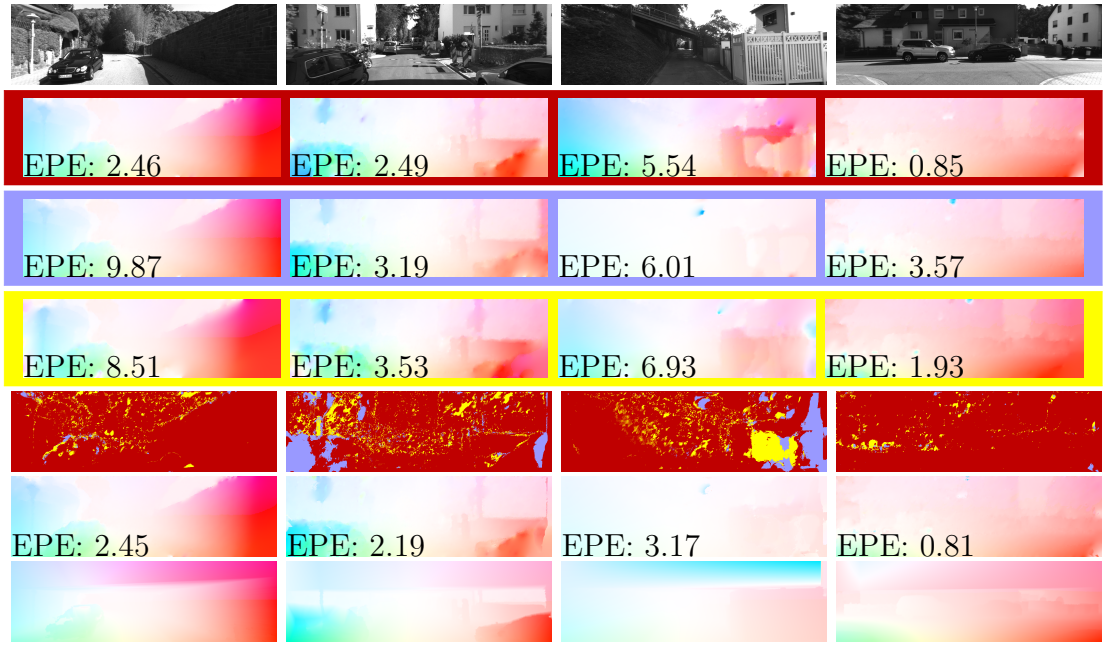


FIGURE A.8: Examples of our multi-frame fusion on the KITTI training dataset. **Top row:** Frame at time t. **Rows 2-4:** Inertial estimates of the flow $[t \to t+1]$, $-[t \to t-1]$, and $\frac{1}{2}[t \to t+2]$. **Row 5:** Fusion classification for each pixel. Color indicates the estimate used at each pixel. Colors correspond to the border colors of the inertial estimates. **Row 6:** Fused flow estimate. **Bottom row:** Groundtruth flow, interpolated using linear interpolation. For all flow estimates, the endpoint error is printed in the image.

Bibliography

Achanta, R., Shaji, A., Smith, K., Lucchi, A., Fua, P., and Susstrunk, S. (2012). SLIC superpixels compared to state-of-the-art superpixel methods. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34(11):2274–2282.

Alvarez, L., Deriche, R., Papadopoulo, T., and Sánchez, J. (2002). Symmetrical dense optical flow estimation with occlusions detection. In *Computer Vision—ECCV 2002*, pages 721–735. Springer.

Alvarez Leon, L. M., Esclarín Monreal, J., Lefébure, M., and Sánchez Pérez, J. (1999). A PDE model for computing the optical flow. *Congreso de Ecuaciones Diferenciales y Aplicaciones*.

Amestoy, P. R., Davis, T. A., and Duff, I. S. (2004). Algorithm 837: AMD, an approximate minimum degree ordering algorithm. *ACM Trans. Math. Softw.*, 30(3):381–388.

Anandan, P. (1989). A computational framework and an algorithm for the measurement of visual motion. *International Journal of Computer Vision*, 2(3):283–310.

