Institute for Visualization and Interactive Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

PatchMatch Algorithms for Motion Estimation and Stereo Reconstruction

Fangjun Kuang

Course of Study: Master of Science Information Technology

Examiner: Prof. Dr.-Ing. Andrés Bruhn

Supervisor: Prof. Dr.-Ing. Andrés Bruhn

Commenced: June 1, 2017

Completed: December 1, 2017

CR-Classification: I.2.10, I.4.8

Abstract

Correspondence problems are difficult and fundamental topics in computer vision, which aim to find the displacement field between two consecutive images within an image sequence. Both stereo matching and motion estimation belong to the domain of correspondence problems. A common technique is to define a parametric model and estimate its parameters via some optimization algorithms. In this context, this thesis extends the recently proposed PatchMatch algorithm [BSFG09], which is originally designed for finding approximate nearest neighbors, to model parameter estimation. Specifically, there are three purposes of this thesis: (i) We analyze and extend the PatchMatch algorithm to model parameter estimation. (ii) Afterwards, some commonly used parametric models for motion estimation and stereo matching in the literature are reviewed. (iii) Finally, the extended PatchMatch algorithm is implemented and applied to estimate the model parameters summarized above. The estimation performance is evaluated and compared with some other methods in the literature based on the three public benchmarks: Middlebury, KITTI and MPI Sintel.

Acknowledgements

First of all, I would like to thank Prof. Dr. Andrés Bruhn for supervising my work and providing me insightful suggestions. I also thank Michael Stoll for his optimization framework, which simplifies the task of tuning parameters a lot. Furthermore, I thank Cong Xiao and Junying Zhao for proofreading. Last but not least, I have to thank my family for their continuous support and great help during my studies.

Contents

1 Introduction **9**
1.1 Motivation 9
1.2 Related Work 10
1.3 Goal of This Thesis 10
1.4 Organization 10

2 Background **11**
2.1 Correspondence Problems 11
2.2 Visualization Techniques 12
2.3 Matching Criteria 12
2.4 The Block Matching Method 14
2.5 Image Descriptors 16
2.5.1 The SIFT Flow Descriptor 16
2.5.2 The Rank Transform 17
2.5.3 The Census Transform 18
2.5.4 The Complete Rank Transform 18
2.5.5 The Complete Census Transform 18
2.5.6 Visualization of Different Descriptors 19
2.6 The Coarse-to-fine Approach 20

3 The PatchMatch Algorithm **25**
3.1 Assumptions of the PatchMatch Algorithm 25
3.2 The PatchMatch Algorithm 25
3.2.1 Random Initialization 26
3.2.2 Iteration 26

4 PatchMatch for Motion Estimation **31**
4.1 The Motion Estimation Pipeline 31
4.1.1 Coarse-to-fine PatchMatch 31
4.1.2 Post-Processing 34
4.2 Motion Models 36
4.2.1 The Translational Motion Model 36
4.2.2 The Affine Motion Model 36
4.2.3 The Projective Motion Model 37
4.2.4 The Quadratic Motion Model 37
4.3 Cost Functions 38
4.4 The Coarse-to-fine Approach 39

5 PatchMatch for Stereo Reconstruction **41**
5.1 The Stereo Reconstruction Pipeline 41
5.2 Models for Stereo Matching 43
5.2.1 The Epipolar Constraint Model 43
5.2.2 The Projective Planar Model 44
5.2.3 The Slanted Plane Model 44
5.3 Cost Functions 45
5.4 View Propagation 46
5.5 Post-Processing 46

6 Evaluation and Experiments: Motion Estimation **49**
6.1 Performance Measures for Motion Estimation 49
6.2 Parameter Optimization 50
6.3 Evaluation On the Middlebury Dataset 50
6.3.1 Optimized Parameters 50
6.3.2 Comparison of Different Motion Models and Image Descriptors . . . 51
6.3.3 Comparison with Other Methods 54
6.4 Evaluation On the KITTI Dataset 55
6.4.1 Optimized Parameters 55
6.4.2 Comparison of Different Motion Models 57
6.4.3 Comparison with Other Methods 58
6.5 Evaluation On the MPI Sintel Dataset 58
6.5.1 Parameter Settings 58
6.5.2 Comparison of Different Motion Models 58
6.5.3 Comparison with Other Methods 59

7 Evaluation and Experiments: Stereo Matching **61**
7.1 Performance Measures for Stereo Matching 61
7.2 Parameter Settings 61
7.3 Middlebury Stereo Evaluation Version 2 62
7.4 Middlebury Stereo Evaluation Version 3 63

8 Conclusion **67**
8.1 Future Work 67

Bibliography **69**

1 Introduction

1.1 Motivation

Correspondence problems are key topics in computer vision. One example is motion estimation, also known as optical flow estimation, whose goal is to estimate the displacement field describing the movement of every pixel between two images. Another example is stereo reconstruction, which tries to reconstruct the depth[1] of every pixel given two images of a static scene captured by two cameras at the same time from different viewpoints. One of the most challenging parts in stereo reconstruction is to identify the displacement between pixels, which is called stereo matching [HZ03].

The application fields of correspondence problems are extensive and diverse. Since optical flow is able to detect motions, it allows to track objects. In combination with methods from pattern recognition and machine learning, action recognition such as hand clapping, walking and running can be realized [BFGV17; EGA+17; UIM08]. Some other application fields are robots navigation and autonomous car driving, where optical flow is used as input to the control systems for automatic guidance [CGN14; GLSG10; SBM06]. By virtue of the capability of stereo reconstruction to reconstruct the 3-D structure of an unknown environment, it is able to aid robots in moving around an unknown environment or in localizing itself in a known map [STR+13; TUI17].

Since the seminal works of [HS81; LK81] published more than 30 years ago, correspondence problems remain to be active research topics. The dominant approaches to solve correspondence problems are variational methods [BBPW04; BM11; BWS05; PBB+06; SRB10; VSR13], which define an energy model containing a data term and a smoothness term. The data term penalizes deviations from the assumption that certain image features, such as the intensity value between corresponding pixels, remain to be constant, while the smoothness term ensures the final result is unique and piecewise smooth. The Euler-Lagrange framework [GZG+10] is usually exploited to solve the defined model. The recently proposed PatchMatch algorithm [BSFG09; BSGF10] has been applied to motion estimation [HSL16] and stereo matching [BRR11]. Moreover, the estimation performances rank high in the public benchmarks and PatchMatch appears to be a promising alternative to the commonly used variational methods. PatchMatch is very fast and efficient to find approximate nearest neighbors via a series of random searches and propagations. The way of solving correspondence problems using PatchMatch is totally different from that using

[1]In binocular stereo reconstruction, the depth of an object is its distance to the baseline, which is the line passing through the optical centers of the two cameras. [CS11].

traditional variational methods. Due to its high performance in finding approximate nearest neighbors, it is very likely to achieve good results in the field of model parameter estimation. However, this has not been thoroughly studied in the literature. The performance of PatchMatch in model parameter estimation compared to the conventional methods based on optimization algorithms remains to be answered.

1.2 Related Work

Barnes et al. [BSFG09] first presented the PatchMatch algorithm for finding approximate nearest neighbors and used it for structural image editing. Bao et al. [BYJ14] and Hu et al. [HSL16] applied the PatchMatch algorithm to displacement field estimation, assuming that the motion is translational. Bleyer [BRR11] applied PatchMatch to stereo matching with the assumption that pixels are projected from a 3-D slanted plane and PatchMatch is used to estimate the normal vector of the plane. Hu et al. [HLS17] assumed a piecewise affine motion model within a superpixel and applied PatchMatch to estimate its parameters.

1.3 Goal of This Thesis

The goal of this thesis is to implement the PatchMatch algorithm [BSFG09] and extend it to estimate the parameters of commonly used parametric models in the field of motion estimation and stereo matching. In addition, the performance should be evaluated based on the public benchmarks, such as Middlebury [BSL+11; SS02], KITTI [GLU12; MG15] and MPI Sintel [BWSB12].

1.4 Organization

This thesis is structured as follows: Chapter 2 presents some background for correspondence problems. The PatchMatch algorithm is described in Chapter 3. In order to estimate model parameters, we summarize the commonly used parametric models for motion estimation and stereo matching in Chapter 4 and Chapter 5 followed by the estimation of their parameters based on the PatchMatch algorithm. The estimation results for motion estimation and stereo matching are evaluated in Chapter 6 and Chapter 7. Finally, we conclude this thesis in Chapter 8.

2 Background

This chapter introduces basic concepts about correspondence problems in computer vision and notations used throughout this thesis. Section 2.1 describes what correspondence problems are and their relationships between motion estimation and stereo matching. The widely used visualization techniques for displacement field in the literature are illustrated in Section 2.2 followed by the introduction of matching criteria in Section 2.3. The block matching method, which is one of the oldest methods for correspondence problems, is described in Section 2.4. In order to improve the estimation performance, image descriptors and the coarse-to-fine approach are discussed in Section 2.5 and Section 2.6.

2.1 Correspondence Problems

Given two entities $F = \{f_1, f_2, \dots, f_N\}$ and $G = \{g_1, g_2, \dots, g_N\}$, correspondence problems are to find correspondences (f_i, g_j) satisfying some matching criteria between these two entities. Figure 2.1 gives an example when $N = 4$. Different matching criteria can lead to completely different correspondences.

$$\begin{tikzcd}
f_1 \arrow[r] & g_1 \\
f_2 \arrow[r] & g_2 \\
f_3 \arrow[r] & g_3 \\
f_4 \arrow[r] & g_4
\end{tikzcd}$$

(a) One-to-One

$$\begin{tikzcd}
f_1 \arrow[dr] & g_1 \\
f_2 \arrow[r] & g_2 \\
f_3 \arrow[r] & g_3 \\
f_4 \arrow[r] & g_4
\end{tikzcd}$$

(b) Many-to-One

$$\begin{tikzcd}
f_1 \arrow[r] & g_1 \\
f_2 \arrow[r] \arrow[ur] & g_2 \\
f_3 \arrow[dr] & g_3 \\
f_4 \arrow[ur] & g_4
\end{tikzcd}$$

(c) One-to-Many

$$\begin{tikzcd}
f_1 \arrow[r] & g_1 \\
f_2 & g_2 \\
f_3 \arrow[r] & g_3 \\
f_4 \arrow[r] & g_4
\end{tikzcd}$$

(d) Non-Dense

Figure 2.1: Illustration of the correspondence problems when $N = 4$. (a) There is a one-to-one mapping between $f_i \in F$ and $g_j \in G$. (b) Multiple elements in F correspond to the same element in G. (c) A single element in F corresponds to multiple elements in G. (d) Some elements in F have no correspondences in G. Image source: Lecture slides from the course Correspondence Problems in Computer Vision taught by Prof. Andrés Bruhn, Summer Semester 2016, Universität Stuttgart, http://www.vis.uni-stuttgart.de/nc/lehre/details/typ/vorlesung/2016/240.html.

In the field of computer vision, the given two entities are usually two images $\mathbf{f} = (f_{1,1}, f_{1,2}, \dots, f_{N,M})^\top$ and $\mathbf{g} = (g_{1,1}, g_{1,2}, \dots, g_{N,M})^\top$, where N denotes the image width and M is the image height. If the corresponding pixel of $f_{i,j}$ is $g_{i+u,j+v}$, then we call $(u, v)^\top$ the displacement field or the optical flow. The displacement field of $f_{i,j}$ is denoted as $\mathbf{u}_{i,j} = (u_{i,j}, v_{i,j})^\top$. Optical flow estimation and motion estimation are used interchangeably in this thesis.

In the case of motion estimation, the goal is to find the displacement field $\mathbf{u} = (\mathbf{u}_{1,1}^\top, \mathbf{u}_{1,2}^\top, \dots, \mathbf{u}_{N,M}^\top)^\top$ given two images $\mathbf{f}$ and $\mathbf{g}$. In the field of binocular stereo reconstruction, $\mathbf{f}$ and $\mathbf{g}$ are the rectified left image and right image respectively. A pair of images is said to be rectified when there is only horizontal motion between them [FTV00]. In this case, $\mathbf{u}_{i,j} = (u_{i,j}, 0)^\top$, which is also called the disparity of $f_{i,j}$, and the goal to find $\mathbf{u}_{i,j}$ is called stereo matching. Since the depth is inversely proportional to the disparity, once the disparity map is obtained, the depth can be reconstructed with known camera calibration via triangulation [CS11; Sze10].

2.2 Visualization Techniques

The displacement field is often visualized as an image. In the case of motion estimation, where the motion field is a vector containing two components, the brightness indicates the magnitude of the vector and the color represents its direction. Figure 2.2 shows the visualization of the Yosemite sequence with clouds[1].

The disparity in stereo matching is just a scalar and is usually visualized as a grayscale image, where the brightness indicates the magnitude. Starting from the Middlebury stereo dataset 2014 [SHK+14], the disparity map is additionally encoded as a color image for visualization[2]. Figure 2.3 visualizes the disparity map of the Teddy stereo pair from the Middlebury dataset [SS02]. Since the disparity is inversely proportional to the depth, pixels with a brighter intensity value in Figure 2.3 (c) have a smaller distance to the camera, while pixels with a darker intensity have a larger distance to the camera. Occluded pixels are shown in black.

2.3 Matching Criteria

Matching criteria in correspondence problems determine the correspondence rules. This section introduces the matching criteria widely used in correspondence problems.

[1]The sequence is available at the address http://www.informatik.uni-ulm.de/ni/staff/PBayerl/homepage/animations/index.html, which was created by Lynn Quam at SRI.

[2]The color coding scheme is available at the address https://github.com/roboception/cvkit/blob/master/gimage/color.h.

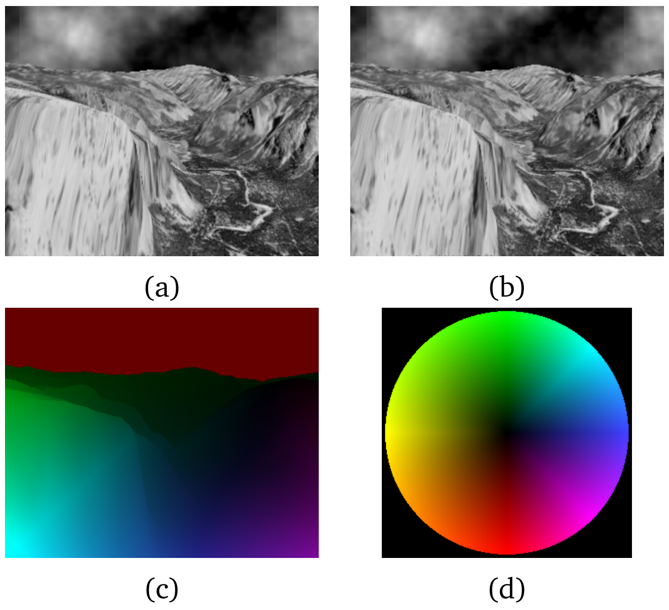


Figure 2.2: Visualization of the Yosemite sequence with clouds. (a) Frame 7. (b) Frame 8. (c) Ground truth. (d) Color code.

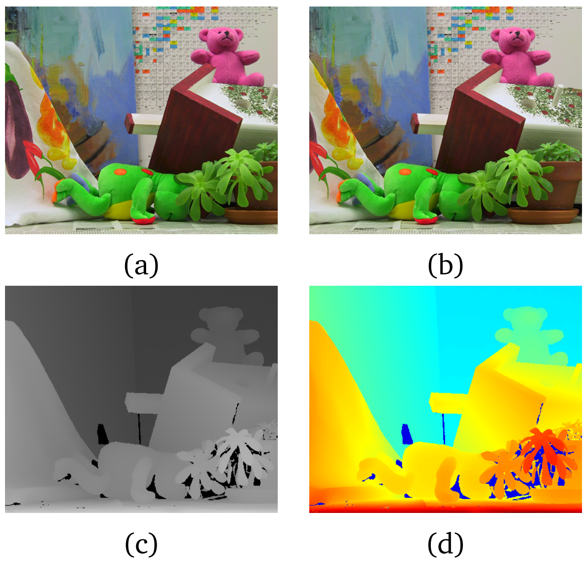


Figure 2.3: Visualization of the Teddy stereo pair from the Middlebury dataset [SS02]. (a) Image 2 (left view). (b) Image 6 (right view). (c) Visualization of the ground truth disparity map of Image 2 with respect to Image 6 in grayscale image. (d) Visualization of the ground truth disparity map in color.

When the displacement is small, we assume that the brightness of pixels does not change between the two images, which is the well-known brightness constancy assumption. Under this assumption, corresponding pixels should have similar brightness. Subsequently, the matching criterion is to find (u, v), such that

$$f_{i,j} - g_{i+u,j+v} = 0 \tag{2.1}$$

It is very common that the brightness constancy assumption is violated in practice. That is, for some pixels, Equation (2.1) does not hold. To find the motion field for every pixel, we can minimize the following cost function

$$\mathbf{u}_{i,j} = \arg\min_{u,v} \, (f_{i,j} - g_{i+u,j+v})^2 \tag{2.2}$$

In Equation (2.2), it needs to search the entire image to find a solution. To limit the search space, we assume that the magnitude of the entries of the motion field are bounded by d

$$|u_{i,j}| \leq d \, , \, |v_{i,j}| \leq d$$

Then Equation (2.2) changes to

$$\mathbf{u}_{i,j} = \arg\min_{u,v \in \mathcal{S}_d} (f_{i,j} - g_{i+u,j+v})^2 \tag{2.3}$$

where $\mathcal{S}_d$ is a square window of size $(2d+1) \times (2d+1)$ centered at the pixel location (i, j). For every pixel, it needs to perform $(2d+1)^2$ searches. For an image of size $N \times M$, the number of searches is $(2d+1)^2 MN$.

2.4 The Block Matching Method

Instead of computing the matching cost between a pair of single pixels, the block matching method incorporates the neighborhood around the matching pixels, which has the following form

$$\mathbf{u}_{i,j} = \arg\min_{u,v \in \mathcal{S}_d} \sum_{\delta_i=-m}^{m} \sum_{\delta_j=-m}^{m} \left(f_{i+\delta_i,j+\delta_j} - g_{(i+u)+\delta_i,(j+v)+\delta_j} \right)^2 \tag{2.4}$$

where m is the neighborhood size.

The above matching cost is the so-called sum-of-squared differences (SSD) [BFB94]. There are some other kinds of matching costs such as sum-of-absolute differences (SAD) [SS02]

$$\mathbf{u}_{i,j} = \arg\min_{u,v \in \mathcal{S}_d} \sum_{\delta_i=-m}^{m} \sum_{\delta_j=-m}^{m} \left| f_{i+\delta_i,j+\delta_j} - g_{(i+u)+\delta_i,(j+v)+\delta_j} \right| \tag{2.5}$$

and normalized cross correlation (NCC) [Gia00]

$$\mathbf{u}_{i,j} = \arg\max_{u,v\in\mathcal{S}_d} \frac{\sum\limits_{\delta_i=-m}^{m}\sum\limits_{\delta_j=-m}^{m}\left(f_{i+\delta_i,j+\delta_j}-\bar{f}_{i,j}\right)\left(g_{(i+u)+\delta_i,(j+v)+\delta_j}-\bar{g}_{i+u,j+v}\right)}{\sqrt{\sum\limits_{\delta_i=-m}^{m}\sum\limits_{\delta_j=-m}^{m}\left(f_{i+\delta_i,j+\delta_j}-\bar{f}_{i,j}\right)^2}\sqrt{\sum\limits_{\delta_i=-m}^{m}\sum\limits_{\delta_j=-m}^{m}\left(g_{(i+u)+\delta_i,(j+v)+\delta_j}-\bar{g}_{i+u,j+v}\right)^2}} \tag{2.6}$$

where

$$\bar{f}_{i,j} = \frac{1}{(2m+1)^2}\sum_{\delta_i=-m}^{m}\sum_{\delta_j=-m}^{m} f_{i+\delta_i,j+\delta_j} \tag{2.7}$$

and

$$\bar{g}_{i+u,j+v} = \frac{1}{(2m+1)^2}\sum_{\delta_i=-m}^{m}\sum_{\delta_j=-m}^{m} g_{(i+u)+\delta_i,(j+v)+\delta_j} \tag{2.8}$$

Note that we have to minimize SAD and SSD (cost) but instead to maximize NCC (similarity). Figure 2.4 shows the plots of $y = x^2$ and $y = |x|$, which illustrate that $y = x^2$ increases much faster than $y = |x|$. SAD is therefore more robust to outliers since they are penalized less severely.

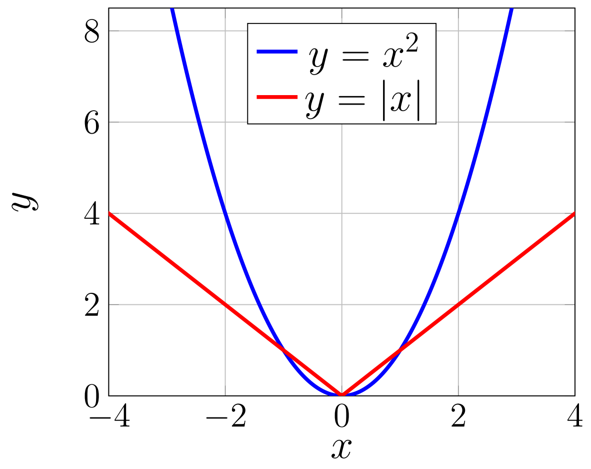


Figure 2.4: Plots of $y = x^2$ and $y = |x|$.

NCC is invariant with respect to global affine illumination changes

$$f^*_{i,j} = af_{i,j} + b \tag{2.9}$$

The additive component b is canceled out via the mean compensation, while the normalization eliminates the multiplicative component a. However, NCC is more computationally costly compared with SSD and SAD. Meanwhile, it expects the block size to be larger than 1×1 and is not defined in homogeneous areas where the variance is 0.

The block matching method performs exhaustive searches in the given region to find the exact nearest neighbor. For an image of size $N \times M$, the complexity is $O\left((2d+1)^2(2m+1)^2MN\right)$. Although it is linear in the image size when m and d are fixed, $(2d+1)^2(2m+1)^2$ is usually quite large and therefore it has very poor efficiency. Moreover, it does not provide sub-pixel precision, as it assumes an integer displacement. On the other hand, the block matching method improves the uniqueness of the matches and is less sensitive to noise since it integrates the neighborhood information into the estimation process.

2.5 Image Descriptors

Although NCC is invariant to global affine illumination changes, it is computationally intensive and the invariance type is limited. This is where image descriptors come into play. Image descriptors have proved to be very successful both in optical flow estimation [BM11; BTS15; HSL16; SVB12; WRHS13] and in stereo matching [MCUP04; SZ02; TLF10]. The descriptor image $\mathbf{F}$ of the image $\mathbf{f}$ is obtained via some transformation $\mathcal{T}$. For a grayscale image $\mathbf{f}$, the pixel $f_{i,j} \in \mathbb{R}$ is a real value and its corresponding pixel in the descriptor image $\mathbf{F}_{i,j} \in \mathbb{R}^k$ is a vector containing k elements. $\mathbf{F}_{i,j}$ is obtained through $\mathcal{T}$ by integrating the neighborhood of $f_{i,j}$ and it possesses usually one or several kinds of the following invariances: illumination changes (e.g., additive, multiplicative, affine and morphological) and image deformations (e.g., affine transformations, scaling and rotation). When $\mathbf{f}$ is a color image, i.e., $\mathbf{f}_{i,j} \in \mathbb{R}^3$, this thesis first converts it to a grayscale image and computes the descriptor image based on the grayscale image. Note that there are some other alternatives. For instance, one may first compute the descriptor images per color channel and then concatenate them together as the final descriptor image.

There are in general two kinds of descriptors: sparse and dense. Sparse descriptors are computed only at some key locations, e.g., corners, such as the SIFT (scale-invariant feature transform) feature descriptor [Low04], while dense descriptors are computed at every location, for instance, the SIFT flow descriptor [LYT11]. In the case of the sparse descriptors we are only able to obtain the displacement field at locations where descriptors are available, hence the resulting displacement field is sparse. Since this thesis aims to obtain a dense displacement field, we consider only dense descriptors.

In this section, we present five descriptors that are used in this thesis, which are the SIFT flow descriptor [LYT11], the rank transform [ZW94], the census transform [ZW94], the complete rank transform [Dem15; DHW13] and the complete census transform [Dem15; DHW13].

2.5.1 The SIFT Flow Descriptor

Different from the SIFT feature descriptor, which is computed only at key points and is sparse, the SIFT flow descriptor is computed at every location and is thus dense. What they have in common is that they have the same feature extraction component and both contain 128 elements in the descriptor. The source code for extracting the SIFT flow descriptor from an image is available at the authors' website [LYT], which takes as input an RGB color image and outputs its SIFT flow descriptor image. According to the evaluation performed by Mikolajczyk et al. [MS05], the SIFT feature descriptor is the second most resistant descriptor among ten to affine illumination changes and image deformations, such as affine transformations, scale changes, rotation, blur and JPEG compression. Since the SIFT flow descriptor shares the same feature extraction component with the SIFT feature descriptor, it is likely to inherit the merits of the SIFT feature descriptor, which is demonstrated by the experimentation conducted by Liu et al. [LYT11].

2.5.2 The Rank Transform

Rank transform transforms the intensity of a pixel to a rank, which is defined as the number of pixels in the chosen neighborhood whose intensities are less than the reference one. Given a window of size $(2m+1) \times (2m+1)$ centered at the pixel $f_{i,j}$, the rank of $f_{i,j}$ is computed using Equation (2.10)

$$\mathcal{T}_r\left(f_{i,j}\right) := \sum_{\delta_i=-m}^{m} \sum_{\delta_j=-m}^{m} \mathbb{1}_{\left(f_{i+\delta_i,j+\delta_j} < f_{i,j}\right)} \tag{2.10}$$

where $\mathbb{1}_{(\cdot)}$ is the indicator function defined as

$$\mathbb{1}_{(x)} := \begin{cases} 1 & \text{if } x = 1, \\ \\ 0 & \text{otherwise.} \end{cases} \tag{2.11}$$

The rank transform is robust under illumination changes. When the pixel values are increasing or decreasing monotonically, the relative order among them does not change. Hence it is morphologically invariant [Dem15], i.e., the rank of a pixel is kept fixed in case of global monotonically rescaling of the pixel values. An example of how to perform the rank transform is given in Figure 2.5.

9	11	22
5	22	10
1	33	35

(a)

	$f_{0,0}$	$f_{1,0}$	$f_{2,0}$	$f_{0,1}$	$f_{1,1}$	$f_{2,1}$	$f_{0,2}$	$f_{1,2}$	$f_{2,2}$
Condition	$9 < 22$	$11 < 22$	$22 < 22$	$5 < 22$	▨	$10 < 22$	$1 < 22$	$33 < 22$	$35 < 22$
Indicator	1	1	0	1	▨	1	1	0	0
$\mathcal{T}_r(f_{1,1}) = 1+1+0+1+1+1+0+0 = 5$									

(b)

Figure 2.5: Example of the rank transform. (a) A patch of size 3×3 centered at the reference pixel $f_{1,1}$ marked with gray color. (b) The process of computing the rank of $f_{1,1}$.

2.5.3 The Census Transform

Different from the rank transform, the census transform records additionally the relative intensity order of the neighbors of the reference pixel. For a window of size of $n := (2m+1) \times (2m+1)$, the census of the pixel $f_{i,j}$ is a bit string of length $n-1$ consisting of 1s and 0s : $\mathcal{T}_c\left(f_{i,j}\right) \in \{0,1\}^{n-1}$. The ordering of bits can be arbitrary, as long as it is consistent across different images. We use the reverse scan order. That is, the least significant bit represents the order of the neighbor at the bottom right, while the most significant bit indicates the order of the neighbor at the top left. Bit k is 1 if the intensity of the k-th neighbor counting from the bottom right is less than $f_{i,j}$, otherwise it is 0. For efficiency reasons during implementation, the bits of eight neighbors are concatenated into one byte. For instance, when the patch size is 3×3, the result can be saved exactly into one byte; and when the neighborhood size is 5×5, the result can be stored into three bytes. It is simple to prove that when the patch size is $n := (2m+1) \times (2m+1)$, where $m \geq 1$, it takes $\frac{(n-1)^2}{8}$ bytes to save the result. Similar to the rank transform, the census transform is morphologically invariant [Dem15]. Figure 2.6 illustrates the procedure of computing a census.

	$f_{0,0}$	$f_{1,0}$	$f_{2,0}$	$f_{0,1}$	$f_{1,1}$	$f_{2,1}$	$f_{0,2}$	$f_{1,2}$	$f_{2,2}$
Condition	$9 < 22$	$11 < 22$	$22 < 22$	$5 < 22$	▨	$10 < 22$	$1 < 22$	$33 < 22$	$35 < 22$
Indicator	1	1	0	1	▨	1	1	0	0
$\mathcal{T}_c(f_{1,1}) = \texttt{0b1101\_1100}$									

Figure 2.6: Example of the census transform of $f_{1,1}$ from Figure 2.5 (a). The result is saved in one byte.

2.5.4 The Complete Rank Transform

While the rank transform stores only one rank number of the reference pixel and discards the information of the neighboring pixels, the complete rank transform $\mathcal{T}_{cr}$ saves the rank of every pixel in the patch. For instance, given a patch of size 3×3 centered at the pixel $f_{i,j}$, $f_{i,j}$ is transformed to a vector with nine elements instead of one element, i.e., $\mathcal{T}_{cr}(f_{i,j}) \in \mathbb{R}^9$. Similar to the census transform, the ordering of elements is irrelevant, providing that it is used consistently. Here we use the scan order. The complete rank transform is morphologically invariant [Dem15], since it only adds additional ranks to the rank transform and it inherits therefore the invariance properties from the rank transform. Figure 2.7 illustrates how the complete rank transform works.

2.5.5 The Complete Census Transform

In addition to the computation of the census of the reference pixel in the census transform, the complete census transform $\mathcal{T}_{cc}$ computes as well the censuses of other pixels in the

$\mathcal{T}_r(f_{0,0}) = 0+0+1+0+0+1+0+0 = 2$	$\mathcal{T}_r(f_{0,1}) = 1+0+1+0+1+1+0+0 = 4$	$\mathcal{T}_r(f_{0,2}) = 1+1+1+0+1+1+0+0 = 5$
$\mathcal{T}_r(f_{0,1}) = 0+0+0+0+0+1+0+0 = 1$	$\mathcal{T}_r(f_{1,1}) = 1+1+0+1+1+1+0+0 = 5$	$\mathcal{T}_r(f_{2,1}) = 1+0+0+1+0+1+0+0 = 3$
$\mathcal{T}_r(f_{0,2}) = 0+0+0+0+0+0+0+0 = 0$	$\mathcal{T}_r(f_{1,2}) = 1+1+1+1+1+1+1+0 = 7$	$\mathcal{T}_r(f_{2,2}) = 1+1+1+1+1+1+1+1 = 8$
$\mathcal{T}_{cr}(f_{1,1}) = \big(\mathcal{T}_r(f_{0,0}), \mathcal{T}_r(f_{1,0}), \mathcal{T}_r(f_{2,0}), \mathcal{T}_r(f_{0,1}), \mathcal{T}_r(f_{1,1}), \mathcal{T}_r(f_{2,1}), \mathcal{T}_r(f_{0,2}), \mathcal{T}_r(f_{1,2}), \mathcal{T}_r(f_{2,2})\big) = (2,4,5,1,5,3,0,7,8)$		

Figure 2.7: Illustration of the complete rank transform of $f_{1,1}$ from Figure 2.5 (a). The result is saved into nine bytes.

patch. It contains more information than the census transform since the intensity order of one pixel with respect to all other pixels in the patch is also recorded. Again like the census transform, we concatenate eight censuses into one byte. For a patch of size $n := (2m+1) \times (2m+1)$, the number of resulting bytes is $\frac{n(n-1)}{8}$. Like the census transform, the complete census transform is morphologically invariant [Dem15]. Figure 2.8 demonstrates how the complete census transform works.

$\mathcal{T}_c(f_{0,0}) =$ `0b0010_0100`	$\mathcal{T}_c(f_{0,1}) =$ `0b1010_1100`	$\mathcal{T}_c(f_{0,2}) =$ `0b1110_1100`
$\mathcal{T}_c(f_{0,1}) =$ `0b0000_0100`	$\mathcal{T}_c(f_{1,1}) =$ `0b1101_1100`	$\mathcal{T}_c(f_{2,1}) =$ `0b1001_0100`
$\mathcal{T}_c(f_{0,2}) =$ `0b0000_0000`	$\mathcal{T}_c(f_{1,2}) =$ `0b1111_1110`	$\mathcal{T}_c(f_{2,2}) =$ `0b1111_1111`
$\mathcal{T}_{cc}(f_{1,1}) = \big(\mathcal{T}_c(f_{0,0}), \mathcal{T}_c(f_{1,0}), \mathcal{T}_c(f_{2,0}), \mathcal{T}_c(f_{0,1}), \mathcal{T}_c(f_{1,1}), \mathcal{T}_c(f_{2,1}), \mathcal{T}_c(f_{0,2}), \mathcal{T}_c(f_{1,2}), \mathcal{T}_c(f_{2,2})\big)$		

Figure 2.8: Illustration of the complete census transform of $f_{1,1}$ from Figure 2.5 (a). The result is saved into nine bytes.

Table 2.1 lists the patch sizes used to compute different descriptors in this thesis.

Descriptor type	**Patch size**
The SIFT flow descriptor	16×16
The rank transform	5×5
The census transform	5×5
The complete rank transform	5×5
The complete census transform	5×5

Table 2.1: The patch sizes used for computing different descriptors in this thesis.

2.5.6 Visualization of Different Descriptors

Figure 2.9 visualizes the descriptor images of the SIFT flow descriptor, the rank transform and the census transform of the Teddy sequence from the Middlebury dataset [SS02]. The 128-D SIFT flow descriptor is projected to 3-D subspace via PCA (principal components analysis) [Str16] for visualization. The PCA basis data are available from the authors' website [LYT]. Since the complete rank transform and the complete census transform have dimensions greater than three and there are no PCA basis data available, they are not visualized here. It can be seen that the descriptor images preserve the general structure of the original image. Due to the additional invariance properties provided by the descriptors and their successful applications in optical flow and stereo matching, this thesis uses descriptor images instead of raw images for computing matching costs.

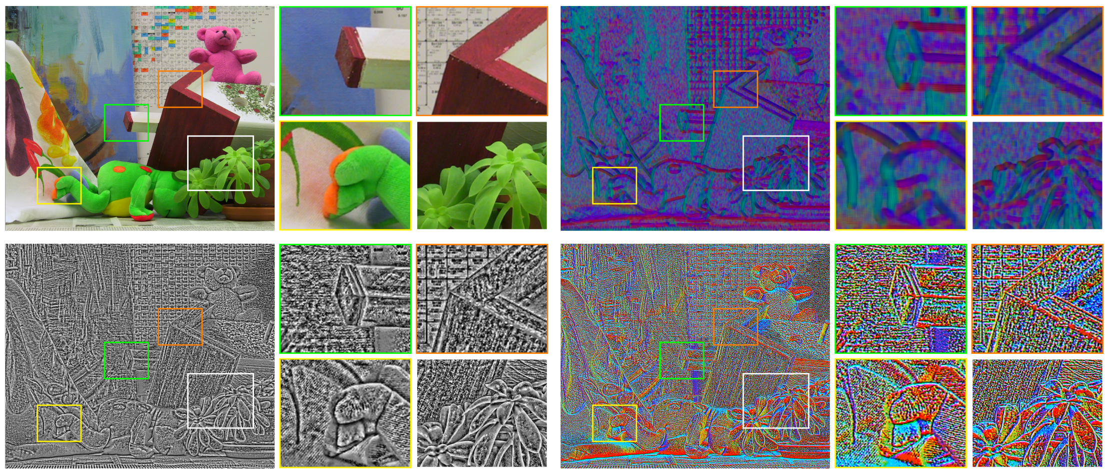


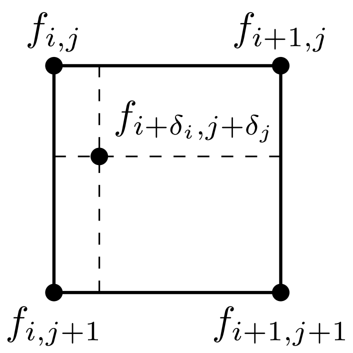
Figure 2.9: Visualization of different descriptors. **(a) Top Left:** The Teddy sequence from the Middlebury dataset [SS02]. **(b) Top Right:** Project the 128-D SIFT flow descriptor to 3-D subspace via PCA and visualize it in the RGB space. **(c) Bottom Left:** Rank transform with patch size 11×11. The rank value has been scaled to $[0, 255]$ for visualization. **(d) Bottom Right:** Census transform with patch size 5×5. Each census is saved in three bytes and is visualized in the RGB space.

2.6 The Coarse-to-fine Approach

Image pyramid [AAB+84] based coarse-to-fine approaches have gained much attention in motion estimation [Ana89; BA96; BBPW04; FBK15; SRB10] and stereo matching [Bar89; FH06; YWA10], since the displacement at coarser levels is usually very small, which fulfills the assumptions of many methods that require a small displacement for linearization. Furthermore, it can skip local minima during the estimation [BAHH92]. This section gives a detailed description of how to construct an image pyramid.

An image pyramid is constructed by down-sampling. For a pyramid with L levels, where level 0 is the raw image and level $L-1$ is the coarsest level, level l is obtained first by smoothing level $l-1$ with a Gaussian filter [GW17] to overcome the aliasing effect in the subsequent sampling and then the size of the filtered image is reduced by a factor of $\eta \in (0, 1)$ via some interpolation method, such as bilinear interpolation. Figure 2.10 illustrates the formula for bilinear interpolation. The procedure for constructing an image pyramid is described in Algorithm 1[3] and the image resize algorithm is given in Algorithm 2. An example pyramid of four levels with $\eta = 0.5$ constructed using Algorithm 1 is shown in Figure 2.11.

[3]It is modified from the source code provided by Ce Liu [Liu09] from the Massachusetts Institute of Technology. Please refer to https://people.csail.mit.edu/celiu/OpticalFlow/ for more information.



$$\begin{aligned} f_{i+\delta_i,j+\delta_j} &= (1-\delta_i)(1-\delta_j)f_{i,j} \\ &+ \delta_i(1-\delta_j)f_{i+1,j} \\ &+ (1-\delta_i)\delta_j f_{i,j+1} \\ &+ \delta_i\delta_j f_{i+1,j+1} \end{aligned}$$

Figure 2.10: Illustration of the bilinear interpolation method. Bilinear interpolation is used to obtain the pixel value at location $(i+\delta_i, j+\delta_j)$ with $\delta_i, \delta_j \in [0,1]$ given the four known pixel values at locations (i,j), $(i+1,j)$, $(i,j+1)$ and $(i+1,j+1)$.

Algorithm 1: Pyramid construction

```
input  : Raw image $\mathbf{f}$
input  : Number of levels $L$
input  : Pyramid ratio $\eta$
output : Images of different levels $\mathbf{f}^{L-1}, \mathbf{f}^{L-2}, \ldots, \mathbf{f}^{0}$

$\mathbf{f}^0 \longleftarrow \mathbf{f}$;                          /* The finest level is the raw image.  */
$\sigma_{base} \longleftarrow \frac{1}{\eta} - 1$;
$n \longleftarrow \frac{\log(0.25)}{\log(\eta)}$;          /* The result is truncated, i.e, $n$ is an integer.  */
$\sigma_n \longleftarrow \sigma_{base} \cdot n$;
for $l \leftarrow 1$ to $L-1$ do
    if $l \leq n$ then
        $\sigma \longleftarrow \sigma_{base} \cdot l$;              /* Sigma for the Gaussian filter.  */
        $\mathbf{g} \longleftarrow$ gaussianBlur($\mathbf{f}, \sigma$);
        $\mathbf{f}^l \longleftarrow$ resize ($\mathbf{g}$, $\eta^l$);                   /* Refer to Algorithm 2.  */
    else
        $\mathbf{g} \longleftarrow$ gaussianBlur($\mathbf{f}^{l-n}, \sigma_n$);
        $\mathbf{f}^l \longleftarrow$ resize ($\mathbf{g}$, $\eta^n$);
```

Algorithm 2: Resize (down-sample) an image using bilinear interpolation

```
input  : Image f
input  : Down-sample ratio $\eta$
output : Down-sampled image g

width $\longleftarrow$ f.width $\times\eta$;                              /* Image width of g.  */
height $\longleftarrow$ f.height $\times\eta$;                            /* Image height of g. */
for $j \leftarrow 0$ to height $- 1$ do
    $j' \longleftarrow \left\lfloor \frac{j}{\eta} \right\rfloor$;                          /* Take the integer part. */
    $\delta_j \longleftarrow \frac{j}{\eta} - j'$;                /* The fractional part in Figure 2.10. */
    for $i \leftarrow 0$ to width $- 1$ do
        $i' \longleftarrow \left\lfloor \frac{i}{\eta} \right\rfloor$;                      /* Take the integer part. */
        $\delta_i \longleftarrow \frac{i}{\eta} - i'$;            /* The fractional part in Figure 2.10. */
        $g_{i,j} \longleftarrow$ bilinearInterpolation ($\mathbf{f}$, $i'$, $j'$, $\delta_i$, $\delta_j$);
```

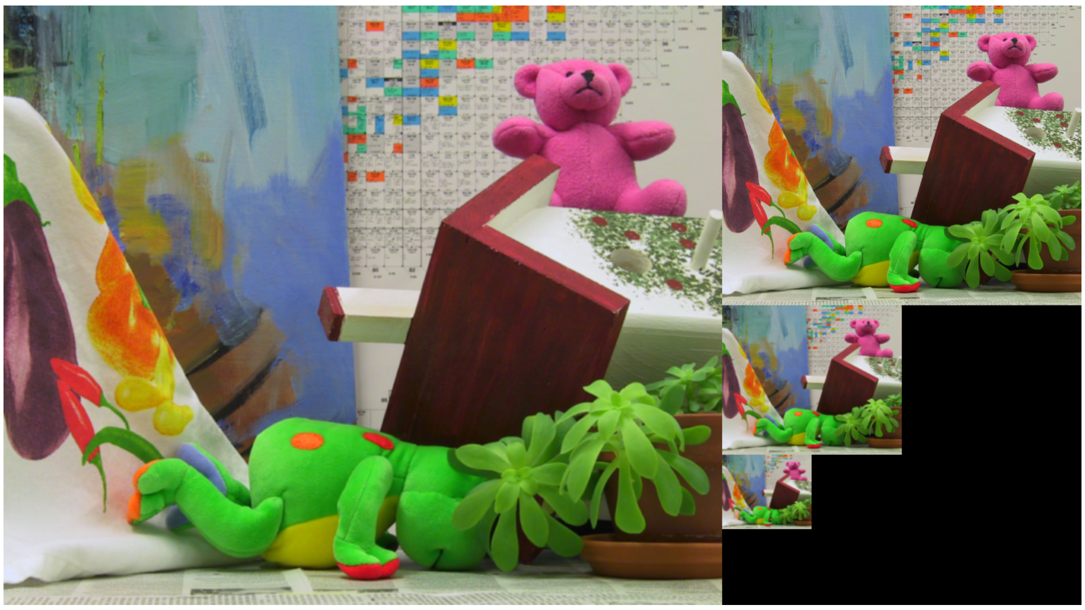


Figure 2.11: A pyramid of 4 levels with $\eta = 0.5$ constructed using Algorithm 1. Image source: The Teddy sequence from the Middlebury dataset [SS02].

Once the pyramid is constructed, the estimation is first performed at the coarsest level $L-1$, which is used to initialize level $L-2$. The process is repeated until the finest level is reached, where the final estimation is obtained. Figure 2.12 demonstrates this process. It should be noted that while using the estimated result from a coarser level, the estimation should be scaled appropriately. For example, if the pyramid ratio is η, then the estimated flow field is scaled by a factor of $\frac{1}{\eta}$ before it is used to initialize the finer level.

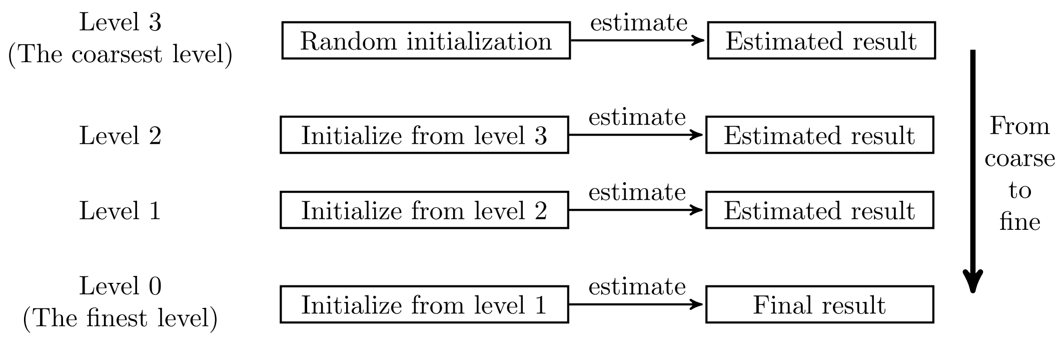


Figure 2.12: Demonstration of the coarse-to-fine approach with a pyramid of 4 levels.