Arbelaez, P. (2006). Boundary extraction in natural images using ultrametric contour maps. In *CVPR*, pages 182–182. IEEE.

Arbelaez, P., Maire, M., Fowlkes, C., and Malik, J. (2011). Contour detection and hierarchical image segmentation. *PAMI*, 33(5):898–916.

Ayvaci, A., Raptis, M., and Soatto, S. (2012). Sparse occlusion detection with optical flow. *International journal of computer vision*, 97(3):322–338.

Baker, S. and Matthews, I. (2004). Lucas-kanade 20 years on: A unifying framework. *IJCV*, 56(3):221–255.

Baker, S., Scharstein, D., Lewis, J., Roth, S., Black, M. J., and Szeliski, R. (2011). A database and evaluation methodology for optical flow. *IJCV*, 92(1):1–31.

Bao, L., Yang, Q., and Jin, H. (2014). Fast edge-preserving PatchMatch for large displacement optical flow. *CVPR*.

Barnes, C., Shechtman, E., Finkelstein, A., and Goldman, D. (2009). PatchMatch: a randomized correspondence algorithm for structural image editing. *ACM Transactions on Graphics*, 28(3):24.

Barnes, C., Shechtman, E., Goldman, D. B., and Finkelstein, A. (2010). The generalized patchmatch correspondence algorithm. In *Computer Vision–ECCV 2010*, pages 29–43. Springer.

Bhattacharya, S. (2013). Recognition of complex events in open-source web-scale videos: a bottom up approach. In *Proceedings of the 21st ACM international conference on Multimedia*, pages 1035–1038. ACM.

Black, M. J. and Anandan, P. (1990). A model for the detection of motion over time. In *Computer Vision, 1990. Proceedings, Third International Conference on*, pages 33–37. IEEE.

Black, M. J. and Anandan, P. (1996). The robust estimation of multiple motions: Parametric and piecewise-smooth flow fields. *CVIU*, 63(1):75–104.

Boykov, Y., Veksler, O., and Zabih, R. (2001). Fast approximate energy minimization via graph cuts. *PAMI*, 23(11):1222–1239.

Brox, T., Bruhn, A., Papenberg, N., and Weickert, J. (2004). High accuracy optical flow estimation based on a theory for warping. *ECCV*.

Brox, T. and Malik, J. (2011). Large displacement optical flow: descriptor matching in variational motion estimation. *PAMI*, 33(3):500–513.

Burt, P. J. and Adelson, E. H. (1983). The laplacian pyramid as a compact image code. *Communications, IEEE Transactions on*, 31(4):532–540.

Butler, D. J., Wulff, J., Stanley, G. B., and Black, M. J. (2012). A naturalistic open source movie for optical flow evaluation. In *ECCV*, pages 611–625. Springer.

Byrne, J. and Shi, J. (2013). Nested shape descriptors. In *ICCV*, pages 1201–1208. IEEE.

Chen, Y., Davis, T. A., Hager, W. W., and Rajamanickam, S. (2008). Algorithm 887: CHOLMOD, supernodal sparse cholesky factorization and update/downdate. *TOMS*, 35(3):22.

Chen, Z., Jin, H., Lin, Z., Cohen, S., and Wu, Y. (2013). Large displacement optical flow from nearest neighbor fields. In *CVPR*, pages 2443–2450. IEEE.

Chung, F. R. and Mumford, D. (1994). Chordal completions of planar graphs. *Journal of Combinatorial Theory, Series B*, 62(1):96–106.

Cohen, I. (1993). Nonlinear variational method for optical flow computation. In *PROCEEDINGS OF THE SCANDINAVIAN CONFERENCE ON IMAGE ANALYSIS*, volume 1, pages 523–523. PROCEEDINGS PUBLISHED BY VARIOUS PUBLISHERS.

Cowper, G. (1973). Gaussian quadrature formulas for triangles. *International Journal for Numerical Methods in Engineering*, 7(3):405–408.

Cremers, D., Pock, T., Kolev, K., and Chambolle, A. (2011). Convex relaxation techniques for segmentation, stereo and multiview reconstruction. *Advances in Markov Random Fields for Vision and Image Processing*.