3 The PatchMatch Algorithm

The PatchMatch algorithm, which was proposed by Barnes et al. [BSFG09], was originally designed for structured image editing. It is based on a randomized algorithm and aims to find approximate nearest neighbor matches. Different from the traditional methods of finding the exact nearest neighbor matches by searching the entire image, which is computationally challenging, PatchMatch tries to find only approximate nearest neighbors and can reduce the computation cost by a factor of 20–100, with a competitive performance [BSFG09]. This chapter gives a detailed introduction to the PatchMatch algorithm and describes how to extend it to model parameter estimation.

3.1 Assumptions of the PatchMatch Algorithm

The major assumptions used in PatchMatch are as follows:

- Adjacent pixels have similar model parameters $\mathbf{p} \in \mathbb{R}^n$.
- There is a cost function $\mathcal{C} : \mathbb{R}^n \to \mathbb{R}$ that computes the cost of a given $\mathbf{p}$.

For instance, in correspondence problems the displacement field $\mathbf{u}$ can be used as model parameters, i.e., model parameters for the pixel $f_{i,j}$ should be $\mathbf{p}_{i,j} = (u_{i,j}, v_{i,j})^\top$. Adjacent pixels would have similar motion patterns, thus their model parameters $\mathbf{p}$ should be similar.

3.2 The PatchMatch Algorithm

The PatchMatch algorithm consists of three components:

- Random Initialization
- Propagation
- Random Search

The following subsections are dedicated to these three components.

3.2.1 Random Initialization

Initialization is executed only once, where it initializes the model parameters of every pixel either randomly or based on some prior information. For example, in the coarse-to-fine approach, model parameters at the coarsest level would be assigned randomly, while a finer level would initialize its parameters from the coarser level that is one level above.

In the random initialization strategy, it assigns model parameters random values with uniform distribution in a given range and computes the corresponding costs. In optical flow estimation, for example, we can assign a random flow field to every pixel and compute the matching cost associated with the flow field.

The pseudocode for random initialization is given in Algorithm 3. It takes as input a pair of images and the range of model parameters, initializes the parameters of each pixel randomly, computes the costs associated with the assigned parameters and returns the initialized parameters and costs. Note that the random initialization step is highly parallelizable since every pixel is initialized independently.

Algorithm 3: Random initialization

```
input  : Two images $\mathbf{f}$ and $\mathbf{g}$
input  : Range of the parameters $\mathbf{r}$
output : Parameter $\mathbf{p}$ and the corresponding cost $\mathbf{c}$

foreach Pixel $f_{i,j}$ in $\mathbf{f}$ do
    $\mathbf{p_{i,j}} \longleftarrow$ uniform(-$\mathbf{r}$, $\mathbf{r}$);                    /* uniform distribution */
    $c_{i,j} \longleftarrow$ computeCost($\mathbf{p}_{i,j}$, $i$, $j$, $\mathbf{f}$, $\mathbf{g}$);  /* cost of the assigned parameter */
```

3.2.2 Iteration

The initialization step only assigns parameters to pixels and computes the corresponding costs. It is in the iteration step that costs are improved. The iteration step contains two sub steps: propagation and random search. During the propagation step, every pixel compares the costs computed using its own current model parameters and their neighbors'. If the cost computed from its owns is higher than that from its neighbors', the pixel replaces its own model parameters by its neighbors' since our goal is to minimize the matching cost and model parameters resulting a lower cost are preferable. Due to the propagation step, pixels with a correct guess of model parameters can propagate them to its neighbors, which are propagated further away, i.e., to the neighbors of neighbors.

The propagation step improves the cost of a pixel by integrating information from its neighborhood, while the random search step assigns a random sequence of parameters to a pixel and hopes that they may improve the cost, which are used in the subsequent propagation step. Therefore, in the area where no correct guesses of model parameters exist, it is still possible to obtain the correct guess through the random search step.

Propagation

In the propagation step, every pixel compares the costs of its own parameters with that of its neighbors and updates them if the parameters of its neighbors lead to a lower cost.

In terms of the definition of neighborhood, there are 4-connected neighborhood and 8-connected neighborhood, respectively [GW17].

$$\mathcal{N}_{i,j}^{4} = \Big\{(i, j-1),\ (i, j+1),\ (i-1, j),\ (i+1, j)\Big\} \tag{3.1}$$

denotes the 4-connected neighbors of $f_{i,j}$, while

$$\mathcal{N}_{i,j}^{8} = \Big\{(i,j-1),(i,j+1),(i-1,j),(i+1,j),(i+1,j-1),(i+1,j+1),(i-1,j-1),(i+1,j+1)\Big\} \tag{3.2}$$

denotes the 8-connected neighborhood of $f_{i,j}$. Let

$$\mathcal{N}_{i,j}^{4\ +} = \Big\{(i, j-1),\ (i-1, j)\Big\} \tag{3.3}$$

be the 4-connected neighborhood of $f_{i,j}$, which is scanned before $f_{i,j}$ when the image is scanned the in scan order, i.e., from top left to bottom right. Correspondingly, let

$$\mathcal{N}_{i,j}^{4\ -} = \Big\{(i, j+1),\ (i+1, j)\Big\} \tag{3.4}$$

be the 4-connected neighborhood of $f_{i,j}$, which is scanned before $f_{i,j}$ when the image is scanned in the reverse scan order, i.e., from bottom right to top left. In a similar manner, let us define

$$\mathcal{N}_{i,j}^{8\ +} = \Big\{(i, j-1),\ (i-1, j),\ (i+1, j-1),\ (i-1, j-1)\Big\} \tag{3.5}$$

and

$$\mathcal{N}_{i,j}^{8\ -} = \Big\{(i, j+1),\ (i+1, j),\ (i+1, j+1),\ (i-1, j+1)\Big\} \tag{3.6}$$

If it uses only the 4-connected neighborhood, let $\mathcal{N}_{i,j}^{+} := \mathcal{N}_{i,j}^{4\ +}$ and $\mathcal{N}_{i,j}^{-} := \mathcal{N}_{i,j}^{4\ -}$. And let $\mathcal{N}_{i,j}^{+} := \mathcal{N}_{i,j}^{8\ +}$ and $\mathcal{N}_{i,j}^{-} := \mathcal{N}_{i,j}^{8\ -}$ when it uses the 8-connected neighborhood.

There are two different kinds of propagations: forward propagation and backward propagation, which are executed in an interleaving order, i.e., first forward propagation, then backward, then forward, ... During the forward propagation, pixel $f_{i,j}$ compares with its neighbors $f_{i',j'}$, where $(i', j') \in \mathcal{N}_{i,j}^{+}$. While in the backward propagation step, it compares with the neighbors $f_{i',j'}$, where $(i', j') \in \mathcal{N}_{i,j}^{-}$.

Mathematically, in the forward propagation step, we have

$$\mathbf{p}_{i,j} = \underset{\substack{\mathbf{p}_{i',j'} \\ (i',j') \in \mathcal{N}_{i,j}^{+} \cup \{(i,j)\}}}{\arg\min} \mathcal{C}(\mathbf{p}_{i',j'}; i, j, \mathbf{f}, \mathbf{g}) \tag{3.7}$$

where $\mathbf{f}$ is the reference image and $\mathbf{g}$ is the corresponding image. And in the backward propagation step, it becomes

$$\mathbf{p}_{i,j} = \underset{\substack{\mathbf{p}_{i',j'} \\ (i',j') \in \mathcal{N}^{-}_{i,j} \cup \{(i,j)\}}}{\arg\min} \mathcal{C}(\mathbf{p}_{i',j'}; i, j, \mathbf{f}, \mathbf{g}) \tag{3.8}$$

Figure 3.1 (a)-(d) illustrates the forward and backward propagation steps of PatchMatch with 4-connected and 8-connected neighbors in the context of optical flow estimation and Figure 3.1 (e)-(h) shows the result of a random initialized flow field after two propagations (first forward and then backward propagation).

Note that propagation using 8-connected neighborhood is able to propagate more information per iteration (see Figure 3.1 (f) and (g)), which may converge faster than using the 4-connected neighborhood. However, propagation using 8-connected neighborhood requires two more cost computations for each pixel. If the cost computation procedure is computationally demanding, the time required per propagation step is much longer. In general, when the number of iterations is fixed, propagation using 8-connected neighborhood is able to propagate more information than using 4-connected neighborhood at the cost of consuming more time per propagation. In this thesis, we use 8-connected neighborhood during the propagation step.

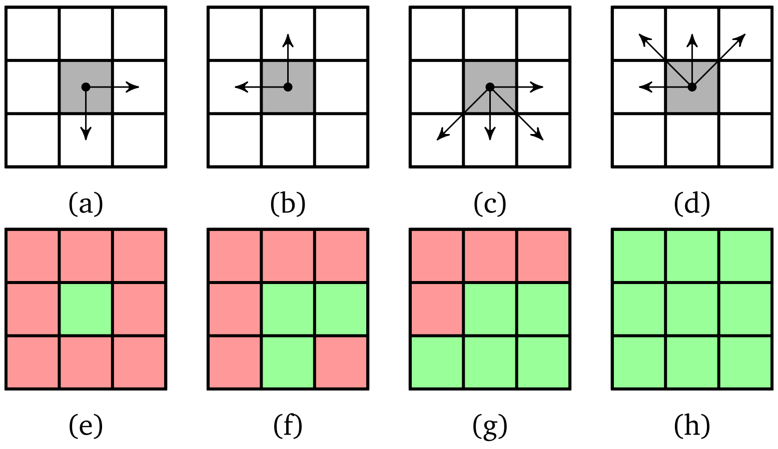


Figure 3.1: Illustration of the propagation steps in the context of optical flow estimation using flow field as the model parameter. (a) and (b) Forward and backward propagation directions using 4-connected neighbors. (c) and (d) Forward and backward propagation directions using 8-connected neighbors. (e) Flow field after random initialization. Pixels with red color indicate incorrect flow field, while green color means correct flow field. (f) Flow field of (e) after forward propagation using 4-connected neighbors (Note that it visualizes only a single propagation step of the green pixel without the propagations of propagated pixels.). (g) Flow field of (e) after forward propagation using 8-connected neighbors. (h) Flow field (f) and (g) after the backward propagation (Note that propagations of propagated pixels are used here.).

The pseudocode for the propagation step is given in Algorithm 4. It takes as input two images, the number of iterations, the value range of the parameters and the initialized parameters and costs from the initialization step. And then it performs either forward or backward propagation depending on the current iteration number to improve costs.

Algorithm 4: Propagation

```
input  : Two images $\mathbf{f}$, $\mathbf{g}$
input  : Number of iterations $k$
inout  : Parameter $\mathbf{p}$ and the corresponding cost $\mathbf{c}$

for $m \leftarrow 0$ to $k-1$ do
    /* when $m$ is even, it iterates in scan order,                */
    /* otherwise, in reverse scan order                             */
    foreach Pixel $f_{i,j}$ in $\mathbf{f}$ do
        if $m$ is even then                         /* forward propagation */
            $\mathcal{N}_{i,j} \longleftarrow \mathcal{N}^{+}_{i,j}$;
        else
            $\mathcal{N}_{i,j} \longleftarrow \mathcal{N}^{-}_{i,j}$;      /* backward propagation */
        foreach $(i', j') \in \mathcal{N}_{i,j}$ do
            $c \longleftarrow$ computeCost($\mathbf{p}_{i',j'}$, $i$, $j$, $\mathbf{f}$, $\mathbf{g}$);
            if $c < c_{i,j}$ then                      /* $\mathbf{p}_{i',j'}$ has a lower cost */
                $\mathbf{p}_{i,j} \longleftarrow \mathbf{p}_{i',j'}$;        /* update the current parameter */
                $c_{i,j} \longleftarrow c$;                 /* update the current cost */
        randomSearch();                               /* See Algorithm 5.  */
```

Random Search

The random search step is an essential step to make PatchMatch work. It introduces new information into the algorithm by selecting a sequence of new parameters randomly from an interval, whose length decreases exponentially with respect to the number of searches. For the parameter $\mathbf{p}_{i,j}$ of the pixel $f_{i,j}$, it selects a new parameter using Equation (3.9):

$$\mathbf{p}^k_{i,j} = \mathbf{p}_{i,j} + \mathbf{w}\alpha^k \mathbf{R}_k \tag{3.9}$$

where $\mathbf{w} \in \mathbb{R}^n$ is the maximum search interval, α is a fixed positive value less than 1, $\mathbf{R}_k \in \mathbb{R}^n$ is a random variable taking values from $[-1, 1]$ and $k = 0, 1, 2, \ldots$ until the search interval is less than a given value. Note that we use element-wise product here between $\mathbf{w}$ and $\mathbf{R}_k$. Let $\mathcal{P}_{i,j} := \{\mathbf{p}^0_{i,j}, \mathbf{p}^1_{i,j}, \mathbf{p}^2_{i,j}, \ldots\}$, it updates the current parameters and costs using Equation (3.10):

$$\mathbf{p}_{i,j} = \underset{\substack{\mathbf{p}_{i',j'} \\ \in \mathcal{P}_{i,j} \cup \{\mathbf{p}_{i,j}\}}}{\arg\min} \; \mathcal{C}(\mathbf{p}_{i',j'}; i, j, \mathbf{f}, \mathbf{g}) \tag{3.10}$$

Algorithm 5 shows the pseudocode for the random search step.

Algorithm 5: Random search

```
input  : Two images $\mathbf{f}$, $\mathbf{g}$
input  : Maximum search interval $\mathbf{w}$, fixed ratio $\alpha$
input  : Stop search ratio $\mathbf{s}$
input  : Pixel position $(i, j)$
inout  : Parameter $\mathbf{p}_{i,j}$ and the corresponding cost $c_{i,j}$

$k \leftarrow 0$;
while true do
    $\mathbf{R} \longleftarrow$ uniform(-1,1);
    $\mathbf{\Delta} \longleftarrow \mathbf{w}\alpha^k\mathbf{R}$;
    if $\mathbf{\Delta} \prec \mathbf{s}$ then
        break;
    $\mathbf{p}' \longleftarrow \mathbf{p}_{i,j} + \mathbf{\Delta}$;
    $c \longleftarrow$ computeCost($\mathbf{p}'$, $i$, $j$, $\mathbf{f}$, $\mathbf{g}$);
    if $c < c_{i,j}$ then                        /* $\mathbf{p}'$ has a lower cost */
        $\mathbf{p}_{i,j} \longleftarrow \mathbf{p}'$;           /* update the current parameter */
        $c_{i,j} \longleftarrow c$;                   /* update the current cost */
```

Different from the random initialization step, the iteration step is highly sequential and it is difficult to parallelize it.

Although PatchMatch is only able to find approximate nearest neighbors, Barnes et al. [BSFG09] has proved that it converges to the exact nearest neighbor in the limit, i.e., in infinite running time. According to our implementation, we find that the percentage of pixels replacing its own parameters by its neighbors' is less than 5% in just a few iterations.

4 PatchMatch for Motion Estimation

This chapter starts with the motion estimation pipeline in Section 4.1 followed by the introduction of commonly used motion models in Section 4.2. Section 4.3 presents the cost functions used by different image descriptors. In addition, the propagation rules of the estimated model parameters from a coarse level to a fine level in the context of coarse-to-fine approaches are discussed in Section 4.4.

4.1 The Motion Estimation Pipeline

The motion estimation pipeline used in this thesis is shown in Figure 4.1. It contains two parts, flow field estimation and flow field post-processing. Many methods presented in the recent literature adopt this pipeline. EpicFlow [RWHS15] (Edge-Preserving Interpolation of Correspondences for Optical Flow) takes as input a sparse initial flow field estimated by DeepFlow [WRHS13] and then interpolates it into a dense flow. FlowFields [BTS15] and CPM-Flow [HSL16] replace the estimation method of the initial flow field of EpicFlow, while Maurer et al. [MSB17] improve the variational refinement step used in EpicFlow. In the meanwhile, RicFlow [HLS17] replaces the interpolation method of EpicFlow. Figure 4.2 visualizes the motion estimation pipeline using the Yosemite sequence with clouds[1] with a pyramid of three levels[2].

4.1.1 Coarse-to-fine PatchMatch

The coarse-to-fine PatchMatch takes as input two images, builds an image pyramid (see Section 2.6), computes the image descriptor (see Section 2.5) of each level in the pyramid, runs the PatchMatch algorithm (See Chapter 3) against each level and outputs a sparse flow field with outliers removed. In the initialization step of the PatchMatch algorithm, the flow field at the coarsest level is initialized randomly, while other levels are initialized from the estimated result one level above (the flow field is scaled appropriately before initialization). The outlier removal step is discussed in the following.

Since some pixels that are visible in one image may become occluded in the other image, we have to discard their estimations. One such method is the so-called forward-backward

[1]The sequence is available at the address http://www.informatik.uni-ulm.de/ni/staff/PBayerl/homepage/animations/index.html, which was created by Lynn Quam at SRI.

[2]Parameter settings: pyramid levels (3), pyramid ratio (0.75), step size (1), number of iterations (8).

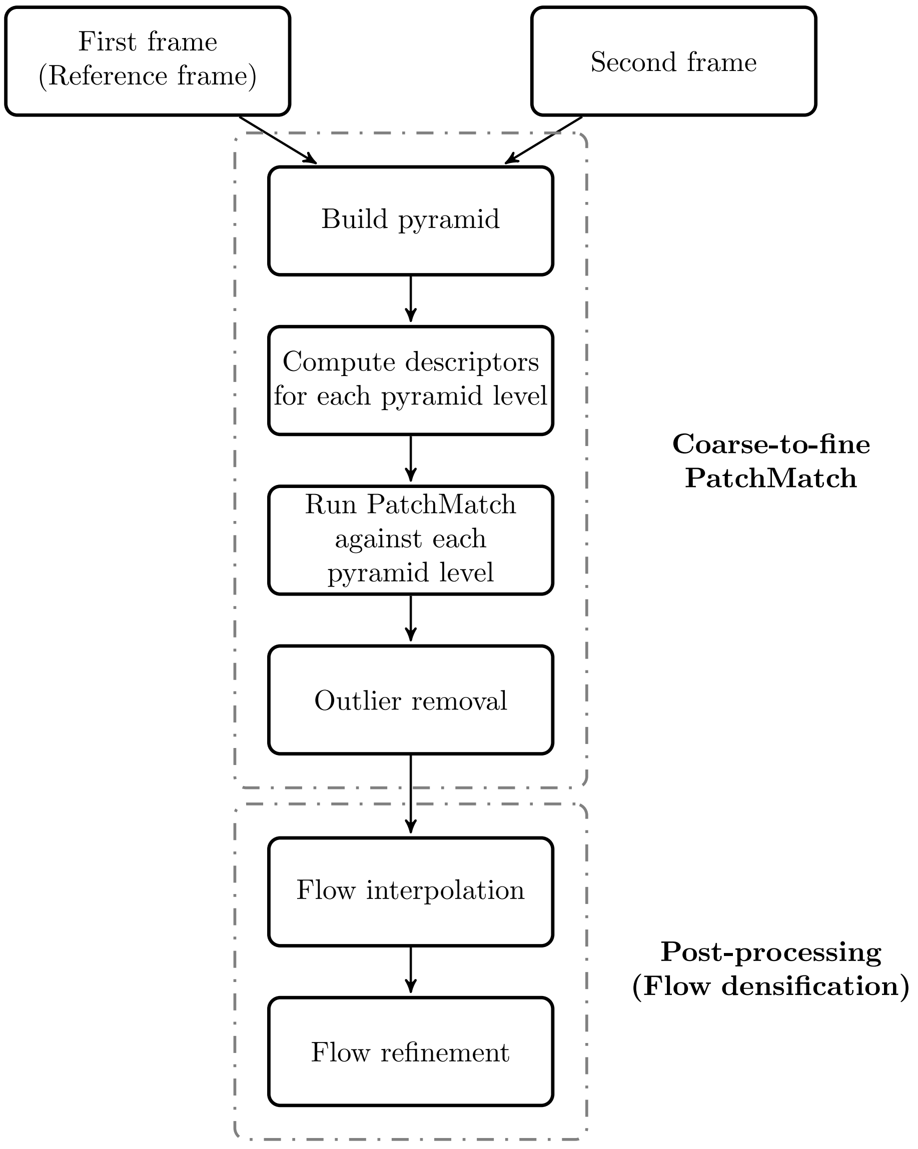

First frame
(Reference frame)
Second frame
Build pyramid
Compute descriptors for each pyramid level
Run PatchMatch against each pyramid level
Outlier removal
Coarse-to-fine PatchMatch
Flow interpolation
Flow refinement
Post-processing (Flow densification)

Figure 4.1: Motion Estimation Pipeline.

consistency check [BTS15; HSL16], which works as follows. First, we compute a forward flow field $\mathbf{u}^{\text{fw}}$ from image $\mathbf{f}$ to $\mathbf{g}$ and a backward flow field $\mathbf{u}^{\text{bw}}$ from image $\mathbf{g}$ to $\mathbf{f}$. And then evaluate the following expression

$$|\Delta \mathbf{u}_{i,j}| = \sqrt{\left(u^{\text{fw}}_{i,j} + u^{\text{bw}}_{i+u^{\text{fw}}_{i,j},j+v^{\text{fw}}_{i,j}}\right)^2 + \left(v^{\text{fw}}_{i,j} + v^{\text{bw}}_{i+u^{\text{fw}}_{i,j},j+v^{\text{fw}}_{i,j}}\right)^2} \tag{4.1}$$

In the ideal case, $|\Delta \mathbf{u}_{i,j}|$ should be 0. In practice, a threshold T_{occ} is used. When $|\Delta \mathbf{u}_{i,j}|$ is larger than T_{occ}, the estimation $\mathbf{u}_{i,j}$ is discarded. In this thesis, we choose $T_{\text{occ}} = 3$.