Dalal, N. and Triggs, B. (2005). Histograms of oriented gradients for human detection. In *CVPR*, volume 1, pages 886–893. IEEE.

Donoser, M. and Schmalstieg, D. (2014). Discrete-continuous gradient orientation estimation for faster image segmentation. *CVPR*.

Dupont, B. (2014). Lion (panthera leo) hidden in the grass.

Fathi, A. and Rehg, J. M. (2013). Modeling actions through state changes. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 2579–2586. IEEE.

Felzenszwalb, P. and Huttenlocher, D. (2004). Distance transforms of sampled functions. Technical report, Cornell University.

Felzenszwalb, P. F., Girshick, R. B., McAllester, D., and Ramanan, D. (2010a). Object detection with discriminatively trained part-based models. *PAMI*, 32(9):1627–1645.

Felzenszwalb, P. F. and Huttenlocher, D. P. (2005). Pictorial structures for object recognition. *IJCV*, 61(1).

Felzenszwalb, P. F., Pap, G., Tardos, E., and Zabih, R. (2010b). Globally optimal pixel labeling algorithms for tree metrics. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 3153–3160. IEEE.

Felzenszwalb, P. F. and Veksler, O. (2010). Tiered scene labeling with dynamic programming. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 3097–3104. IEEE.

Feng, X., Williams, C. K., and Felderhof, S. N. (2002). Combining belief networks and neural networks for scene segmentation. *PAMI*, 24(4):467–483.

Fortun, D., Bouthemy, P., and Kervrann, C. (2014). Aggregation of local parametric candidates with exemplar-based occlusion handling for optical flow. *arXiv preprint arXiv:1407.5759*.

Fragkiadaki, K., Hu, H., and Shi, J. (2013). Pose from flow and flow from pose. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 2059–2066. IEEE.

Geiger, A., Lenz, P., and Urtasun, R. (2012). Are we ready for autonomous driving? the KITTI vision benchmark suite. In *CVPR*.

George, A. (1973). Nested dissection of a regular finite element mesh. *Journal on Numerical Analysis*, 10(2):345–363.

Glocker, B., Heibel, T. H., Navab, N., Kohli, P., and Rother, C. (2010). Triangleflow: Optical flow with triangulation-based higher-order likelihoods. *ECCV*.

Glocker, B., Paragios, N., Komodakis, N., Tziritas, G., and Navab, N. (2008). Optical flow estimation with uncertainties through dynamic MRFs. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE.

Goldluecke, B. and Cremers, D. (2010). Convex relaxation for multilabel problems with product label spaces. In *Computer Vision–ECCV 2010*, pages 225–238. Springer.

Goldstein, T., Bresson, X., Osher, S., and Chambolle, A. (2012). GLOBAL MINIMIZATION OF MARKOV RANDOM FIELDS WITH APPLICATIONS TO OPTICAL FLOW. *Inverse Problems & Imaging*, 6(4).

Greig, D., Porteous, B., and Seheult, A. H. (1989). Exact maximum a posteriori estimation for binary images. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 271–279.

Hartley, R. and Zisserman, A. (2003). *Multiple view geometry in computer vision*. Cambridge university press.

Hassner, T., Mayzels, V., and Zelnik-Manor, L. (2012). On SIFTs and their scales. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 1522–1528. IEEE.

Horn, B. K. and Schunck, B. G. (1981). Determining optical flow. *Artificial intelligence*, 17(1):185–203.

Humayun, A., Mac Aodha, O., and Brostow, G. J. (2011). Learning to find occlusion regions. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 2161–2168. IEEE.

Ishikawa, H. (2003). Exact optimization for markov random fields with convex priors. *PAMI*, 25(10).

Jepson, A. and Black, M. J. (1993). Mixture models for optical flow computation. In *Computer Vision and Pattern Recognition, 1993. Proceedings CVPR'93., 1993 IEEE Computer Society Conference on*, pages 760–761. IEEE.

Jung, H. Y., Lee, K. M., and Lee, S. U. (2008). Toward global minimum through combined local minima. In *Computer Vision–ECCV 2008*, pages 298–311. Springer.