In addition, the displacement cannot be arbitrarily large. We discard the estimation $\mathbf{u}_{i,j}$ when its magnitude is larger than 400 pixels in this thesis, which is the same value used

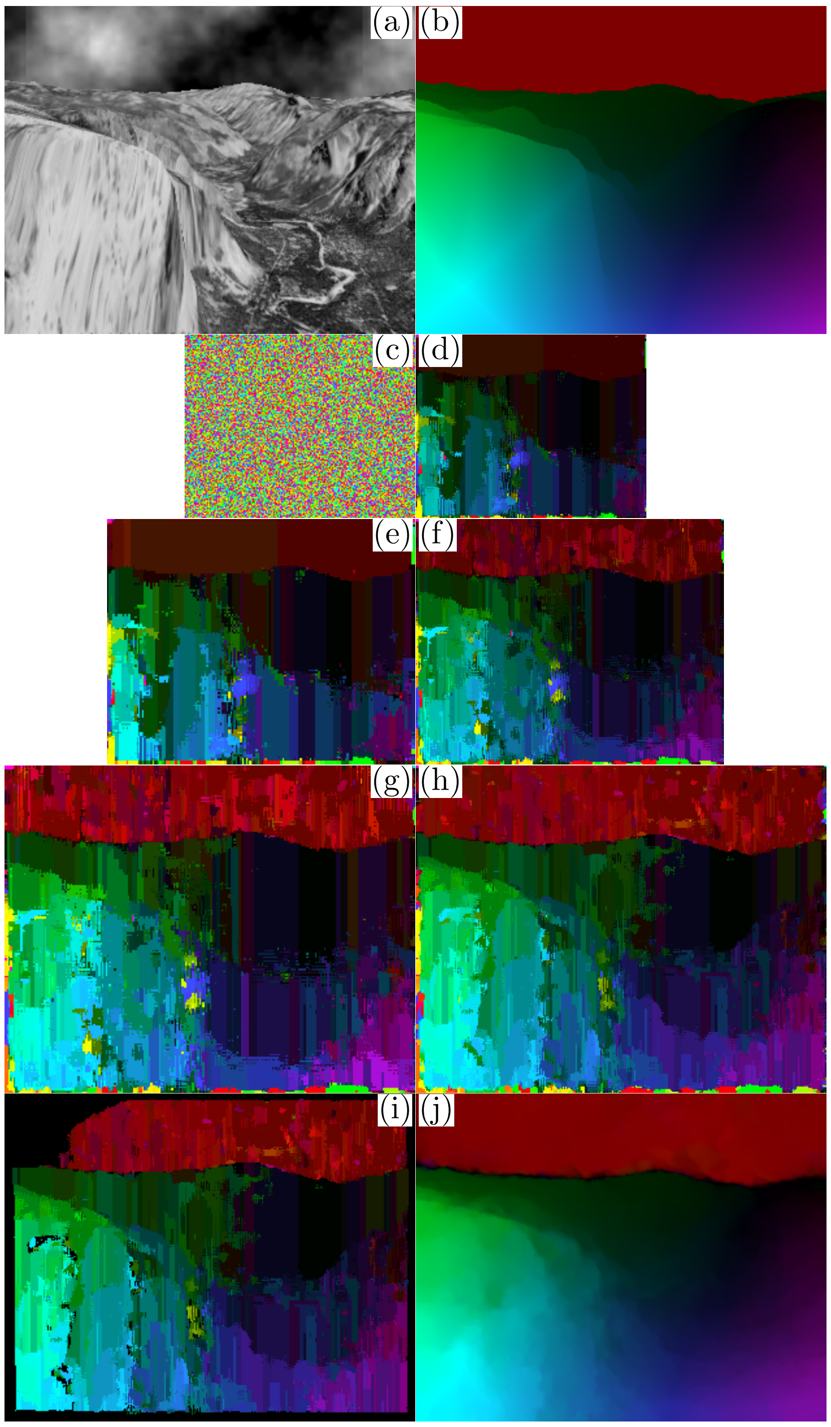


Figure 4.2: Illustration of the motion estimation pipeline with a pyramid of 3 levels using the Yosemite sequence with clouds. (a) Frame 7. (b) Ground truth. (c) Flow field of level 2 after random initialization (the coarsest level). (d) Estimated flow field of level 2. (e) Flow field of level 1 initialized from (d). (f) Estimated flow field of level 1. (g) Flow field of level 0 initialized from (f). (h) Estimated flow field of level 0. (i) Flow field after outlier removal. (j) Flow field after post-processing (flow interpolation + flow refinement).

by Hu et al. [HSL16]. Of course, T_{occ} depends on image sizes and applications, one may choose another value that suits best to the specific application.

4.1.2 Post-Processing

The post-processing step is responsible to fill in the locations where no flow fields are available, e.g., locations that are considered as outliers. After filling in the missing flow fields, variational methods [ZBW11] are applied to refine and smooth the flow field. Figure 4.2 (j) gives an example of the flow field after post-processing.

One of the state-of-the-art methods for post-processing is the edge preserving interpolation algorithm (EpicFlow) proposed by Revaud et al. [RWHS15], whose source code is available at the address https://thoth.inrialpes.fr/src/epicflow/. This thesis focuses on computing the initial flow field using the PatchMatch algorithm. The post-processing step from the EpicFlow algorithm[3] is used to post-process the estimated flow field.

While using EpicFlow for post-processing, a common technique is to make the initial flow field sparse. Although the initial flow field after outlier removal is already sparse, it is not sparse enough. For instance, Bailer et al. [BTS15] divide the initial flow field after outlier removal into 3×3 regions, select only one match from each region and feed only the selected matches to EpicFlow, while Hu et al. [HSL16] estimate the initial flow field only at specified locations, which are known as seeds. The distance between adjacent seeds is called step size, which is denoted as S. Figure 4.3 shows an example with $S = 3$. This thesis follows the similar approach as Hu et al. [HSL16].

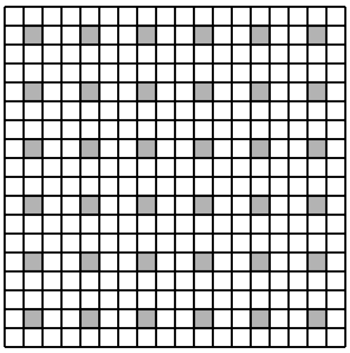


Figure 4.3: Flow field estimation is only performed at locations marked with gray color. These locations are called seeds, whose step size S is 3 in this example.

There are two advantages to use a sparse flow field as input for EpicFlow. On the one hand, EpicFlow consumes less time when the flow field is sparse. The less the input matches are, the less time EpicFlow requires. On the other hand, a sparse flow field requires less time to

[3]Note that there are two types of interpolation methods in EpicFlow, we use the LA (Locally-weighted affine) method for interpolation. Parameters uses default values provided by the authors.

Step size	1	2	3	4	5	6
# matches after outlier removal	70786	18063	7744	4446	2787	1991
Estimation time (s)	7.991	2.868	1.721	1.124	0.776	0.561
Post-processing time (s)	5.158	1.082	0.791	0.709	0.652	0.634
AAE	0.125	0.115	0.117	0.113	0.114	0.109

Table 4.1: The relationships among the step size, the processing time and AEE. Settings: Input images (Frame 7 and 8 of the Yosemite sequence with clouds. Raw image size: 316×252), number of pyramid levels (4), pyramid ratio (0.75), number of iterations (8). Environment: C++, 6 CPU cores @ 2.5GHz.

estimate. In other words, a sparse flow field decreases not only the estimation time but also the post-processing time. However, the flow field cannot be too sparse, otherwise the performance may be degraded. One has to find a balance between the time consumption and the required performance according to his/her own needs. Table 4.1 illustrates the relationships among the step size, the processing time and the AEE[4] using the Yosemite sequence with clouds. It shows that the processing time is indeed decreased when the step size is increased. From $S = 1$ to $S = 2$, the total processing time drops from 13.149 seconds to 3.950 seconds and the AEE is also improved, while from $S = 5$ to $S = 6$ the total time consumed and the final AEE do not change too much. Figure 4.4 visualizes the result after post-processing the initial flow field with $S = 6$.

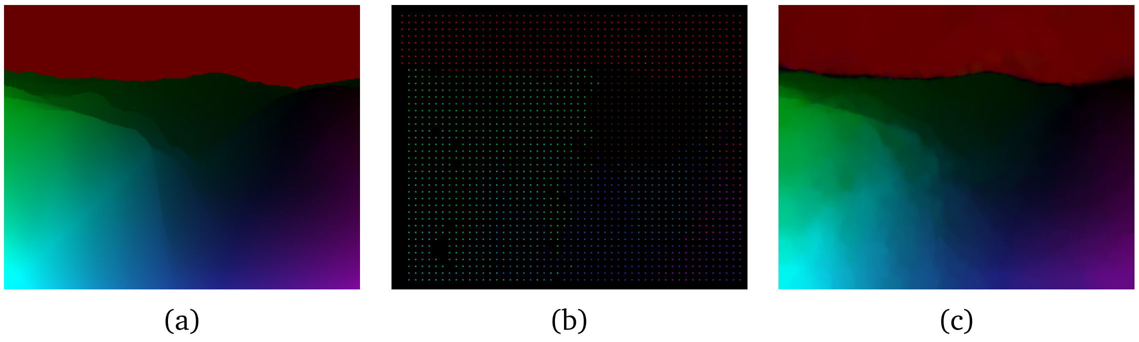


Figure 4.4: Estimation of the Yosemite sequence with clouds (Frame 7). (a) Ground truth. (b) Initial flow field estimated using PatchMatch with $S = 6$ (with outlier removed). (c) After post-processing of (b) using EpicFlow.

Since one of the goals of this thesis is to estimate model parameters, in the following section we will discuss the parametric models that are commonly used.

[4]Please refer to Section 6.1 for the definition of AEE (Average Endpoint Error), which is used to evaluate the estimated flow field. The smaller the AEE, the better the estimation.

4.2 Motion Models

In this section, we present four commonly used motion models in the literature: the translational motion model (Section 4.2.1), the affine motion model (Section 4.2.2), the projective motion model (Section 4.2.3) and the quadratic motion model (Section 4.2.4).

Given two images $\mathbf{f}$ and $\mathbf{g}$, a motion model tells us how a point $\mathbf{x}_1 = (x_1, y_1)^\top$ in the image $\mathbf{f}$ transforms to a new location $\mathbf{x}_2 = (x_2, y_2)^\top$ in the image $\mathbf{g}$. In addition, the motion field is defined as $\mathbf{u} = (u, v)^\top = (x_2 - x_1, y_2 - y_1)^\top$. In general, a motion model can be expressed as

$$\begin{aligned} x_2 &= \mathcal{M}_1(\mathbf{p}; x_1, y_1) \\ y_2 &= \mathcal{M}_2(\mathbf{p}; x_1, y_1) \end{aligned} \tag{4.2}$$

where $\mathbf{p} \in \mathbb{R}^n$ is the model parameter.

4.2.1 The Translational Motion Model

The translational motion model assumes there are only translations between the given images and is given by [GGN13]

$$\begin{aligned} x_2 &= x_1 + b_1 \\ y_2 &= y_1 + b_2 \end{aligned} \tag{4.3}$$

while the motion field is defined as

$$\begin{aligned} u &= b_1 \\ v &= b_2 \end{aligned} \tag{4.4}$$

It has only two parameters and is the simplest motion model. We can observe such a motion when, for example, a 3-D object undergoing rigid translation is projected to the image plane via orthographic projection [SK99].

4.2.2 The Affine Motion Model

The affine motion model has six parameters and is given by [BAHH92; GGN13; SK99]

$$\begin{aligned} x_2 &= a_1 x_1 + a_2 y_1 + b_1 \\ y_2 &= a_3 x_1 + a_4 y_1 + b_2 \end{aligned} \tag{4.5}$$

When $a_1 = a_4 = 1$ and $a_2 = a_3 = 0$, we get the translational motion model. The affine motion model can also represent the rotation

$$\begin{aligned} x_2 &= x_1 \cos\theta - y_1 \sin\theta \\ y_2 &= x_1 \sin\theta + y_1 \cos\theta \end{aligned} \tag{4.6}$$

the scaling

$$\begin{aligned} x_2 &= a_1 x_1 \\ y_2 &= a_4 y_1 \end{aligned} \tag{4.7}$$

and the shear transformation

$$\begin{aligned} x_2 &= x_1 + a_2 y_1 \\ y_2 &= a_3 x_1 + y_1 \end{aligned} \tag{4.8}$$

4.2.3 The Projective Motion Model

In the projective motion model [CHY10; MP97; THZ82; YL15], it assumes that there is a 2-D planar homograhpy $\mathbf{H}$

$$\mathbf{H} = \begin{pmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{pmatrix} \tag{4.9}$$

such that

$$\omega\,\tilde{\mathbf{x}}_2 = \mathbf{H}\,\tilde{\mathbf{x}}_1 \tag{4.10}$$

where $\tilde{\mathbf{x}}_1 = (x_1, y_1, 1)^\top$ and $\tilde{\mathbf{x}}_2 = (x_2, y_2, 1)^\top$ are the homogeneous coordinate representations of $\mathbf{x}_1$ and $\mathbf{x}_2$, respectively. $\omega \in \mathbb{R}$ is a real number. Thus, we get

$$\omega \begin{pmatrix} x_2 \\ y_2 \\ 1 \end{pmatrix} = \begin{pmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \\ 1 \end{pmatrix} \tag{4.11}$$

which can be further simplified to

$$\begin{aligned} x_2 &= \frac{h_1 x_1 + h_2 y_1 + h_3}{h_7 x_1 + h_8 y_1 + h_9} \\ y_2 &= \frac{h_4 x_1 + h_5 y_1 + h_6}{h_7 x_1 + h_8 y_1 + h_9} \end{aligned} \tag{4.12}$$

Because $\mathbf{H}$ is defined only up to a scale, it has eight degrees of freedom (DoF). As pointed out by Hartley and Zisserman [HZ03], it is not a good idea to constrain one of the elements of $\mathbf{H}$ to be 1. For example, if the true value of h_9 is 0 but it is fixed to be 1 during the estimation, then there is no way to obtain the true solution.

4.2.4 The Quadratic Motion Model

The quadratic motion model is defined as [BAHH92]

$$\begin{aligned} x_2 - x_1 &= a_1 + a_2 x_1 + a_3 y_1 + a_7 x_1^2 + a_8 x_1 y_1 \\ y_2 - y_1 &= a_4 + a_5 x_1 + a_6 y_1 + a_7 x_1 y_1 + a_8 y_1^2 \end{aligned} \tag{4.13}$$

It has eight parameters and is linear with respect to the parameters. Table 4.2 lists the motion models that are used in this thesis and the parameters to be estimated of each motion model are summarized in Table 4.3.

Motion model	**Number of parameters**	**Transformation rule**
Translational	2	$x_2 = x_1 + b_1$ $y_2 = y_1 + b_2$
Affine	6	$x_2 = a_1x_1 + a_2y_1 + b_1$ $y_2 = a_3x_1 + a_4y_1 + b_2$
Projective	9	$x_2 = \dfrac{h_1x_1 + h_2y_1 + h_3}{h_7x_1 + h_8y_1 + h_9}$ $y_2 = \dfrac{h_4x_1 + h_5y_1 + h_6}{h_7x_1 + h_8y_1 + h_9}$
Quadratic	8	$x_2 - x_1 = a_1 + a_2x_1 + a_3y_1 + a_7x_1^2 + a_8x_1y_1$ $y_2 - y_1 = a_4 + a_5x_1 + a_6y_1 + a_7x_1y_1 + a_8y_1^2$

Table 4.2: Motion models. This table summarizes the motion models used in this thesis.

Motion model	**Model parameters**
Translational	$\mathbf{p} = (b_1, b_2)^\top \in \mathbb{R}^2$
Affine	$\mathbf{p} = (a_1, a_2, a_3, a_4, b_1, b_2)^\top \in \mathbb{R}^6$
Projective	$\mathbf{p} = (h_1, h_2, h_3, h_4, h_5, h_6, h_7, h_8, h_9)^\top \in \mathbb{R}^9$
Quadratic	$\mathbf{p} = (a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8)^\top \in \mathbb{R}^8$

Table 4.3: Prameters to be estimated of each motion model.

4.3 Cost Functions

While using the PatchMatch algorithm, we have to identify the parameters to be estimated and the cost function for evaluating a given set of parameters. Since the parameters to be estimated have been given in Table 4.3, this section focuses on the cost functions.

Once we have obtained the model parameters, it is straightforward to get the new location (x_2, y_2) corresponding to the initial location (x_1, y_1) using Equation (4.2).

Similar to the matching cost used by the block matching method presented in Section 2.4, we use the brightness similarity as the cost function. However, instead of using the raw images, we compute the cost between their descriptor images (see Section 2.5).

For the SIFT flow descriptor (Section 2.5.1), the rank transform (Section 2.5.2) and the complete rank transform (Section 2.5.4) we use the L1-norm to compute the matching cost, which is defined as

$$\mathcal{C}(\mathbf{p}; x_1, y_1) = \sum_{\delta_x=-m}^{m} \sum_{\delta_y=-m}^{m} \left\| \mathbf{F}(x_1 + \delta_x, y_1 + \delta_y) - \mathbf{G}(x_2 + \delta_x, y_2 + \delta_y) \right\|_1 \tag{4.14}$$

where m is the radius of the patch $(2m+1) \times (2m+1)$ centered at location (x_1, y_1), $\mathbf{F}$ and $\mathbf{G}$ are the descriptor images of image $\mathbf{f}$ and image $\mathbf{g}$, respectively. $\mathbf{F}(x_1, y_1) \in \mathbb{R}^k$ and $\mathbf{G}(x_2, y_2) \in \mathbb{R}^k$ represent the descriptor value with dimension k at locations (x_1, y_1) and (x_2, y_2) in the corresponding descriptor images, respectively.

For the census transform (Section 2.5.3) and the complete census transform (Section 2.5.5), we use the same cost function as used by Demetz [Dem15], which is the Hamming distance defined as

$$\mathcal{C}(\mathbf{p}; x_1, y_1) = \sum_{\delta_x=-m}^{m} \sum_{\delta_y=-m}^{m} \sum_{t=0}^{k-1} \mathbb{1}_{(\mathbf{F}(x+\delta_x, y+\delta_y)_t \neq \mathbf{G}(x_2+\delta_x, y_2+\delta_y)_t)} \tag{4.15}$$

where $\mathbb{1}_{(\cdot)}$ is the indicator function defined in Equation (2.11). $\mathbf{F}(x+\delta_x, y+\delta_y)_t$ and $\mathbf{G}(x_2+\delta_x, y_2+\delta_y)_t$ indicate the t-th element in $\mathbf{F}(x+\delta_x, y+\delta_y)$ and $\mathbf{G}(x_2+\delta_x, y_2+\delta_y)$, respectively. Table 4.4 lists the patch sizes used by different descriptors in this thesis. Pay attention to the differences between the patch sizes listed in this table and the one in Table 2.1. The values in Table 2.1 are used for computing descriptors, while patch sizes listed in Table 4.4 are used for computing costs.

Descriptor type	**Patch size**
The SIFT flow descriptor	1×1
The rank transform	17×17
The census transform	17×17
The complete rank transform	17×17
The complete census transform	17×17

Table 4.4: The patch sizes used by different descriptors in Equation (4.14) and Equation (4.15) in this thesis.

4.4 The Coarse-to-fine Approach

In the coarse-to-fine approach, we first estimate the parameters at a coarse level and then use the estimated results to initialize the fine level. The question is how to propagate the estimated results from the coarse level to the fine level. In other words, if we have the motion model at level l

$$\mathbf{x}_2^l = \mathcal{M}(\mathbf{p}^l; \mathbf{x}_1^l) \tag{4.16}$$

and the model at level $l-1$

$$\mathbf{x}_2^{l-1} = \mathcal{M}(\mathbf{p}^{l-1}; \mathbf{x}_1^{l-1}) \tag{4.17}$$

what is the relationship between $\mathbf{p}^l$ and $\mathbf{p}^{l-1}$? We will discuss this problem in the following.

Let $\eta \in (0, 1)$ be the image size ratio between two consecutive levels, then we have $\mathbf{x}_1^l = \eta \mathbf{x}_1^{l-1}$ and $\mathbf{x}_2^l = \eta \mathbf{x}_2^{l-1}$. Thus for the translational model

$$\begin{aligned} x_2^l &= x_1^l + b_1^l \\ y_2^l &= y_1^l + b_2^l \end{aligned} \tag{4.18}$$

we have

$$\begin{aligned} \eta x_2^{l-1} &= \eta x_1^{l-1} + b_1^l \\ \eta y_2^{l-1} &= \eta y_1^{l-1} + b_2^l \end{aligned} \tag{4.19}$$

which can be transformed to

$$\begin{aligned} x_2^{l-1} &= x_1^{l-1} + \frac{b_1^l}{\eta} \\ y_2^{l-1} &= y_1^{l-1} + \frac{b_2^l}{\eta} \end{aligned} \tag{4.20}$$

Therefore, we obtain

$$\begin{aligned} b_1^{l-1} &= \frac{b_1^l}{\eta} \\ b_2^{l-1} &= \frac{b_2^l}{\eta} \end{aligned} \tag{4.21}$$

The same procedure can be applied to other motion models. Table 4.5 summaries the propagation rules of the estimated results between two consecutive levels for different motion models.

Motion model	**Propagation rules**
Translational	$b_1^{l-1} = \frac{b_1^l}{\eta},\ b_2^{l-1} = \frac{b_2^l}{\eta}$
Affine	$a_1^{l-1} = a_1^l,\ a_2^{l-1} = a_2^l,\ b_1^{l-1} = \frac{b_1^l}{\eta},$ $a_3^{l-1} = a_3^l,\ a_4^{l-1} = a_4^l,\ b_2^{l-1} = \frac{b_2^l}{\eta}$
Projective	$h_1^{l-1} = h_1^l,\quad h_2^{l-1} = h_2^l,\quad h_3^{l-1} = \frac{h_3^l}{\eta},$ $h_4^{l-1} = h_4^l,\quad h_5^{l-1} = h_5^l,\quad h_6^{l-1} = \frac{h_6^l}{\eta},$ $h_7^{l-1} = \eta h_7^l,\ h_8^{l-1} = \eta h_8^l,\ h_9^{l-1} = h_9^l$
Quadratic	$a_1^{l-1} = \frac{a_1^l}{\eta},\ a_2^{l-1} = a_2^l,\ a_3^{l-1} = a_3^l,\ a_7^{l-1} = \eta a_7^l,$ $a_4^{l-1} = \frac{a_4^l}{\eta},\ a_5^{l-1} = a_5^l,\ a_6^{l-1} = a_6^l,\ a_8^{l-1} = \eta a_8^l$

Table 4.5: Propagation rules of the estimated results from level l to level $l-1$.

5 PatchMatch for Stereo Reconstruction

This chapter begins with a brief introduction to the stereo reconstruction pipeline in Section 5.1. Section 5.2 describes three models for stereo matching: the epipolar constraint model (Section 5.2.1), the projective planar model (Section 5.2.2) and the slanted plane model (Section 5.2.3). The cost functions for evaluating model parameters are given in Section 5.3. In addition to the propagation step presented in Chapter 3, which is known as spatial propagation, a new kind of propagation, the so-called view propagation, is introduced in Section 5.4, which was first presented by Bleyer et al. [BRR11]. Section 5.5 concludes this chapter with the post-processing step for stereo matching.

5.1 The Stereo Reconstruction Pipeline

There are in general four steps in stereo reconstruction, which are camera calibration [Zha00], image rectification [FTV00], disparity estimation (also known as stereo matching) and depth reconstruction. Camera calibration is to obtain the intrinsic (e.g., the focal length) and extrinsic (e.g., the position and orientation) parameters of a camera, while image rectification simplifies the task of stereo matching, which needs only to perform searches horizontally [HZ03]. With known camera calibration and disparity, the depth can be obtained via triangulation, which is simply a technical problem.