Kang, S. B., Szeliski, R., and Chai, J. (2001). Handling occlusions in dense multi-view stereo. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–103. IEEE.

Karypis, G. and Kumar, V. (1998). METIS, a software package for partitioning unstructured graphs and computing fill-reduced orderings of sparse matrices. *University of Minnesota, Department of Computer Science*, 180.

Kearney, J. K., Thompson, W. B., and Boley, D. L. (1987). Optical flow estimation: An error analysis of gradient-based methods with local optimization. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (2):229–244.

Kennedy, R. and Taylor, C. J. (2015). Optical flow with geometric occlusion estimation and fusion of multiple frames. *EMMCVPR*.

Kim, J., Liu, C., Sha, F., and Grauman, K. (2013a). Deformable spatial pyramid matching for fast dense correspondences. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 2307–2314. IEEE.

Kim, T. H., Lee, H. S., and Lee, K. M. (2013b). Optical flow via locally adaptive fusion of complementary data costs. *ICCV*.

Koller, D. and Friedman, N. (2009). *Probabilistic graphical models: principles and techniques*. MIT press.

Koller, D., Friedman, N., Getoor, L., and Taskar, B. (2007). Graphical models in a nutshell. *STATISTICAL RELATIONAL LEARNING*, page 13.

Kolmogorov, V. (2006). Convergent tree-reweighted message passing for energy minimization. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(10):1568–1583.

Kolmogorov, V. and Zabih, R. (2001). Computing visual correspondence with occlusions using graph cuts. In *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, volume 2, pages 508–515. IEEE.

Komodakis, N., Paragios, N., and Tziritas, G. (2007). MRF optimization via dual decomposition: Message-passing revisited. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8. IEEE.

Krishnan, D., Fattal, R., and Szeliski, R. (2013). Efficient preconditioning of laplacian matrices for computer graphics. *ACM Transactions on Graphics (TOG)*, 32(4):142.

Krähenbühl, P. and Koltun, V. (2012). Efficient nonlocal regularization for optical flow. In *Computer Vision–ECCV 2012*, pages 356–369. Springer.

Kumar, S. and Hebert, M. (2005). A hierarchical field framework for unified context-based classification. In *ICCV*, volume 2, pages 1284–1291. IEEE.

Kybic, J. and Unser, M. (2003). Fast parametric elastic image registration. *Image Processing, IEEE Transactions on*, 12(11):1427–1442.

Lempitsky, V., Roth, S., and Rother, C. (2008). FusionFlow: Discrete-continuous optimization for optical flow estimation. In *CVPR*, pages 1–8. IEEE.

Leordeanu, M., Zanfir, A., and Sminchisescu, C. (2013). Locally affine sparse-to-dense matching for motion and occlusion estimation. In *Computer Vision (ICCV), 2013 IEEE International Conference on*, pages 1721–1728. IEEE.

Lipton, R. J., Rose, D. J., and Tarjan, R. E. (1979). Generalized nested dissection. *Journal on numerical analysis*, 16(2):346–358.

Liu, C., Yuen, J., and Torralba, A. (2009). Nonparametric scene parsing: Label transfer via dense scene alignment. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 1972–1979. IEEE.

Liu, C., Yuen, J., and Torralba, A. (2011). Sift flow: Dense correspondence across scenes and its applications. *PAMI*, 33(5):978–994.

Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110.

Lucas, B. D., Kanade, T., and others (1981). An iterative image registration technique with an application to stereo vision. In *IJCAI*, volume 81, pages 674–679.

Mac Aodha, O., Brostow, G. J., and Pollefeys, M. (2010). Segmenting video into classes of algorithm-suitability. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 1054–1061. IEEE.

Mac Aodha, O., Humayun, A., Pollefeys, M., and Brostow, G. J. (2013). Learning a confidence measure for optical flow. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(5):1107–1120.

Mahajan, D., Huang, F.-C., Matusik, W., Ramamoorthi, R., and Belhumeur, P. (2009). Moving gradients: a path-based method for plausible image interpolation. *ACM Transactions on Graphics (TOG)*, 28(3):42.

Martin, D., Fowlkes, C., Tal, D., and Malik, J. (2001). A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, volume 2, pages 416–423. IEEE.