One of the most common techniques for camera calibration is to take some images of a known object (a so-called calibration rig, such as a checkerboard) from different viewpoints, identify correspondences between the images and the object by extracted corners and estimate the camera parameters by minimizing an energy function via some optimization algorithms [MSKS12]. Some widely used tools for camera calibration are the Matlab camera calibration toolbox [Bou] and the OpenCV library [Bra00].

After camera calibration, image rectification is performed so that there exists only horizontal displacement [FTV00]. An example is given in Figure 5.1. Before rectification, there are both horizontal and vertical displacements (Figure 5.1 (a) and (b)), while after rectification, there is only horizontal displacement (Figure 5.1 (c) and (d)).

Figure 5.2 illustrates how to reconstruct the depth via triangulation once we know the disparity and the camera parameters. By similar triangles, it is easy to obtain

$$Z = \frac{f \cdot b}{x_1 - x_2},\ Y = \frac{y_1 \cdot Z}{f},\ X = \frac{x_1 \cdot Z}{f} \tag{5.1}$$

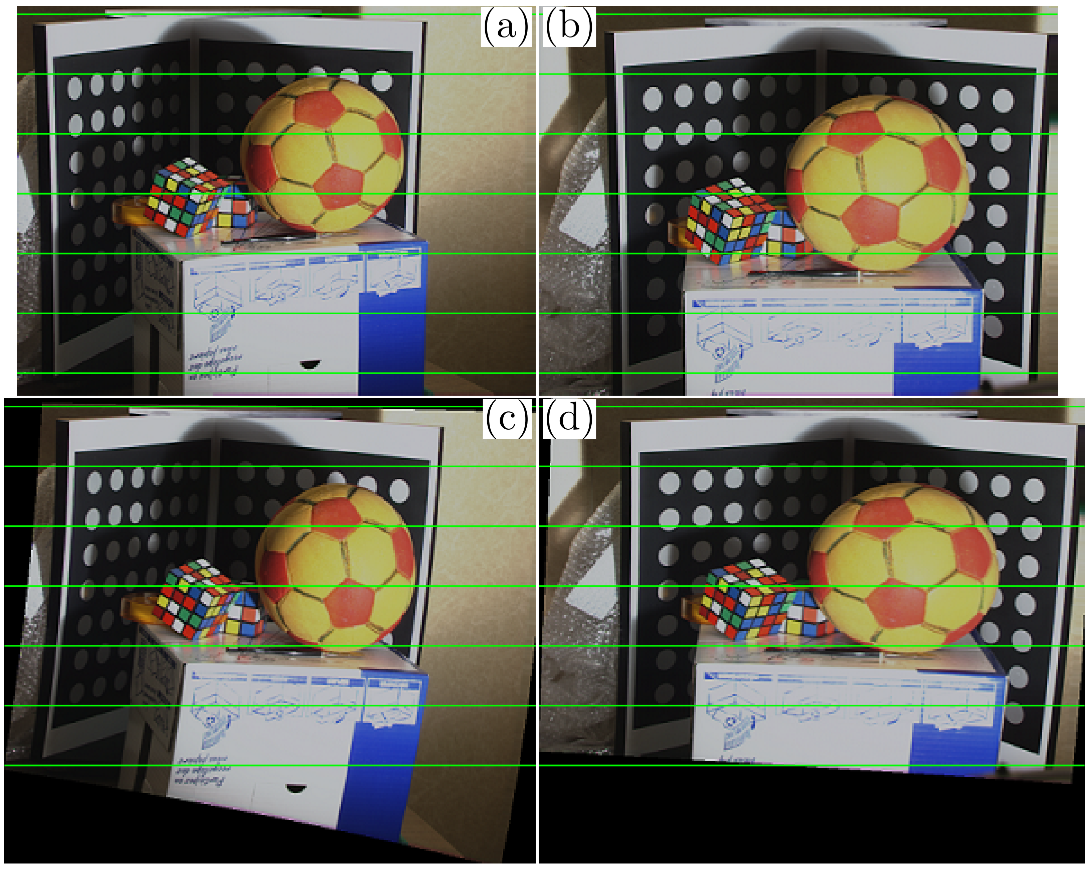


Figure 5.1: Illustration of the image rectification. Green lines are added for visualization. (a) and (b) Unrectified left and right view . (c) and (d) Rectified left and right view. Image source: (a) and (b) are from the public SYNTIM dataset, which is available at the address http://perso.lcpc.fr/tarel.jean-philippe/syntim/paires.html. (c) and (d) are from [ZDL17].

where $d = x_1 - x_2$ is the disparity computed by stereo matching;[1] the focal length f and the baseline b are obtained via camera calibration. Note that when the images are rectified, the disparity d is always non-negative [CS11], which can be used as an additional condition in the random search step of the PatchMatch algorithm. In other words, we can avoid searching the interval where the disparity is negative.

This thesis focuses on the stereo matching part, which is based on the PatchMatch algorithm. Our input images are from public benchmarks, which have already been rectified and the camera parameters are also available.

[1]Note that we have used the same symbol d in Equation (2.3) to denote the maximum displacement. In the stereo part, we reuse it to denote the disparity. Its meaning should be clear from the context.

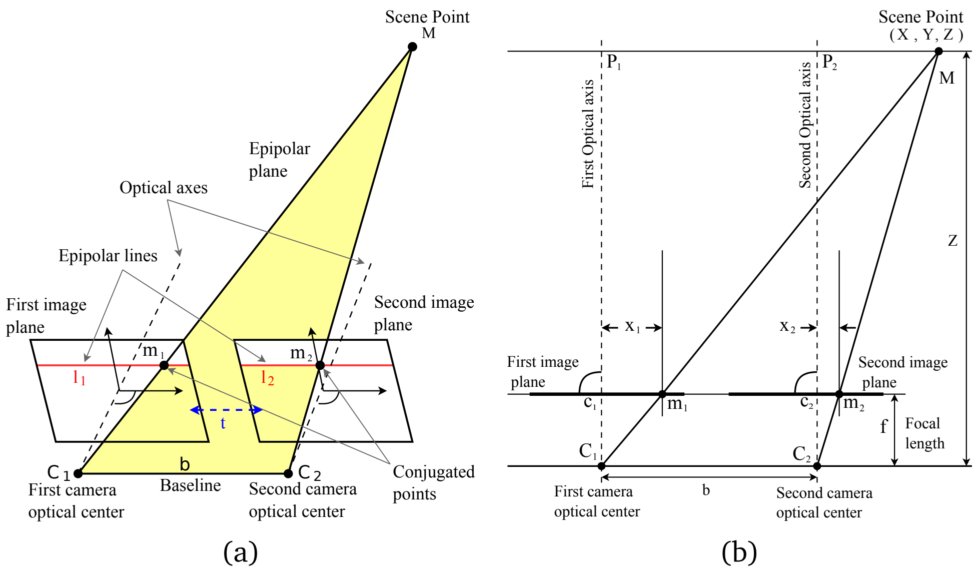


Figure 5.2: Illustration of depth recovery via triangulation. (a) Ortho-parallel setting (3-D view). (b) Top view. Image source: Lecture slides from the course Correspondence Problems in Computer Vision taught by Prof. Andrés Bruhn, Summer Semester 2016, Universität Stuttgart, http://www.vis.uni-stuttgart.de/nc/lehre/details/typ/vorlesung/2016/240.html.

5.2 Models for Stereo Matching

Similar to the motion models presented in Chapter 4, we first introduce some models for stereo matching and then use the PatchMatch algorithm to estimate their parameters.

5.2.1 The Epipolar Constraint Model

Let $\mathbf{x}_1 = (x_1, y_1)$ be a point in the left view $\mathbf{f}$; its corresponding point in the right view $\mathbf{g}$ is $\mathbf{x}_2 = (x_2, y_2)$ and $\mathbf{F}$ is the fundamental matrix. The epipolar constraint reads [HZ03; MSKS12]

$$\tilde{\mathbf{x}}_2^\top \mathbf{F} \tilde{\mathbf{x}}_1 = 0 \tag{5.2}$$

where $\tilde{\mathbf{x}}_1$ and $\tilde{\mathbf{x}}_2$ are the homogeneous coordinates of $\mathbf{x}_1$ and $\mathbf{x}_2$, respectively. In the case of ortho-parallel configuration, in other words, the second camera is only translated along the x-axis with respect to the first (reference) camera, $\mathbf{F}$ reduces to [HZ03]

$$\mathbf{F} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} . \tag{5.3}$$

Hence Equation (5.2) changes to

$$y_2 = y_1 \tag{5.4}$$

which means $\mathbf{x}_2$ has the same y coordinate as $\mathbf{x}_1$ and we only need to perform searches along the x-axis direction to find $\mathbf{x}_2$. The stereo matching problem is therefore easier than the motion estimation problem, which needs to conduct searches in two directions.

The disparity d is defined as

$$d = x_1 - x_2 \tag{5.5}$$

It is reasonable to assume that adjacent pixels have similar disparities. We can therefore use the disparity d as the model parameter, which is going to be estimated by PatchMatch.

5.2.2 The Projective Planar Model

Similar to the projective motion model presented in Section 4.2.3, we assume there exists a 2-D homography between (x_1, y_1) and (x_2, y_2). Different from the motion estimation, we have an additional constraint in stereo matching. When the images are rectified, we have $y_1 = y_2$. Taking this additional condition into consideration, Equation (4.12) changes to

$$\begin{aligned} x_2 &= \frac{h_1 x_1 + h_2 y_1 + h_3}{h_4 x_1 + h_5 y_1 + h_6} \cdot y_2 \\ y_2 &= y_1 \end{aligned} \tag{5.6}$$

Once we know the parameters $h_i, i \in \{1, 2, \ldots, 6\}$, which are going to be estimated using the PatchMatch algorithm, the disparity can be obtained using Equation (5.6).

5.2.3 The Slanted Plane Model

The slanted plane model is first presented by Bleyer et al. [BRR11] for stereo matching, which assumes that (x, y, d) lies on a 3-D plane with normal vector (n_x, n_y, n_z), where d is the disparity at position (x, y). The equation of the 3-D plane passing through point (x_1, y_1, d_1) with normal vector (n_x, n_y, n_z) is

$$n_x(x - x_1) + n_y(y - y_1) + n_y(d - d_1) = 0 \tag{5.7}$$

It assumes that adjacent pixels lie on the same 3-D plane. Therefore, if we know the disparity d_1 at position (x_1, y_1), the disparity of its neighboring pixel (x, y) can be obtained via

$$d = \frac{n_x x_1 + n_y y_1 + n_z d_1}{n_z} - \frac{n_x}{n_z} x - \frac{n_y}{n_z} y \tag{5.8}$$

Table 5.1 summarizes the models for stereo matching used in this thesis.

Model	Model parameters	Transformation rule
Epipolar constraint	$\mathbf{p} = d \in \mathbb{R}$	$d = x_1 - x_2$ (by definition)
Projective planar	$\mathbf{p} = (h_1, h_2, h_3, h_4, h_5, h_6) \in \mathbb{R}^6$	$x_2 = \dfrac{h_1 x_1 + h_2 y_1 + h_3}{h_4 x_1 + h_5 y_1 + h_6} \cdot y_2$ $y_2 = y_1$
Slanted plane	$\mathbf{p} = (n_x, n_y, n_z, d) \in \mathbb{R}^4$	$d = \frac{n_x x_1 + n_y y_1 + n_z d_1}{n_z} - \frac{n_x}{n_z} x - \frac{n_y}{n_z} y$

Table 5.1: Stereo matching models used in this thesis.

5.3 Cost Functions

A cost function is needed by PatchMatch to evaluate the cost of a given set of parameters. To simplify the notation, let $\mathcal{D}(\mathbf{p}; x, y)$ denote the disparity at the location (x, y) given the model parameter $\mathbf{p}$. Additionally, let (x_1, y_1) be a point in the left view $\mathbf{f}$ and (x_2, y_2) be a point in the right view $\mathbf{g}$.

We use the same cost functions as Hosni et al. [HRB+13] and Bleyer et al. [BRR11] that have been shown to be robust under illumination changes, which are defined as

$$\begin{aligned}\mathcal{C}(\mathbf{p}; x_1, y_1) = \sum_{\delta_x=-m}^{m} \sum_{\delta_y=-m}^{m} \Big(&\mathcal{W}\Big(f(x_1, y_1), f(x_1 + \delta_x, y_1 + \delta_y)\Big) \\ &\cdot \rho\Big(f(x_1+\delta_x, y_1+\delta_y), g(x_1+\delta_x-\mathcal{D}(\mathbf{p}; x_1+\delta_x, y_1+\delta_y), y_1+\delta_y)\Big)\Big)\end{aligned} \tag{5.9}$$

and

$$\begin{aligned}\mathcal{C}(\mathbf{p}; x_2, y_2) = \sum_{\delta_x=-m}^{m} \sum_{\delta_y=-m}^{m} \Big(&\mathcal{W}\Big(g(x_2, y_2), g(x_2 + \delta_x, y_2 + \delta_y)\Big) \\ &\cdot \rho\Big(g(x_2+\delta_x, y_2+\delta_y), f(x_2+\delta_x+\mathcal{D}(\mathbf{p}; x_2+\delta_x, y_2+\delta_y), y_2+\delta_y)\Big)\Big)\end{aligned} \tag{5.10}$$

Here, m is the size of the neighborhood and $\mathcal{W}$ is a weight function defined as

$$\mathcal{W}\Big(f(x, y), f(x', y')\Big) = \exp\left(-\frac{\left\|f(x, y) - f(x', y')\right\|_1}{\gamma}\right), \tag{5.11}$$

where γ is a user defined parameter and $\|\cdot\|_1$ is the L_1–norm. The function ρ computes the dissimilarity between the pixel $f(x, y)$ and its corresponding pixel $g(x - d, y)$ in the other view and is defined as[2]

$$\begin{aligned}\rho\Big(f(x, y), g(x - d, y)\Big) = &(1 - \beta) \cdot \min(\|f(x, y) - g(x - d, y)\|_1, \tau_{col}) \\ &+ \beta \cdot \min(\|\nabla f(x, y) - \nabla g(x - d, y)\|_1, \tau_{grad})\end{aligned} \tag{5.12}$$

[2] Of course, it has to use $x + d$ when (x, y) is in the right view.

where β, τ_{col} and τ_{grad} are user defined parameters and ∇ denotes the gradient operator. Note that the function $\mathcal{W}$ is the well known range kernel in the context of bilateral filtering [TM98]. As suggested by Bleyer et al. [BRR11], the spatial kernel of bilateral filtering has little contribution to the estimation. For simplicity, we omit the spatial kernel here.

5.4 View Propagation

In addition to the spatial propagation in the standard PatchMatch algorithm, which assumes that adjacent pixels have similar model parameters, Bleyer et al. [BRR11] add another assumption that corresponding pixels also have similar model parameters. Therefore, during the propagation step a pixel should try model parameters not only from its neighbors but also from its corresponding pixel, which is the so-called view propagation[3], since model parameters are propagated from one view to the other view.

5.5 Post-Processing

We follow the same post-processing step as Bleyer et al. [BRR11][4], which consists of three steps: outlier removal, invalid pixels handling and weighted median filter.

Outlier Removal. Similar to the outlier removal strategy adopted in the motion estimation, we perform the forward-backward consistency check to remove estimations of occluded pixels. Equation (4.1) is reused here with the vertical displacement v set to 0. We set the threshold $T_{occ} = 0.5$.

Invalid Pixels Handling. After removing the estimations of occluded pixels, we have to assign some values to them. For each occluded pixel, we select two of its closest non-occluded neighbors. One is from the left side and the other comes from the right side. Afterwards, it computes two disparities based on the model parameters from these two neighbors and select the one resulting a smaller disparity as its own model parameter.

Weighted Median Filter. Weighted median filter is executed as the last step to refine the estimations. Note that it is applied only to occluded pixels that failed the forward-backward consistency check. The weight used in the weighted median filter is computed according to Equation (5.11). Figure 5.3 illustrates the disparity estimation process using the PatchMatch algorithm (with slanted plane model). Only one iteration is performed here. In other words, only the forward propagation is executed and no backward propagation is performed. It shows that the final result after weighted median filtering (see Figure 5.3 (l)) is very visually promising.

[3]There is also another type of propagation in the case of stereo video sequences, which is known as temporal propagation. Refer to [BRR11] for more details.

[4] We have tried to use the interpolation and refinement components from the EpicFlow algorithm for post-processing, but the final result is not satisfactory.

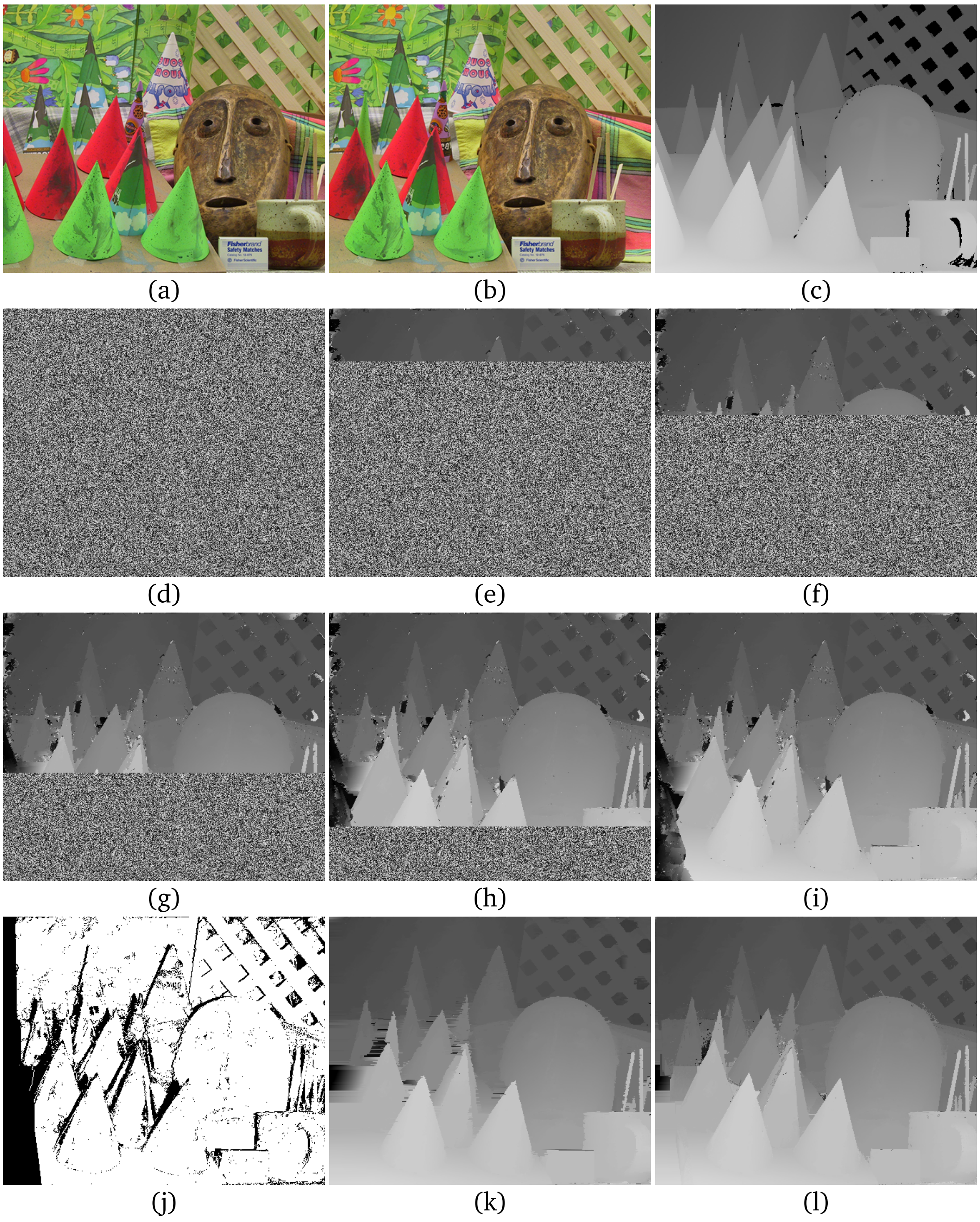


Figure 5.3: Illustration of the disparity estimation process using the Cones sequence from the Middlebury stereo dataset [SS02]. (a) Left view. (b) Right View. (c) Ground truth disparity. (d) After random initialization. (e) After $\frac{1}{5}$ iteration. (f) After $\frac{2}{5}$ iteration. (g) After $\frac{3}{5}$ iteration. (h) After $\frac{4}{5}$ iteration. (i) After one iteration. (j) Occluded map. Occluded pixels are shown in black. (k) After filling in invalid pixels. (l) Final result after weighted median filtering.

6 Evaluation and Experiments: Motion Estimation

This chapter presents the evaluation results of our algorithm for motion estimation using three public benchmarks: Middlebury [BSL+11], KITTI [GLU12; MG15] and MPI Sintel [BWSB12].

6.1 Performance Measures for Motion Estimation

In order to evaluate our algorithm, we present first the evaluation metrics for motion estimation that are widely used in the literature, which are

1. Average angular error (AAE)
2. Average endpoint error (AEE)
3. Percentage of bad pixels (BP)

AAE attracts a lot of attention after the seminal survey by Barron et al. [BFB94], which was first proposed by Fleet and Jepson [FJ90]. It measures the average angular error between the estimated optical flow $\mathbf{u}^e = (u_e, v_e, 1)$ and the ground truth optical flow $\mathbf{u}^t = (u_t, v_t, 1)$. The last component in the flow field is 1, which avoids division by 0. Furthermore, the flow field can be considered as a space-time direction vector in units of (pixel, pixel, frame) [BFB94]. The definition of AAE is as follows:

$$\mathrm{AAE}(\mathbf{u}^e, \mathbf{u}^t) = \frac{1}{NM} \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} \arccos\left(\frac{u^e_{i,j} u^t_{i,j} + v^e_{i,j} v^t_{i,j} + 1}{\sqrt{{u^e_{i,j}}^2 + {v^e_{i,j}}^2 + 1}\sqrt{{u^t_{i,j}}^2 + {v^t_{i,j}}^2 + 1}} \right). \tag{6.1}$$

AEE was proposed by Otte and Nagel [ON94] to avoid one of the drawbacks of AAE that it produces relatively small angular errors for differences of large vectors. AEE is defined as

$$\mathrm{AEE}(\mathbf{u}^e, \mathbf{u}^t) = \frac{1}{NM} \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} \sqrt{(u^e_{i,j} - u^t_{i,j})^2 + (v^e_{i,j} - v^t_{i,j})^2}. \tag{6.2}$$

BP is borrowed by Geiger et al. [GLU12] from the stereo correspondence [SS02], which is defined as

$$\mathrm{BP}(\mathbf{u}^e, \mathbf{u}^t) = \frac{100}{NM} \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} \mathbb{1}_{\sqrt{(u^e_{i,j} - u^t_{i,j})^2 + (v^e_{i,j} - v^t_{i,j})^2} > \delta_d}, \tag{6.3}$$

where $\mathbb{1}_{(\cdot)}$ is the indicator function defined in Equation (2.11) and δ_d is a positive integer denoting the threshold. In this thesis, we set δ_d to 3.

6.2 Parameter Optimization

There are three parameters in our implementation that are needed to be optimized: the number of pyramid levels L, the pyramid ratio η and the step size S. L and η are used for building the pyramid, while S determines the location (sampling distance) where the estimation should be performed. For instance, $S = 1$ means to estimate the flow field at every pixel and $S = 3$ means to perform estimation only at position $(1,1),(1,4),(1,7),\dots,(4,1),(4,4),\dots$, etc. Figure 4.4 (b) shows an example for $S = 6$.

The optimization framework written by Stoll et al. [SVMB17] is used, which could perform a parallel brute force search over a given set of parameters. The framework is available at the address http://www.vis.uni-stuttgart.de/∼cvis/research/scia2017/index.shtml.

In the following evaluations, we provide an optimized parameter set for each benchmark and keep it fixed within the benchmark. In other words, the training and test dataset from the same benchmark use the same set of parameters.

6.3 Evaluation On the Middlebury Dataset

The following sections present the evaluation results of our algorithm for the Middlebury dataset.

6.3.1 Optimized Parameters

Motion model	**Step size S**	**Pyramid ratio η**	**Pyramid levels L**
Translational	3	0.5	4
Affine	4	0.85	4
Projective	2	0.8	10
Quadratic	3	0.8	11

Table 6.1: Optimized parameters in terms of AEE for the Middlebury dataset.

Table 6.1 lists the optimized parameters of different motion models for the Middlebury dataset. They are obtained by running our algorithm over the 8 training sequences using the parameter combinations with $S \in \{2,3,4\}$, $\eta \in (0.45, 0.95)$ and $L \in \{1,2,\dots,12\}$. The combination with the minimal AEE is considered as the best one.

6.3.2 Comparison of Different Motion Models and Image Descriptors

Figure 6.1 visualizes the estimation results of different motion models over the eight training sequences from the Middlebury dataset[1] using the SIFT flow descriptor. It shows that the four models have visually similar estimations for the six sequences: Dimetrodon, Grove2, Grove3, Hydrangea, RubberWhale and Venus. For the Urban3 sequence, all models have some false estimations and the quadratic model performs worse than others.

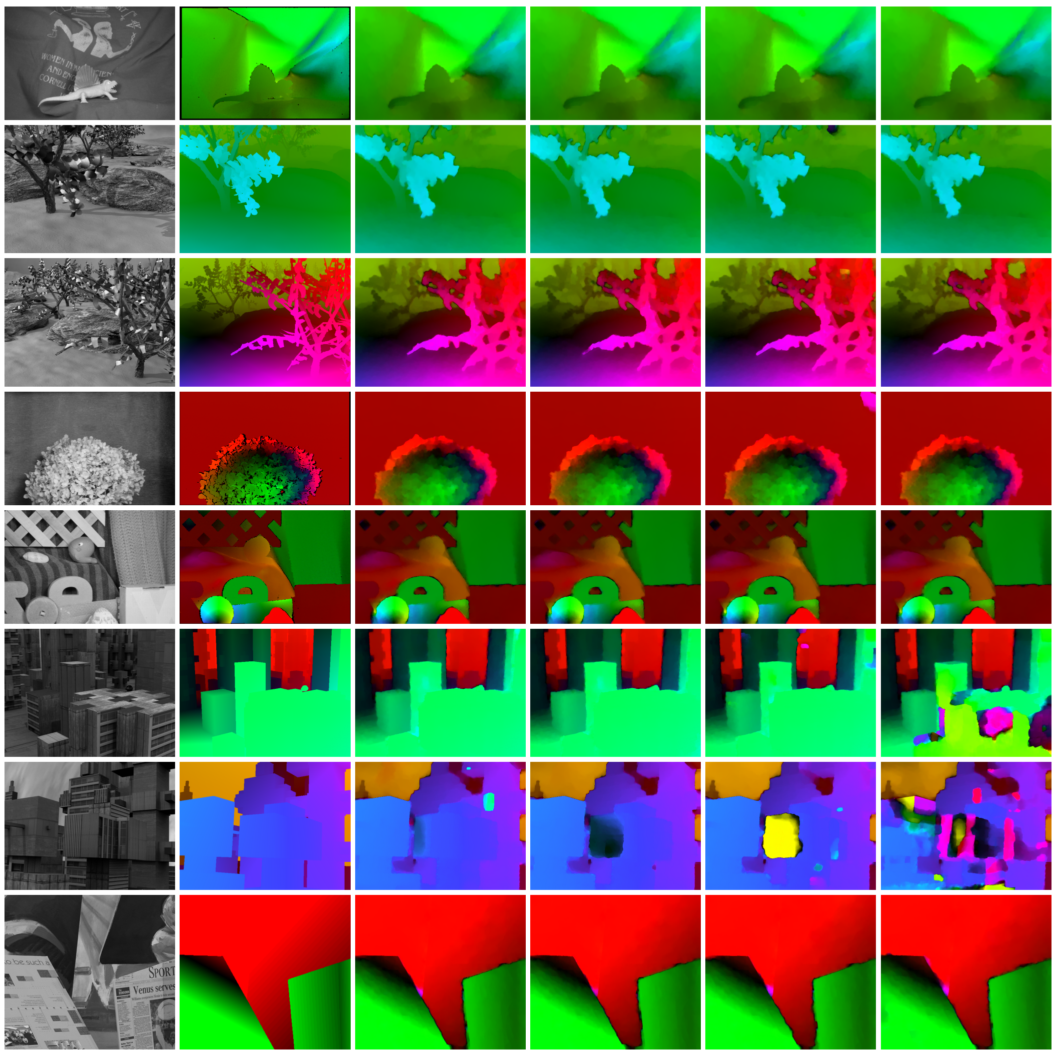


Figure 6.1: Visualization of the estimated results for different motion models on the Middlebury training dataset. **From left to right:** Reference frame, ground truth, results of translational, affine, projective and quadratic model. **From top to bottom:** Sequences of Dimetrodon, Grove2, Grove3, Hydrangea, RubberWhale, Urban2, Urban3 and Venus.

[1]The Middlebury training dataset is available at the address http://vision.middlebury.edu/flow/data/ .

Table 6.2 lists the results of different models and different descriptors for the Middlebury training dataset. It shows that the SIFT flow descriptor has the best results followed by the complete census descriptor. The results of different descriptors for different models are given in Table 6.3 to Table 6.12. It clearly shows that the translational model performs best among others both in AAE and AEE, while the affine model performs slightly worse than the translational model. As is shown in Figure 6.1, all models cannot handle the Urban3 sequence properly and the corresponding error rate is much larger than other sequences. Since the SIFT flow descriptor has the best performance and it runs faster than the complete census transform, we show only the results for the SIFT flow descriptor in the following evaluations.

Motion model	**SIFT flow**		**Rank**		**Census**		**Complete rank**		**Complete census**	
	AAE	AEE	AAE	AEE	AAE	AEE	AAE	AEE	AAE	AEE
Translational	3.374	0.293	3.957	0.391	3.681	0.363	3.845	0.393	3.331	0.353
Affine	3.511	0.337	4.486	0.418	4.471	0.424	4.458	0.407	4.452	0.407
Projective	5.135	0.489	18.357	5.605	5.714	0.616	6.064	0.831	5.392	0.480
Quadratic	8.226	1.398	10.376	1.769	9.201	1.575	9.281	1.595	8.228	1.437

Table 6.2: Comparison of different Motion models and image descriptors

Motion model	**Average**	**Dimetr.**	**Grove2**	**Grove3**	**Hydr.**	**Rubber.**	**Urban2**	**Urban3**	**Venus**
Translational	3.374	1.510	2.366	5.931	1.935	3.475	3.056	4.620	4.097
Affine	3.511	1.516	2.512	6.213	1.943	3.510	2.641	5.658	4.095
Projective	5.135	1.531	2.798	7.311	2.459	3.659	3.435	15.387	4.500
Quadratic	8.226	1.557	2.457	6.621	1.994	3.588	25.201	20.094	4.293

Table 6.3: Comparison of the AAE of different motion models for the Middlebury training dataset using the SIFT flow descriptor.

Motion model	**Average**	**Dimetr.**	**Grove2**	**Grove3**	**Hydr.**	**Rubber.**	**Urban2**	**Urban3**	**Venus**
Translational	0.293	0.076	0.156	0.627	0.158	0.105	0.280	0.674	0.266
Affine	0.337	0.077	0.166	0.654	0.162	0.106	0.367	0.899	0.266
Projective	0.489	0.084	0.177	0.747	0.215	0.110	0.515	1.779	0.285
Quadratic	1.398	0.079	0.165	0.677	0.173	0.109	6.297	3.367	0.318

Table 6.4: Comparison of the AEE of different motion models for the Middlebury training dataset using the SIFT flow descriptor.

Motion model	**Average**	**Dimetr.**	**Grove2**	**Grove3**	**Hydr.**	**Rubber.**	**Urban2**	**Urban3**	**Venus**
Translational	3.957	1.561	2.358	5.971	1.914	3.415	2.381	10.235	3.820
Affine	4.486	1.553	2.362	5.928	1.933	3.421	2.402	14.493	3.795
Projective	18.357	5.892	3.763	30.043	7.320	3.804	24.706	40.483	30.842
Quadratic	10.376	1.584	2.535	7.390	2.061	3.565	36.086	25.005	4.779

Table 6.5: Comparison of the AAE of different motion models for the Middlebury training dataset using the rank transform descriptor.

Motion model	**Average**	**Dimetr.**	**Grove2**	**Grove3**	**Hydr.**	**Rubber.**	**Urban2**	**Urban3**	**Venus**
Translational	0.391	0.078	0.157	0.625	0.157	0.105	0.257	1.488	0.259
Affine	0.418	0.078	0.157	0.629	0.162	0.105	0.274	1.684	0.255
Projective	5.605	0.254	0.383	27.547	0.458	0.115	5.918	5.757	4.410
Quadratic	1.769	0.080	0.172	0.727	0.200	0.110	8.409	3.948	0.502

Table 6.6: Comparison of the AEE of different motion models for the Middlebury training dataset using the rank transform descriptor.

Motion model	**Average**	**Dimetr.**	**Grove2**	**Grove3**	**Hydr.**	**Rubber.**	**Urban2**	**Urban3**	**Venus**
Translational	3.681	1.597	2.373	5.946	1.916	3.452	2.443	7.690	4.027
Affine	4.471	1.570	2.371	5.998	1.934	3.459	2.375	14.192	3.895
Projective	5.714	1.484	2.648	7.030	6.906	3.611	3.146	15.694	5.189
Quadratic	9.201	1.614	3.798	7.025	2.079	3.633	27.970	23.040	4.449

Table 6.7: Comparison of the AAE of different motion models for the Middlebury training dataset using the census transform descriptor.

Motion model	**Average**	**Dimetr.**	**Grove2**	**Grove3**	**Hydr.**	**Rubber.**	**Urban2**	**Urban3**	**Venus**
Translational	0.363	0.080	0.157	0.626	0.156	0.106	0.268	1.245	0.264
Affine	0.424	0.079	0.157	0.637	0.161	0.106	0.254	1.741	0.258
Projective	0.616	0.074	0.173	0.730	1.225	0.109	0.298	1.867	0.448
Quadratic	1.575	0.081	0.246	0.707	0.192	0.113	7.230	3.632	0.397

Table 6.8: Comparison of the AEE of different motion models for the Middlebury training dataset using the census transform descriptor.

Motion model	**Average**	**Dimetr.**	**Grove2**	**Grove3**	**Hydr.**	**Rubber.**	**Urban2**	**Urban3**	**Venus**
Translational	3.845	1.588	2.387	5.982	1.920	3.485	2.438	8.909	4.054
Affine	4.458	1.546	2.370	6.049	1.931	3.476	2.387	14.020	3.883
Projective	6.064	1.494	2.526	9.488	2.033	9.678	2.746	15.160	5.389
Quadratic	9.281	1.636	2.875	6.889	2.006	3.616	27.784	24.200	5.240

Table 6.9: Comparison of the AAE of different motion models for the Middlebury training dataset using the complete rank transform descriptor.

Motion model	**Average**	**Dimetr.**	**Grove2**	**Grove3**	**Hydr.**	**Rubber.**	**Urban2**	**Urban3**	**Venus**
Translational	0.393	0.079	0.159	0.629	0.157	0.107	0.269	1.479	0.267
Affine	0.407	0.078	0.157	0.641	0.161	0.106	0.250	1.604	0.259
Projective	0.831	0.075	0.167	0.999	0.181	2.820	0.279	1.804	0.323
Quadratic	1.595	0.083	0.189	0.697	0.177	0.112	7.433	3.588	0.483

Table 6.10: Comparison of the AEE of different motion models for the Middlebury training dataset using the complete rank transform descriptor.

Motion model	Average	Dimetr.	Grove2	Grove3	Hydr.	Rubber.	Urban2	Urban3	Venus
Translational	3.331	1.578	2.380	5.929	1.933	3.481	2.549	4.837	3.963
Affine	4.452	1.560	2.387	6.019	1.919	3.487	2.413	13.194	3.918
Projective	5.392	1.568	2.498	6.813	2.002	3.622	2.653	14.440	9.541
Quadratic	8.228	1.666	2.550	6.932	2.008	3.626	24.937	19.641	4.462

Table 6.11: Comparison of the AAE of different motion models for the Middlebury training dataset using the complete census transform descriptor.

Motion model	Average	Dimetr.	Grove2	Grove3	Hydr.	Rubber.	Urban2	Urban3	Venus
Translational	0.353	0.080	0.158	0.627	0.158	0.106	0.273	1.163	0.262
Affine	0.407	0.078	0.159	0.639	0.159	0.107	0.269	1.581	0.260
Projective	0.480	0.081	0.165	0.738	0.168	0.110	0.262	1.579	0.733
Quadratic	1.437	0.083	0.171	0.700	0.177	0.113	6.873	2.977	0.401

Table 6.12: Comparison of the AEE of different motion models for the Middlebury training dataset using the complete census transform descriptor.

6.3.3 Comparison with Other Methods

Table 6.13 and Table 6.14 list the estimation results of our method using the translational model and the results of some other methods in the literature. It shows that our algorithm has comparable performance in terms of AEE with others, while it performs slightly worse in terms of AAE.

Method	Average	Dimetr.	Grove2	Grove3	Hydr.	Rubber.	Urban2	Urban3	Venus
CostFilter [HRB+13]	0.280	0.200	0.160	0.510	0.170	0.090	0.330	0.590	0.170
A-Huber-L1 [WTP+09]	0.288	0.140	0.140	0.550	0.160	0.090	0.400	0.480	0.340
Ours	0.293	0.076	0.156	0.627	0.158	0.105	0.280	0.674	0.266
Steered-L1 [Zay16]	0.300	0.140	0.170	0.570	0.160	0.080	0.530	0.460	0.310
TV-L1-improved [WPZ+09]	0.307	0.190	0.154	0.665	0.147	0.092	0.319	0.630	0.260

Table 6.13: Comparison of the AEE with other methods in the literature for the Middlebury training dataset.

Method	Average	Dimetr.	Grove2	Grove3	Hydr.	Rubber.	Urban2	Urban3	Venus
ALD-Flow [SVB12]	2.567	1.830	1.750	4.940	1.680	2.230	1.880	2.740	3.480
CostFilter [HRB+13]	3.200	3.900	2.320	5.430	2.150	2.910	2.890	4.510	1.480
Ours	3.374	1.510	2.366	5.931	1.935	3.475	3.056	4.620	4.097
Correlation Flow [DN13]	3.585	4.540	2.130	5.640	2.030	2.650	2.920	4.750	4.020

Table 6.14: Comparison of the AAE with other methods in the literature on the Middlebury training dataset.

Figure 6.2 visualizes the estimation results of the four models over the 12 test sequences of the Middlebury dataset. As expected from the training result, the quadratic model performs badly on the Urban sequence. The affine motion model nearly misses the ball in the Backyard sequence. This is due to the larger step size ($S = 4$) it used compared with others. It could have captured the ball if $S = 3$ were used, while the quadratic model misses the ball completely. The running time[2] of different motion models on the evaluation dataset is listed in Table 6.15, which clearly shows that the translational model needs the least time. The reason is that the translational model has a smaller pyramid ratio compared with other models. Due to the smaller step size used by the projective model, it produces more estimations than others and increases the interpolation time significantly.

Motion model	Army	Backyard	Basketball	Dumptruck	Evergreen	Grove
Translational	5.7	8.7	10.9	8.1	9.0	9.3
Affine	5.8	9.1	10.6	7.8	9.0	9.1
Projective	78.9	101.4	121.1	93.1	102.3	110.1
Quadratic	16.5	25.7	31.3	21.4	23.7	30.1
Motion model	**Mequon**	**Schefflera**	**Teddy**	**Urban**	**Wooden**	**Yosemite**
Translational	6.3	6.6	4.0	9.9	6.3	1.8
Affine	7.3	6.6	4.7	9.1	6.6	2.2
Projective	77.6	83.2	51.1	107.4	77.2	22.9
Quadratic	19.6	21.4	14.9	28.5	19.8	6.0

Table 6.15: Running time in second of different motion models on the Middlebury evaluation dataset.

Figure 6.3 shows the rank of our algorithm on the Middlebury benchmark in terms of AAE and AEE.

6.4 Evaluation On the KITTI Dataset

The following sections present the evaluation results of our algorithm on the KITTI dataset.

6.4.1 Optimized Parameters

There are two KITTI datasets: KITTI-2012 and KITTI-2015[3]. Each dataset contains two different kinds of ground truth: occluded (occ) and non-occluded (noc). Because the displacement in the KITTI dataset is much larger than that in the Middlebury dataset, we cannot reuse the parameters obtained for the Middlebury dataset.

We use 10% data from each dataset, which is 20 image pairs, to train our algorithm. The obtained optimized parameters are listed in Table 6.16 and are kept fixed while performing evaluation on the two datasets.

[2]They are measured on the processor AMD FX(tm)-6300 Six-Core, 3.5 GHz and the algorithm is implemented using C++.

[3]The KITTI-2012 dataset can be downloaded from the address http://kitti.is.tue.mpg.de/kitti/data_stereo_flow.zip and KITTI-2015 is available at the address http://kitti.is.tue.mpg.de/kitti/data_scene_flow.zip

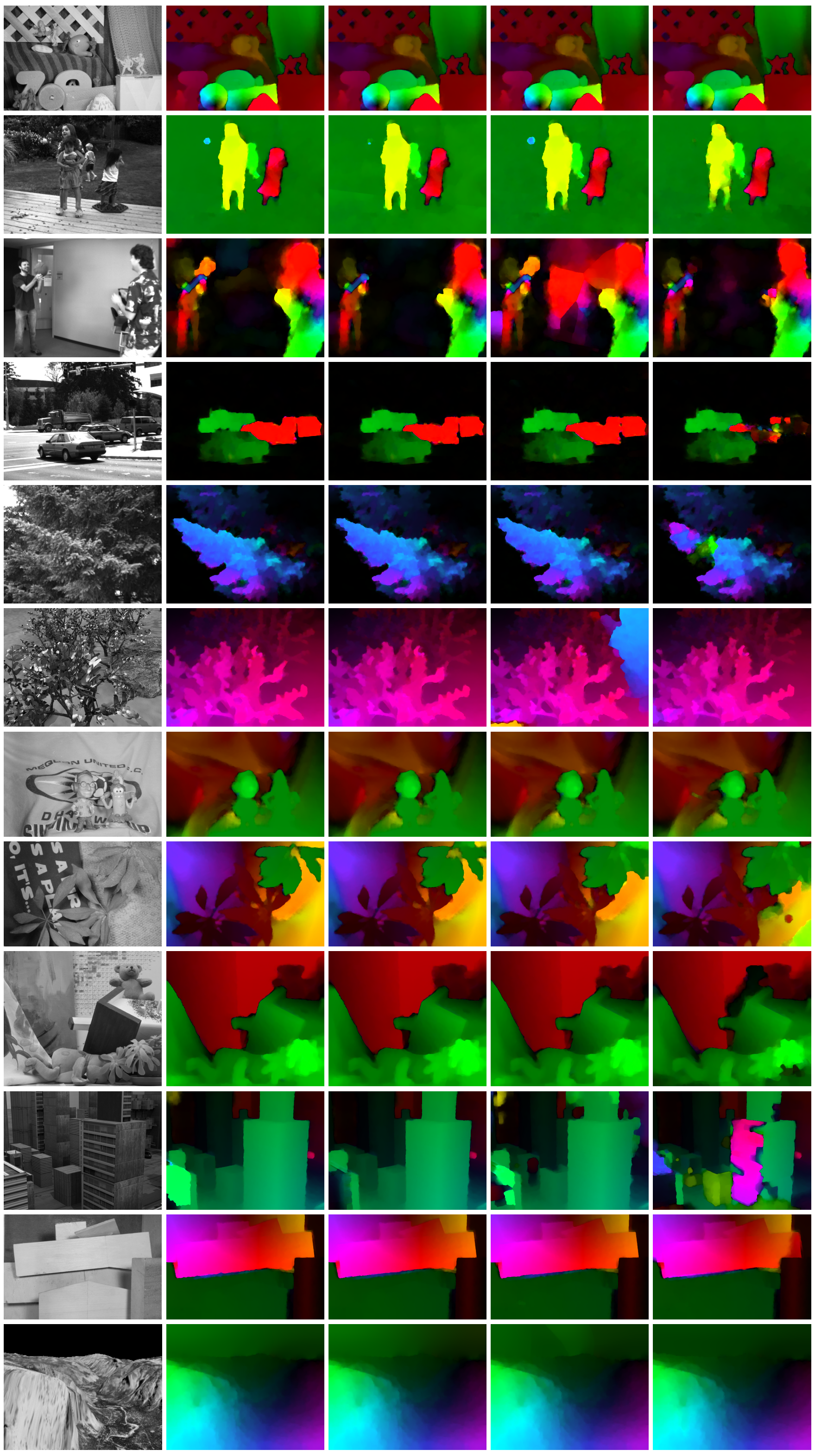
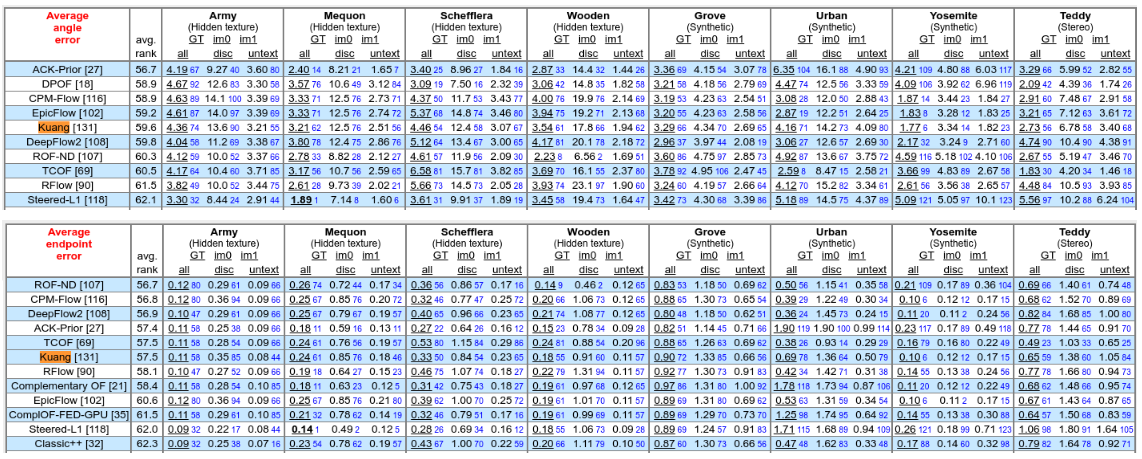


Figure 6.2: Visualization of the estimated results for different motion models on Middlebury test dataset. **From left to right:** Reference frame, results of translational, affine, projective and quadratic model. **From top to right:** Sequences of Army, Backyard, Basketball, Dumptruck, Evergreen, Grove, Mequon, Schefflera, Teddy, Urban, Wooden and Yosemite.