Mei, X., Sun, X., Zhou, M., Wang, H., Zhang, X., and others (2011). On building an accurate stereo matching system on graphics hardware. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pages 467–474. IEEE.

Muja, M. and Lowe, D. G. (2009). Fast approximate nearest neighbors with automatic algorithm configuration. In *VISAPP*, pages 331–340.

Murray, D. W. and Buxton, B. F. (1987). Scene segmentation from visual motion using global optimization. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (2):220–228.

Nagel, H.-H. (1990). Extending the 'oriented smoothness constraint'into the temporal domain and the estimation of derivatives of optical flow. In *Computer Vision—ECCV 90*, pages 139–148. Springer.

Nagel, H.-H. and Enkelmann, W. (1986). An investigation of smoothness constraints for the estimation of displacement vector fields from image sequences. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (5):565–593.

Negahdaripour, S. (1998). Revised definition of optical flow: Integration of radiometric and geometric cues for dynamic scene analysis. *PAMI*, 20(9):961–979.

Newcombe, R. A., Lovegrove, S. J., and Davison, A. J. (2011). DTAM: Dense tracking and mapping in real-time. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2320–2327. IEEE.

Okutomi, M. and Kanade, T. (1993). A multiple-baseline stereo. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 15(4):353–363.

Papenberg, N., Bruhn, A., Brox, T., Didas, S., and Weickert, J. (2006). Highly accurate optic flow computation with theoretically justified warping. *International Journal of Computer Vision*, 67(2):141–158.

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann.

Pellegrini, S., Ess, A., Schindler, K., and Van Gool, L. (2009). You'll never walk alone: Modeling social behavior for multi-target tracking. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 261–268. IEEE.

Pock, T., Schoenemann, T., Graber, G., Bischof, H., and Cremers, D. (2008). A convex formulation of continuous multi-label problems. In *Computer Vision–ECCV 2008*, pages 792–805. Springer.

Proesmans, M., Van Gool, L., Pauwels, E., and Oosterlinck, A. (1994). Determination of optical flow and its discontinuities using non-linear diffusion. In *Computer Vision—ECCV'94*, pages 294–304. Springer.

Revaud, J., Weinzaepfel, P., Harchaoui, Z., and Schmid, C. (2014). EpicFlow: Edge-preserving interpolation of correspondences for optical flow.

Reynolds, J. and Murphy, K. (2007). Figure-ground segmentation using a hierarchical conditional random field. In *Canadian Conference on Computer and Robot Vision*, pages 175–182. IEEE.

Roosendaal, T. (2012). *Tears of Steel*. Blender Institute. `http://mango.blender.org/`.

Salgado, A. and Sánchez, J. (2007). Temporal constraints in large optical flow estimation. In *Computer Aided Systems Theory–EUROCAST 2007*, pages 709–716. Springer.

Sand, P. and Teller, S. (2008). Particle video: Long-range motion estimation using point trajectories. *International Journal of Computer Vision*, 80(1):72–91.

Sapp, B., Weiss, D., and Taskar, B. (2011). Parsing human motion with stretchable models. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1281–1288. IEEE.

Schlesinger, D. and Flach, B. (2006). *Transforming an arbitrary minsum problem into a binary one*. TU, Fak. Informatik.

Seitz, S. M. and Baker, S. (2009). Filter flow. In *ICCV*, pages 143–150. IEEE.

Sevilla-Lara, L., Sun, D., Learned-Miller, E. G., and Black, M. J. (2014). Optical flow estimation with channel constancy. In *Computer Vision–ECCV 2014*, pages 423–438. Springer.

Shi, J. and Tomasi, C. (1994). Good features to track. In *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on*, pages 593–600. IEEE.

Simoncelli, E. P., Adelson, E. H., and Heeger, D. J. (1991). Probability distributions of optical flow. In *Computer Vision and Pattern Recognition, 1991. Proceedings CVPR'91., IEEE Computer Society Conference on*, pages 310–315. IEEE.

Stein, A. N. and Hebert, M. (2009). Occlusion boundaries from motion: Low-level detection and mid-level reasoning. *International journal of computer vision*, 82(3):325–357.

Steinbrucker, F., Pock, T., and Cremers, D. (2009). Large displacement optical flow computation without warping. In *ICCV*, pages 1609–1614. IEEE.

Steinbrücker, F., Pock, T., and Cremers, D. (2009). Advanced data terms for variational optic flow estimation. In *VMV*, pages 155–164.

Stevens, K. A. (2006). Binocular vision in theropod dinosaurs. *Journal of Vertebrate Paleontology*, 26(2):321–330.

Strecha, C., Fransens, R., and Van Gool, L. (2004). A probabilistic approach to large displacement optical flow and occlusion detection. In *Statistical methods in video processing*, pages 71–82. Springer.

Strekalovskiy, E., Goldluecke, B., and Cremers, D. (2011). Tight convex relaxations for vector-valued labeling problems. In *ICCV*, pages 2328–2335. IEEE.

Sun, D., Liu, C., and Pfister, H. (2014). Local layering for joint motion estimation and occlusion detection. *CVPR*.

Sun, D., Roth, S., and Black, M. J. (2010a). Secrets of optical flow estimation and their principles. *CVPR*.

Sun, D., Roth, S., Lewis, J., and Black, M. J. (2008). Learning optical flow. In *ECCV*, pages 83–97. Springer.

Sun, D., Sudderth, E. B., and Black, M. J. (2010b). Layered image motion with explicit occlusions, temporal consistency, and depth ordering. In *NIPS*, pages 2226–2234.

Sun, D., Sudderth, E. B., and Black, M. J. (2012). Layered segmentation and optical flow estimation over time. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 1768–1775. IEEE.

Sun, D., Wulff, J., Sudderth, E. B., Pfister, H., and Black, M. J. (2013). A fully-connected layered model of foreground and background flow. *CVPR*.

Sun, S., Haynor, D., and Kim, Y. (2000). Motion estimation based on optical flow with adaptive gradients. In *Image Processing, 2000. Proceedings. 2000 International Conference on*, volume 1, pages 852–855. IEEE.

Sundberg, P., Brox, T., Maire, M., Arbeláez, P., and Malik, J. (2011). Occlusion boundary detection and figure/ground assignment from optical flow. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 2233–2240. IEEE.

Szeliski, R. (2010). *Computer vision: algorithms and applications*. Springer.

Szeliski, R. and Coughlan, J. (1994). Hierarchical spline-based image registration. In *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on*, pages 194–201. IEEE.

Szeliski, R. and Coughlan, J. (1997). Spline-based image registration. *International Journal of Computer Vision*, 22(3):199–218.

Szeliski, R. and Shum, H.-Y. (1996). Motion estimation with quadtree splines. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 18(12):1199–1210.

Szeliski, R., Zabih, R., Scharstein, D., Veksler, O., Kolmogorov, V., Agarwala, A., Tappen, M., and Rother, C. (2006). A comparative study of energy minimization methods for markov random fields. In *Computer Vision–ECCV 2006*, pages 16–29. Springer.

Ullman, S. (1979). The interpretation of structure from motion. *Proceedings of the Royal Society of London. Series B. Biological Sciences*, 203(1153):405–426.

Vogel, C., Roth, S., and Schindler, K. (2013). An evaluation of data costs for optical flow. In *Pattern Recognition*, pages 343–353. Springer.

Vogel, C., Roth, S., and Schindler, K. (2014). View-consistent 3d scene flow estimation over multiple frames. In *Computer Vision–ECCV 2014*, pages 263–278. Springer.

Volz, S., Bruhn, A., Valgaerts, L., and Zimmer, H. (2011). Modeling temporal coherence for optical flow. In *ICCV*, pages 1116–1123. IEEE.

Wainwright, M., Jaakkola, T., and Willsky, A. (2002). MAP estimation via agreement on (hyper) trees: Message-passing and linear programming approaches. In *PROCEEDINGS OF THE ANNUAL ALLERTON CONFERENCE ON COMMUNICATION CONTROL AND COMPUTING*, volume 40, pages 1565–1575. The University; 1998.