Average angle error	avg. rank	Army (Hidden texture) GT im0 im1 all	disc	untext	Mequon (Hidden texture) GT im0 im1 all	disc	untext	Schefflera (Hidden texture) GT im0 im1 all	disc	untext	Wooden (Hidden texture) GT im0 im1 all	disc	untext	Grove (Synthetic) GT im0 im1 all	disc	untext	Urban (Synthetic) GT im0 im1 all	disc	untext	Yosemite (Synthetic) GT im0 im1 all	disc	untext	Teddy (Stereo) GT im0 im1 all	disc	untext
ACK-Prior [27]	56.7	4.19 67	9.27 40	3.60 80	2.40 14	8.21 21	1.65 7	3.40 25	8.96 27	1.84 16	2.87 33	14.4 32	1.44 26	3.36 69	4.15 54	3.07 78	6.35 104	16.1 88	4.90 93	4.21 109	4.80 88	6.03 117	3.29 66	5.99 52	2.82 55
DPOF [18]	58.9	4.67 92	12.6 83	3.30 58	3.57 76	10.6 49	3.12 84	3.09 19	7.50 16	2.32 39	3.06 42	14.8 35	1.82 58	3.21 58	4.18 56	2.79 69	4.47 74	12.5 56	3.33 59	4.09 106	3.92 62	6.96 119	2.09 42	4.39 36	1.74 26
CPM-Flow [116]	58.9	4.63 89	14.1 100	3.39 69	3.33 71	12.5 76	2.73 71	4.37 50	11.7 53	3.43 77	4.00 76	19.9 76	2.14 69	3.19 53	4.23 63	2.54 51	3.08 28	12.0 50	2.88 43	1.87 14	3.44 23	1.84 27	2.91 60	7.48 67	2.91 58
EpicFlow [102]	59.2	4.61 87	14.0 97	3.39 69	3.33 71	12.5 76	2.74 72	5.37 68	14.8 74	3.46 80	3.94 75	19.2 71	2.13 68	3.20 55	4.23 63	2.58 56	2.87 19	12.2 51	2.64 25	1.83 8	3.28 12	1.83 25	3.21 65	7.12 63	3.61 72
Kuang [131]	59.6	4.36 74	13.6 90	3.21 55	3.21 62	12.5 76	2.51 56	4.46 54	12.4 58	3.07 67	3.54 61	17.8 66	1.94 62	3.29 66	4.34 70	2.69 65	4.16 71	14.2 73	4.09 80	1.77 6	3.34 14	1.82 23	2.73 56	6.78 58	3.40 68
DeepFlow2 [108]	59.8	4.04 58	11.2 69	3.38 67	3.80 78	12.4 75	2.86 76	5.12 64	13.4 67	3.00 65	4.17 81	20.1 78	2.18 72	2.96 37	3.97 44	2.08 19	3.06 27	12.6 57	2.69 30	2.17 32	3.24 9	2.71 60	4.74 90	10.4 90	4.38 91
ROF-ND [107]	60.3	4.12 59	10.0 52	3.37 66	2.78 33	8.82 28	2.12 27	4.61 57	11.9 56	2.09 30	2.23 8	6.56 2	1.69 51	3.60 86	4.75 97	2.85 73	4.92 87	13.6 67	3.75 72	4.59 116	5.18 102	4.10 106	2.67 55	5.19 47	3.46 70
TCOF [69]	60.5	4.17 64	10.4 60	3.71 85	3.17 56	10.7 56	2.59 65	6.58 81	15.7 81	3.82 85	3.69 70	16.1 55	2.37 80	3.78 92	4.95 106	2.47 45	2.59 8	8.47 15	2.58 21	3.66 99	4.83 89	2.67 58	1.83 30	4.20 34	1.46 18
RFlow [90]	61.5	3.82 49	10.0 52	3.44 75	2.61 28	9.73 39	2.02 21	5.66 73	14.5 73	2.05 28	3.93 74	23.1 97	1.90 60	3.24 60	4.19 57	2.66 64	4.12 70	15.2 82	3.34 61	2.61 56	3.56 38	2.65 57	4.48 84	10.5 93	3.93 85
Steered-L1 [118]	62.1	3.30 32	8.44 24	2.91 44	**1.89** 1	7.14 8	1.60 6	3.61 31	9.91 37	1.89 19	3.45 58	19.4 73	1.64 47	3.42 73	4.30 68	3.39 86	5.18 89	14.5 75	4.37 89	5.09 121	5.05 97	10.1 123	5.56 97	10.2 88	6.24 104

Average endpoint error	avg. rank	Army (Hidden texture) GT im0 im1 all	disc	untext	Mequon (Hidden texture) GT im0 im1 all	disc	untext	Schefflera (Hidden texture) GT im0 im1 all	disc	untext	Wooden (Hidden texture) GT im0 im1 all	disc	untext	Grove (Synthetic) GT im0 im1 all	disc	untext	Urban (Synthetic) GT im0 im1 all	disc	untext	Yosemite (Synthetic) GT im0 im1 all	disc	untext	Teddy (Stereo) GT im0 im1 all	disc	untext
ROF-ND [107]	56.7	0.12 80	0.29 61	0.09 66	0.26 74	0.72 44	0.17 34	0.36 56	0.86 57	0.17 16	0.14 9	0.46 2	0.12 65	0.83 53	1.18 50	0.69 62	0.50 56	1.15 41	0.35 58	0.21 109	0.17 89	0.36 104	0.69 66	1.40 61	0.74 48
CPM-Flow [116]	56.8	0.12 80	0.36 94	0.09 66	0.25 67	0.85 76	0.20 72	0.32 46	0.77 47	0.25 72	0.20 66	1.06 73	0.12 65	0.88 65	1.30 73	0.65 54	0.39 29	1.22 49	0.30 34	0.10 6	0.12 12	0.17 15	0.68 62	1.52 70	0.89 69
DeepFlow2 [108]	56.9	0.10 47	0.29 61	0.09 66	0.25 67	0.79 67	0.19 57	0.40 65	0.96 66	0.23 65	0.21 74	1.08 77	0.12 65	0.80 48	1.18 50	0.62 51	0.36 24	1.45 73	0.24 15	0.11 20	0.11 2	0.24 56	0.82 84	1.68 85	1.00 80
ACK-Prior [27]	57.4	0.11 58	0.25 38	0.09 66	0.18 11	0.59 16	0.13 11	0.27 22	0.64 26	0.16 12	0.15 23	0.78 34	0.09 28	0.82 51	1.14 45	0.71 66	1.90 119	1.90 100	0.99 114	0.23 117	0.17 89	0.49 118	0.77 78	1.44 65	0.91 70
TCOF [69]	57.5	0.11 58	0.28 54	0.09 66	0.24 61	0.76 56	0.19 57	0.53 80	1.15 84	0.29 86	0.24 81	0.88 54	0.20 96	0.88 65	1.26 63	0.69 62	0.38 26	0.93 14	0.29 29	0.16 79	0.16 80	0.22 49	0.49 23	1.03 33	0.65 25
Kuang [131]	57.5	0.11 58	0.35 85	0.08 44	0.24 61	0.85 76	0.18 46	0.33 50	0.84 54	0.23 65	0.18 55	0.91 60	0.11 57	0.90 72	1.33 85	0.66 56	0.69 78	1.36 64	0.50 79	0.10 6	0.12 12	0.17 15	0.65 59	1.38 60	1.05 84
RFlow [90]	58.1	0.10 47	0.27 52	0.09 66	0.19 18	0.64 27	0.15 23	0.46 75	1.07 74	0.18 27	0.22 79	1.31 94	0.11 57	0.92 77	1.30 73	0.91 83	0.42 34	1.42 71	0.31 38	0.14 55	0.13 38	0.24 56	0.77 78	1.66 80	0.94 73
Complementary OF [21]	58.4	0.11 58	0.28 54	0.10 85	0.18 11	0.63 23	0.12 5	0.31 42	0.75 43	0.18 27	0.19 61	0.97 68	0.12 65	0.97 86	1.31 80	1.00 92	1.78 118	1.73 94	0.87 106	0.11 20	0.12 12	0.22 49	0.68 62	1.48 66	0.95 74
EpicFlow [102]	60.6	0.12 80	0.36 94	0.09 66	0.25 67	0.85 76	0.21 80	0.39 62	1.00 70	0.25 72	0.19 61	1.01 70	0.11 57	0.89 69	1.31 80	0.69 62	0.53 63	1.31 59	0.34 54	0.10 6	0.11 2	0.17 15	0.67 61	1.43 64	0.87 65
ComplOF-FED-GPU [35]	61.5	0.11 58	0.29 61	0.10 85	0.21 32	0.78 62	0.14 19	0.32 46	0.79 51	0.17 16	0.19 61	0.99 69	0.11 57	0.89 69	1.29 70	0.73 70	1.25 98	1.74 95	0.64 92	0.14 55	0.13 38	0.30 88	0.64 57	1.50 68	0.83 59
Steered-L1 [118]	62.0	0.09 32	0.22 17	0.08 44	**0.14** 1	0.49 2	0.12 5	0.28 26	0.69 34	0.16 12	0.18 55	1.06 73	0.09 28	0.89 69	1.24 57	0.91 83	1.71 115	1.68 89	0.94 109	0.26 121	0.18 99	0.71 123	1.06 98	1.80 91	1.64 105
Classic++ [32]	62.3	0.09 32	0.25 38	0.07 16	0.23 54	0.78 62	0.19 57	0.43 67	1.00 70	0.22 59	0.20 66	1.11 79	0.10 50	0.87 60	1.30 73	0.66 56	0.47 48	1.62 83	0.33 48	0.17 88	0.14 60	0.32 98	0.79 82	1.64 78	0.92 71

Figure 6.3: The rank of our algorithm on the Middlebury benchmark, which is available at address http://vision.middlebury.edu/flow/eval/results-fangjun-kuang/. Last accessed: November 29, 2017.

	KITTI-2012			KITTI-2015		
Motion model	S	η	L	S	η	L
Translational	2	0.75	8	3	0.7	7
Affine	2	0.9	10	3	0.85	10
Projective	2	0.8	10	3	0.8	10
Quadratic	2	0.8	10	3	0.75	7

Table 6.16: Optimized parameters in terms of AEE (noc) on the KITTI-2012 and KITTI-2015 training dataset. They are kept fixed during the evaluation process.

6.4.2 Comparison of Different Motion Models

Table 6.17 compares the results of different motion models for the KITTI-2012 and KITTI-2015 training datasets. Like the results for the Middlebury dataset, the translational model performs again best here.

	KITTI-2012						KITTI-2015					
	noc			occ			noc			occ		
	AAE	AEE	BP(%)	AAE	AEE	BP(%)	AAE	AEE	BP(%)	AAE	AEE	BP(%)
Translational	1.53	1.34	5.92	2.36	4.06	14.59	3.67	3.94	15.82	5.33	9.70	24.79
Affine	4.09	5.39	15.99	9.24	12.68	24.96	11.70	14.36	34.31	18.55	24.51	40.75
Projective	8.90	9.04	27.49	9.08	12.09	34.42	20.44	21.91	46.98	21.20	28.66	51.44
Quadratic	32.90	16.66	53.62	40.17	24.72	58.87	40.61	21.93	59.11	46.85	31.83	62.65

Table 6.17: Results of different motion models for the KITTI-2012 and KITTI-2015 training datasets.

6.4.3 Comparison with Other Methods

Table 6.18 and Table 6.19 list the results of our algorithm on the KITTI benchmark along with other methods. Although we use the same interpolation method as EpicFlow, our results outperform its.

Method	Out-Noc	Out-All	Avg-Noc	Avg-All
S2F-IF	6.20 %	15.68 %	1.4 px	3.5 px
DiscreteFlow [MHG15]	6.23 %	16.63 %	1.3 px	3.6 px
Ours	6.23 %	15.91 %	1.4 px	5.0 px
BTF-ILLUM [DSV+14]	6.52 %	11.03 %	1.5 px	2.8 px
DeepFlow2 [RWHS16]	6.61 %	17.35 %	1.4 px	5.3 px
EpicFlow [RWHS15]	7.88 %	17.08 %	1.5 px	3.8 px

Table 6.18: Results for the KITTI-2012 test dataset, which are available at address http://www.cvlibs.net/datasets/kitti/eval_stereo_flow.php?benchmark=flow. Last accessed: November 13, 2017.

Method	Fl-bg	Fl-fg	Fl-all
FullFlow [CK16]	23.09 %	24.79 %	23.37 %
SPM-BP [LMB+15]	24.06 %	24.97 %	24.21 %
Ours	25.87 %	23.67 %	25.50 %
EpicFlow [RWHS15]	25.81 %	28.69 %	26.29 %
DeepFlow [WRHS13]	27.96 %	31.06 %	28.48 %

Table 6.19: Results for the KITTI-2015 test dataset, which are available at address http://www.cvlibs.net/datasets/kitti/eval_scene_flow.php?benchmark=flow. Last accessed: November 13, 2017.

6.5 Evaluation On the MPI Sintel Dataset

The following sections present the evaluation results of our algorithm on the MPI Sintel dataset.

6.5.1 Parameter Settings

We use the same parameter settings as KITTI-2015 (refer to Table 6.16).

6.5.2 Comparison of Different Motion Models

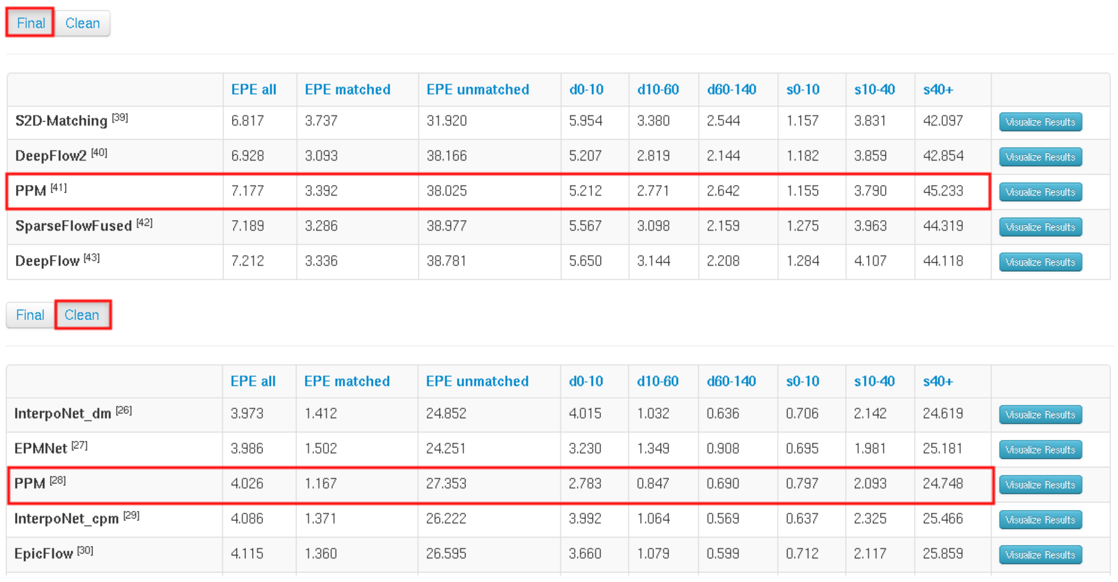
Table 6.20 lists the estimation results of different models for the MPI Sintel clean dataset. It shows that the translational model performs best and the quadratic model has the highest error rate. The affine model performs slightly worse than the translational model.

Motion model	AAE	AEE	BP(%)
Translational	4.238	2.153	5.926
Affine	4.834	3.133	7.183
Projective	9.249	5.609	13.725
Quadratic	18.678	9.652	27.124

Table 6.20: Comparison of different motion models for the MPI Sintel clean dataset.

6.5.3 Comparison with Other Methods

Figure 6.4 shows the rank of the translational motion model for the MPI Sintel test dataset. Note that our estimation for the clean version is slightly better than the EpicFlow algorithm, which indicates that the estimation of the initial flow field of our algorithm is better than the DeepFlow algorithm.



Final Clean

	EPE all	EPE matched	EPE unmatched	d0-10	d10-60	d60-140	s0-10	s10-40	s40+	
S2D-Matching [39]	6.817	3.737	31.920	5.954	3.380	2.544	1.157	3.831	42.097	Visualize Results
DeepFlow2 [40]	6.928	3.093	38.166	5.207	2.819	2.144	1.182	3.859	42.854	Visualize Results
PPM [41]	7.177	3.392	38.025	5.212	2.771	2.642	1.155	3.790	45.233	Visualize Results
SparseFlowFused [42]	7.189	3.286	38.977	5.567	3.098	2.159	1.275	3.963	44.319	Visualize Results
DeepFlow [43]	7.212	3.336	38.781	5.650	3.144	2.208	1.284	4.107	44.118	Visualize Results

Final Clean

	EPE all	EPE matched	EPE unmatched	d0-10	d10-60	d60-140	s0-10	s10-40	s40+	
InterpoNet_dm [26]	3.973	1.412	24.852	4.015	1.032	0.636	0.706	2.142	24.619	Visualize Results
EPMNet [27]	3.986	1.502	24.251	3.230	1.349	0.908	0.695	1.981	25.181	Visualize Results
PPM [28]	4.026	1.167	27.353	2.783	0.847	0.690	0.797	2.093	24.748	Visualize Results
InterpoNet_cpm [29]	4.086	1.371	26.222	3.992	1.064	0.569	0.637	2.325	25.466	Visualize Results
EpicFlow [30]	4.115	1.360	26.595	3.660	1.079	0.599	0.712	2.117	25.859	Visualize Results

Figure 6.4: Comparison with other methods for the MPI Sintel test dataset using the translational motion model. **Top:** The final version. **Bottom:** The clean version. The results are available at the address http://sintel.is.tue.mpg.de/results. Last accessed: November 29, 2017.

7 Evaluation and Experiments: Stereo Matching

This chapter presents our evaluation results for the Middlebury stereo evaluation version 2 [SS02] and version 3 [SHK+14].

7.1 Performance Measures for Stereo Matching

In the stereo correspondence community, BP (see the definition in Section 6.1) is usually used to evaluate the estimation performance. In addition, stereo matching distinguishes BP over different regions, such as textureless regions, occluded regions and depth discontinuity regions. A mask image is provided for each region. Only areas with non-zero intensity values in the mask image contribute to the final BP. Please refer to the seminal survey by Scharstein et al. [SS02] for more details.

7.2 Parameter Settings

Table 7.1 lists the common parameters shared by the three stereo matching models. The patch sizes of different models for the Middlebury stereo evaluation version 2 and 3 are given in Table 7.2. We only perform one iteration in the PatchMatch algorithm. In other words, only the forward propagation is executed and no backward propagation is performed. Because we find that multiple iterations do not improve the final estimations too much but instead increase the total estimation time.

Parameters	γ	β	τ_{col}	τ_{grad}
Values	10	0.9	10	2

Table 7.1: Common parameters for all stereo matching models.

Model	**V2**	**V3**
Slanted plane	31×31	39×39
Epipolar constraint	27×27	39×39
Projective planar	35×35	39×39

Table 7.2: Patch sizes of different stereo matching models for the Middlebury stereo evaluation version 2 and 3.

7.3 Middlebury Stereo Evaluation Version 2

Figure 7.1 visualizes the estimation results of different models for the Middlebury stereo evaluation version 2. It shows that the three models have visually similar estimations.

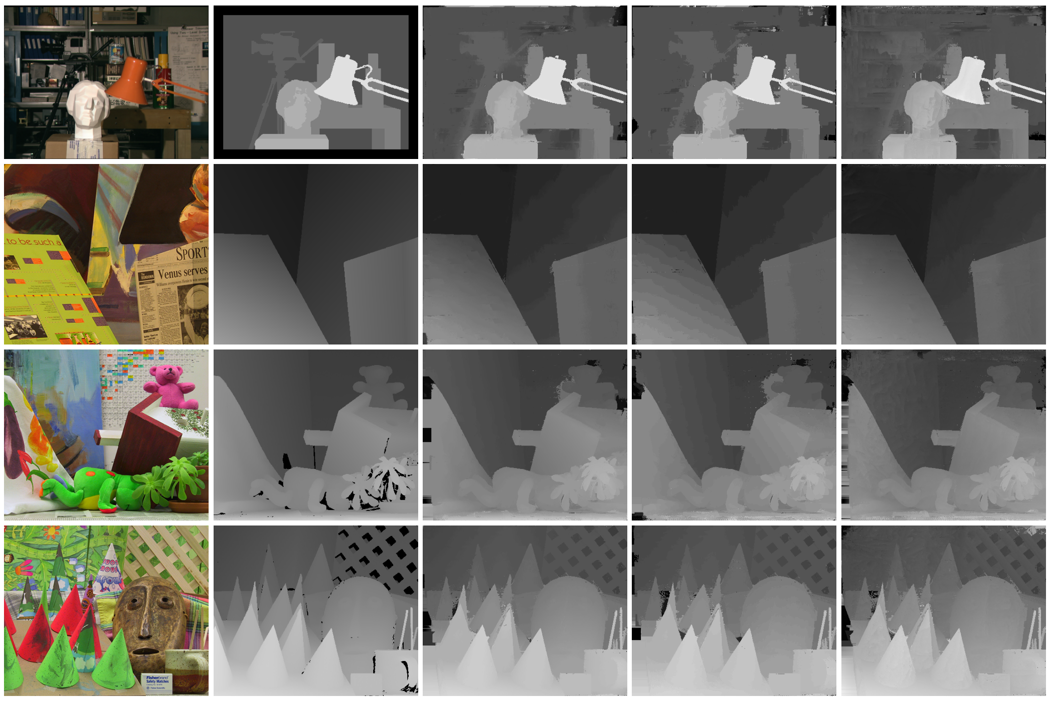


Figure 7.1: Results for the Middlebury stereo evaluation version 2. **From left to right:** Left view, ground truth disparity map, results of the slanted plane model, the epipolar constraint model and the projective planar model. **From top to left:** Sequences of Tsukuba, Venus, Teddy and Cones.