Wang, H., Klaser, A., Schmid, C., and Liu, C.-L. (2011). Action recognition by dense trajectories. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 3169–3176. IEEE.

Wang, L., Shi, J., Song, G., and Shen, I.-f. (2007). Object detection combining recognition and segmentation. In *Computer Vision–ACCV 2007*, pages 189–199. Springer.

Wedel, A., Cremers, D., Pock, T., and Bischof, H. (2009a). Structure-and motion-adaptive regularization for high accuracy optic flow. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 1663–1668. IEEE.

Wedel, A., Pock, T., Zach, C., Bischof, H., and Cremers, D. (2009b). An improved algorithm for TV-l 1 optical flow. In *Statistical and Geometrical Approaches to Visual Motion Analysis*, pages 23–45. Springer.

Weickert, J. and Schnörr, C. (2001). Variational optic flow computation with a spatio-temporal smoothness constraint. *Journal of Mathematical Imaging and Vision*, 14(3):245–255.

Weinzaepfel, P., Revaud, J., Harchaoui, Z., and Schmid, C. (2013). DeepFlow: Large displacement optical flow with deep matching. *ICCV*.

Werlberger, M., Pock, T., and Bischof, H. (2010). Motion estimation with non-local total variation regularization. In *CVPR*, pages 2464–2471. IEEE.

Werlberger, M., Trobin, W., Pock, T., Wedel, A., Cremers, D., and Bischof, H. (2009). Anisotropic huber-l1 optical flow. In *BMVC*, page 3.

Wu, S., He, X., Lu, H., and Yuille, A. L. (2010). A unified model of short-range and long-range motion perception. In *Advances in Neural Information Processing Systems*, pages 2478–2486.

Xiao, J., Cheng, H., Sawhney, H., Rao, C., and Isnardi, M. (2006). Bilateral filtering-based optical flow estimation with occlusion detection. In *Computer Vision–ECCV 2006*, pages 211–224. Springer.

Xu, L., Jia, J., and Matsushita, Y. (2012). Motion detail preserving optical flow estimation. *PAMI*, 34(9):1744–1757.

Yamaguchi, K., McAllester, D., and Urtasun, R. (2013). Robust monocular epipolar flow estimation. In *CVPR*, pages 1862–1869. IEEE.

Yamaguchi, K., McAllester, D., and Urtasun, R. (2014). Efficient joint segmentation, occlusion labeling, stereo and flow estimation. In *Computer Vision–ECCV 2014*, pages 756–771. Springer.

Yang, Q., Wang, L., Yang, R., Stewénius, H., and Nistér, D. (2009). Stereo matching with color-weighted correlation, hierarchical belief propagation, and occlusion handling. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 31(3):492–504.

Zabih, R. and Woodfill, J. (1994). Non-parametric local transforms for computing visual correspondence. In *Computer Vision—ECCV'94*, pages 151–158. Springer.

Zach, C. (2014). Robust bundle adjustment revisited. In *Computer Vision–ECCV 2014*, pages 772–787. Springer.

Zach, C., Pock, T., and Bischof, H. (2007). A duality based approach for realtime TV-l 1 optical flow. In *Pattern Recognition*, pages 214–223. Springer.

Zhang, Z. (1997). Parameter estimation techniques: A tutorial with application to conic fitting. *Image and vision Computing*, 15(1):59–76.

Zhu, L., Chen, Y., Lin, Y., Lin, C., and Yuille, A. (2012). Recursive segmentation and recognition templates for image parsing. *PAMI*, 34(2):359–371.

Zimmer, H., Bruhn, A., and Weickert, J. (2011). Optic flow in harmony. *International Journal of Computer Vision*, 93(3):368–388.

Zimmer, H., Bruhn, A., Weickert, J., Valgaerts, L., Salgado, A., Rosenhahn, B., and Seidel, H.-P. (2009). Complementary optic flow. In *Energy minimization methods in computer vision and pattern recognition*, pages 207–220. Springer.

Zitnick, C. L., Kang, S. B., Uyttendaele, M., Winder, S., and Szeliski, R. (2004). High-quality video view interpolation using a layered representation. In *ACM Transactions on Graphics (TOG)*, volume 23, pages 600–608. ACM.