The quantitative results of each model and their comparisons with some classical methods are shown in Table 7.3. It shows that the three stereo matching models have competitive performances for the Tsukuba and Venus sequence. The performances of the epipolar constraint model and the projective planar model become worse compared with the slanted plane model for the Teddy and Cones sequence. One possible reason is that the true disparities of Tsukuba and Venus, which are about 20 pixels, are less than that of Teddy and Cones, which are about 60 pixels. From the average results we can see that our implementation of the slanted plane model performs slightly worse than [BRR11]. The epipolar constraint model and the projective planar model have competitive performances or even outperform the following classical methods: SemiGlob [Hir08], VarMSOH [BS10], CSBP [YWA10] and GC [BVZ01].

Method	Tsukuba			Venus			Teddy			Cones			Average		
	Nonocc	All	Disc	Nonocc	All	Disc	Nonocc	All	Disc	Nonocc	All	Disc	Nonocc	All	Disc
Slanted plane (our implementation)	2.70	3.01	9.65	0.34	0.59	2.72	4.06	9.55	11.31	2.52	7.79	7.18	2.41	5.24	7.72
Epipolar constraint	2.10	2.37	8.51	0.28	0.49	1.94	6.93	12.57	17.15	3.24	9.29	8.65	3.14	6.18	9.06
Projective planar	3.38	3.78	11.62	0.24	0.43	2.11	10.13	15.84	17.32	6.49	11.86	12.34	5.06	7.98	10.85
Slanted plane [BRR11]	2.09	2.33	9.31	0.21	0.39	2.62	2.99	8.16	9.62	2.47	7.80	7.11	1.94	4.67	7.17
SemiGlob [Hir08]	3.26	3.96	12.80	1.00	1.57	11.30	6.02	12.20	16.30	3.06	9.75	8.90	3.34	6.87	12.33
VarMSOH [BS10]	3.97	5.23	14.90	0.28	0.76	3.78	9.34	14.30	20.00	4.14	9.91	11.40	4.43	7.55	12.52
CSBP [YWA10]	2.00	4.17	10.50	1.48	3.11	17.70	11.10	20.20	27.50	5.98	16.50	16.00	5.14	11.00	17.93
GC [BVZ01]	1.94	4.12	9.39	1.79	3.44	8.75	16.50	25.00	24.90	7.70	18.20	15.30	6.98	12.69	14.59

Table 7.3: Quantitative results of our methods and comparisons with some classical methods for the Middlebury stereo evaluation version 2 using error threshold 1.0 (Middlebury default threshold).

7.4 Middlebury Stereo Evaluation Version 3

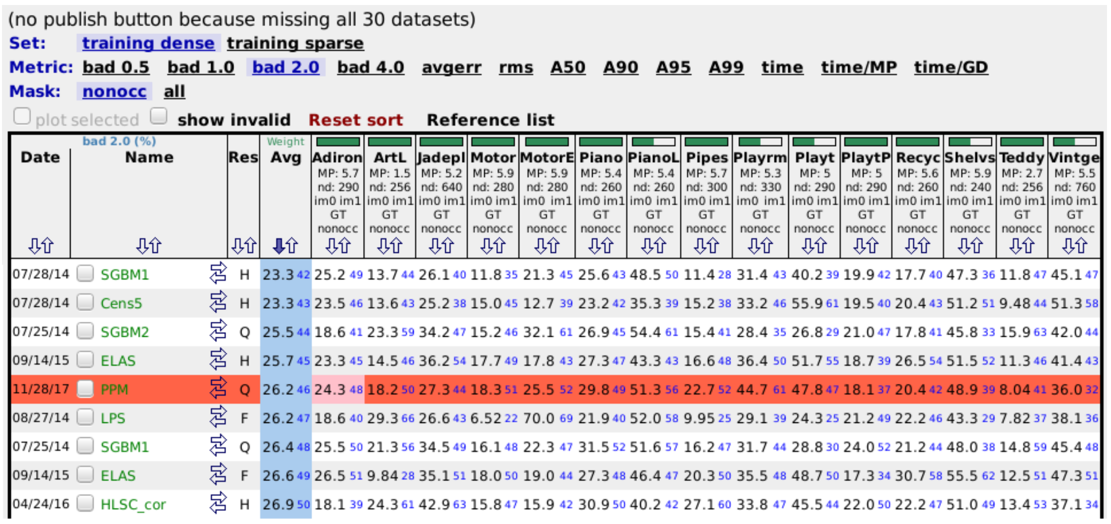
Table 7.4 gives the quantitative results of different models for the Middlebury stereo evaluation version 3[1]. It shows that the slanted plane model outperforms the epipolar constraint model and the projective planar model, while the epipolar constraint model and the projective model have similar performances. Since the true disparities in the Middlebury stereo evaluation version 3 is much larger than that in the Middlebury stereo evaluation version 2, it shows again the slanted plane model performs best among the three models in the case of large disparities.

Datasets	Slanted plane	Epipolar constraint	Projective planar
Adirondack	24.30	42.70	47.00
ArtL	18.20	26.90	46.00
Jadeplant	27.30	45.90	59.10
MotorCycle	18.30	40.20	31.90
MotorCycleE	25.50	45.20	37.90
Piano	29.80	53.00	46.50
PianoL	51.30	64.00	61.70
Pipes	22.70	33.60	43.50
Playroom	44.70	53.30	50.80
Playtable	47.80	68.10	59.60
PlaytableP	18.10	56.80	27.00
Recycle	20.40	35.80	39.10
Shelves	48.90	55.30	55.00
Teddy	8.04	15.20	29.70
Vintage	36.00	66.60	51.30
Weighted average	26.20	43.90	43.70

Table 7.4: Error rates of different models for the Middlebury stereo evaluation version 3. The error threshold is set to 2.0, which is the default value used by the benchmark.

Figure 7.2 shows the rank of the slanted plane model for the training dataset. The estimation results of different models are visualized in Figure 7.3 and Figure 7.4.

[1] The weights for computing the weighted average are available at the address http://vision.middlebury.edu/stereo/eval3/



(no publish button because missing all 30 datasets)

Set: training dense training sparse

Metric: bad 0.5 bad 1.0 bad 2.0 bad 4.0 avgerr rms A50 A90 A95 A99 time time/MP time/GD

Mask: nonocc all

plot selected show invalid Reset sort Reference list

Date	bad 2.0 (%) Name	Res	Weight Avg	Adiron MP: 5.7 nd: 290 im0 im1 GT nonocc	ArtL MP: 1.5 nd: 256 im0 im1 GT nonocc	Jadepl MP: 5.2 nd: 640 im0 im1 GT nonocc	Motor MP: 5.9 nd: 280 im0 im1 GT nonocc	MotorE MP: 5.9 nd: 280 im0 im1 GT nonocc	Piano MP: 5.4 nd: 260 im0 im1 GT nonocc	PianoL MP: 5.4 nd: 260 im0 im1 GT nonocc	Pipes MP: 5.7 nd: 300 im0 im1 GT nonocc	Playrm MP: 5.3 nd: 330 im0 im1 GT nonocc	Playt MP: 5 nd: 290 im0 im1 GT nonocc	PlaytP MP: 5 nd: 290 im0 im1 GT nonocc	Recyc MP: 5.6 nd: 260 im0 im1 GT nonocc	Shelvs MP: 5.9 nd: 240 im0 im1 GT nonocc	Teddy MP: 2.7 nd: 256 im0 im1 GT nonocc	Vintge MP: 5.5 nd: 760 im0 im1 GT nonocc
07/28/14	SGBM1	H	23.3 42	25.2 49	13.7 44	26.1 40	11.8 35	21.3 45	25.6 43	48.5 50	11.4 28	31.4 43	40.2 39	19.9 42	17.7 40	47.3 36	11.8 47	45.1 47
07/28/14	Cens5	H	23.3 43	23.5 46	13.6 43	25.2 38	15.0 45	12.7 39	23.2 42	35.3 39	15.2 38	33.2 46	55.9 61	19.5 40	20.4 43	51.2 51	9.48 44	51.3 58
07/25/14	SGBM2	Q	25.5 44	18.6 41	23.3 59	34.2 47	15.2 46	32.1 61	26.9 45	54.4 61	15.4 41	28.4 35	26.8 29	21.0 47	17.8 41	45.8 33	15.9 63	42.0 44
09/14/15	ELAS	H	25.7 45	23.3 45	14.5 46	36.2 54	17.7 49	17.8 43	27.3 47	43.3 43	16.6 48	36.4 50	51.7 55	18.7 39	26.5 54	51.5 52	11.3 46	41.4 43
11/28/17	PPM	Q	26.2 46	24.3 48	18.2 50	27.3 44	18.3 51	25.5 52	29.8 49	51.3 56	22.7 52	44.7 61	47.8 47	18.1 37	20.4 42	48.9 39	8.04 41	36.0 32
08/27/14	LPS	F	26.2 47	18.6 40	29.3 66	26.6 43	6.52 22	70.0 69	21.9 40	52.0 58	9.95 25	29.1 39	24.3 25	21.2 49	22.2 46	43.3 29	7.82 37	38.1 36
07/25/14	SGBM1	Q	26.4 48	25.5 50	21.3 56	34.5 49	16.1 48	22.3 47	31.5 52	51.6 57	16.2 47	31.7 44	28.8 30	24.0 52	21.2 44	48.0 38	14.8 59	45.4 48
09/14/15	ELAS	F	26.6 49	26.5 51	9.84 28	35.1 51	18.0 50	19.0 44	27.3 48	46.4 47	20.3 50	35.5 48	48.7 50	17.3 34	30.7 58	55.5 62	12.5 51	47.3 51
04/24/16	HLSC_cor	H	26.9 50	18.1 39	24.3 61	42.9 63	15.8 47	15.9 42	30.9 50	40.2 42	27.1 60	33.8 47	45.5 44	22.0 50	22.2 47	51.0 49	13.4 53	37.1 34

Figure 7.2: Screenshot of the estimations of the slanted plane model (PPM) and some other methods in the literature for the training dataset.

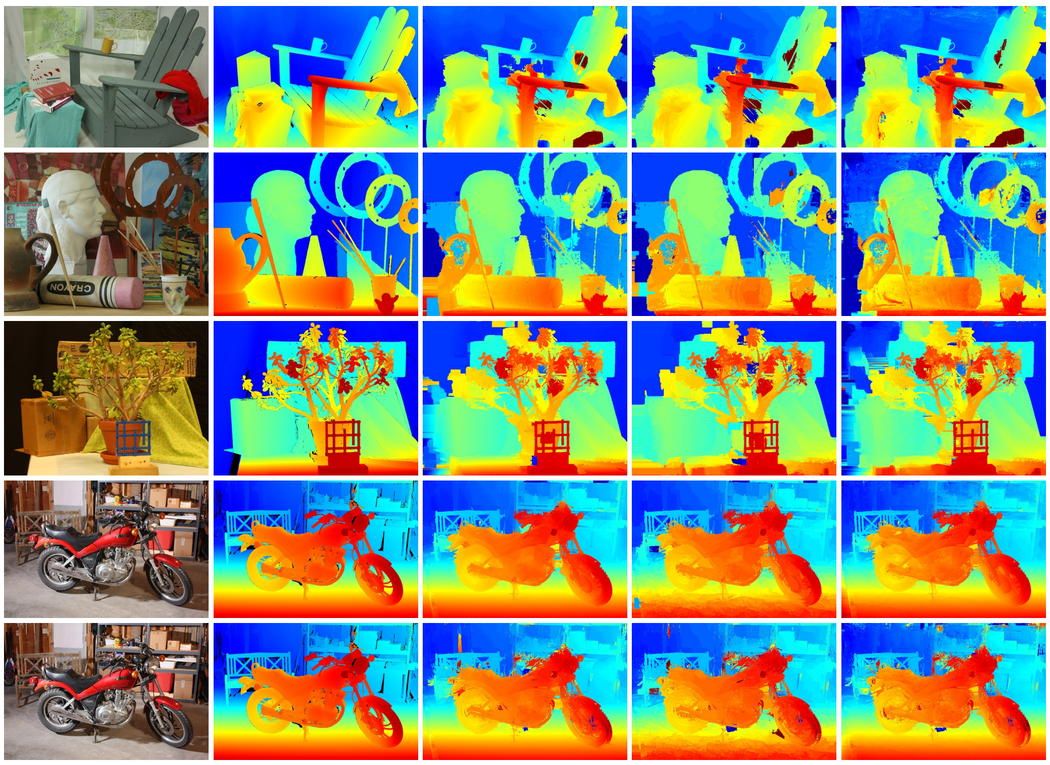


Figure 7.3: Visualization of the estimations of different models for the Middlebury evaluation version 3. **From left to right:** Left view, ground truth, estimations of the slanted plane model, the epipolar constraint model and the projective planar model. **From top to bottom:** Sequences of Adirondack, ArtL, Jadeplant, Motorcycle, MotorcycleE.

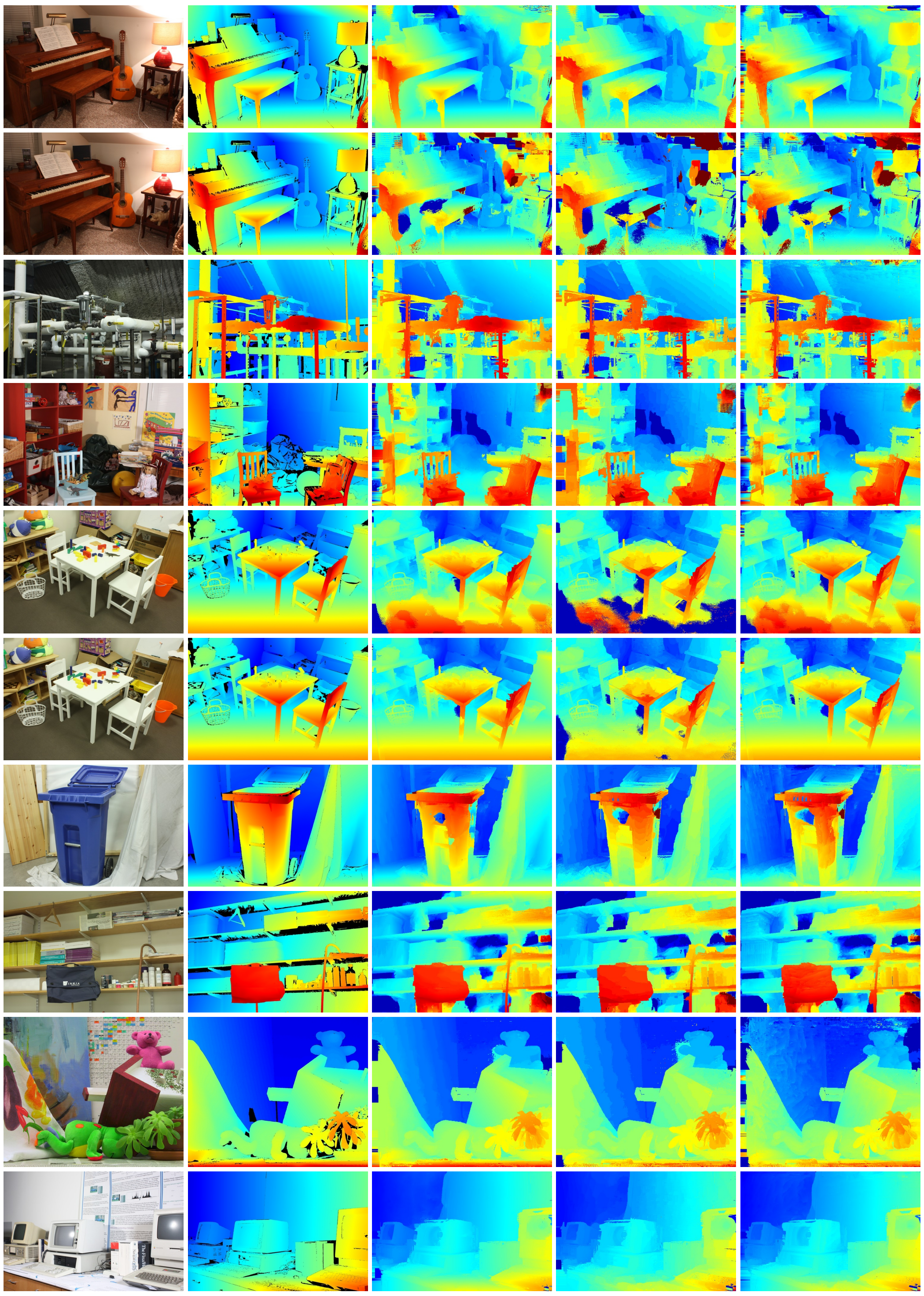


Figure 7.4: Visualization of the estimations of different models for the Middlebury evaluation version 3 (continued). **From left to right:** Left view, ground truth, estimations of the slanted plane model, the epipolar constraint model and the projective planar model. **From top to bottom:** Sequences of Piano, PianoL, Pipes and Playroom, Playtable, PlaytableP, Recycle, Shelves, Teddy and Vintage.

8 Conclusion

In this master thesis, a model parameter estimation method has been presented, which is based on the recently proposed PatchMatch algorithm [BSFG09]. Furthermore, it has been applied to estimate the parameters of some commonly used models in motion estimation and stereo matching. The estimation performance has been evaluated based on the three public benchmarks: Middlebury, KITTI and MPI Sintel.

First some background knowledge about correspondence problems has been introduced in Chapter 2. In Chapter 3 the PatchMatch algorithm in the context of parameter estimation was described. Some commonly used parametric models for motion estimation and stereo matching were presented in Chapter 4 and Chapter 5. Finally, the estimation performance has been evaluated and compared with some other methods in the literature in Chapter 6 and Chapter 7.

Different from the traditional parameter estimation methods based on some optimization algorithms, such as the variational methods, the PatchMatch algorithm tries to find the approximate optimal parameters through a series of random searches and propagations. The evaluation shows the estimation has a better performance in the case of small displacement, such as the Middlebury flow dataset and the Middlebury stereo evaluation version 2. Although the final estimation does not have a high rank in the public benchmarks, it outperforms some classical methods and shows that the PatchMatch algorithm is a promising approach to parameter estimation. In general, if a model satisfies that neighboring pixels have similar model parameters and there is a cost function to evaluate a given set of parameters, PatchMatch can be exploited to estimate its parameters.

8.1 Future Work

Aliasing Handling. The evaluation of the Urban3 sequence from the Middlebury dataset shows that the current implementation of the coarse-to-fine approach performs badly when there exist high frequency variations in the image. Some effort should be made to investigate this issue.

Robust Estimation. Currently it estimates a model for every pixel and is sensitive to noise. Hu et al. [HLS17] has shown that a more robust approach is to divide the image into superpixels and to fit a common model within a superpixel using RANSAC (Random sample consensus) [FB81] based on the current estimations.

Post-Processing. The current method uses the interpolation component and the variational refinement component from the EpicFlow algorithm for post-processing. Hu et al. [HLS17] and Maurer et al. [MSB17] have shown that there are better interpolation and refinement methods for post-processing. Therefore, the post-processing step deserves further investigation to improve the final estimations.

Bibliography

[AAB+84] E. H. Adelson, C. H. Anderson, J. R. Bergen, P. J. Burt, J. M. Ogden. “Pyramid methods in image processing.” In: *Radio Corporation of America Engineer* 29.6 (1984), pp. 33–41 (cit. on p. 20).

[Ana89] P. Anandan. “A computational framework and an algorithm for the measurement of visual motion.” In: *International Journal of Computer Vision* 2.3 (1989), pp. 283–310 (cit. on p. 20).

[BA96] M. J. Black, P. Anandan. “The robust estimation of multiple motions: Parametric and piecewise-smooth flow fields.” In: *Computer Vision and Image Understanding* 63.1 (1996), pp. 75–104 (cit. on p. 20).

[BAHH92] J. R. Bergen, P. Anandan, K. J. Hanna, R. Hingorani. “Hierarchical model-based motion estimation.” In: *Proceedings of the European Conference on Computer Vision*. 1992, pp. 237–252 (cit. on pp. 20, 36, 37).

[Bar89] S. T. Barnard. “Stochastic stereo matching over scale.” In: *International Journal of Computer Vision* 3.1 (1989), pp. 17–32 (cit. on p. 20).

[BBPW04] T. Brox, A. Bruhn, N. Papenberg, J. Weickert. “High accuracy optical flow estimation based on a theory for warping.” In: *Proceedings of the European Conference on Computer Vision*. 2004, pp. 25–36 (cit. on pp. 9, 20).

[BFB94] J. L. Barron, D. J. Fleet, S. S. Beauchemin. “Performance of optical flow techniques.” In: *International Journal of Computer Vision* 12.1 (1994), pp. 43–77 (cit. on pp. 14, 49).

[BFGV17] H. Bilen, B. Fernando, E. Gavves, A. Vedaldi. “Action recognition with dynamic image networks.” In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2017) (cit. on p. 9).

[BM11] T. Brox, J. Malik. “Large displacement optical flow: Descriptor matching in variational motion estimation.” In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33.3 (2011), pp. 500–513 (cit. on pp. 9, 16).

[Bou] J.-Y. Bouguet. *Complete camera calibration toolbox for Matlab*. http://www.vision.caltech.edu/bouguetj/calib_doc/index.html. Accessed November 25, 2017 (cit. on p. 41).

[Bra00] G. Bradski. “The OpenCV Library.” In: *Dr. Dobb’s Journal of Software Tools* (2000) (cit. on p. 41).

[BRR11] M. Bleyer, C. Rhemann, C. Rother. “PatchMatch stereo — Stereo matching with slanted support windows.” In: *Proceedings of the British Machine Vision Conference*. 2011, pp. 1–11 (cit. on pp. 9, 10, 41, 44–46, 62, 63).

[BS10] R. Ben-Ari, N. Sochen. “Stereo matching with Mumford-Shah regularization and occlusion handling.” In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 32.11 (2010), pp. 2071–2084 (cit. on pp. 62, 63).

[BSFG09] C. Barnes, E. Shechtman, A. Finkelstein, D. B. Goldman. “PatchMatch: A randomized correspondence algorithm for structural image editing.” In: *ACM Transactions on Graphics* 28.3 (2009), pp. 24–1 (cit. on pp. 3, 9, 10, 25, 30, 67).

[BSGF10] C. Barnes, E. Shechtman, D. B. Goldman, A. Finkelstein. “The generalized PatchMatch correspondence algorithm.” In: *Proceedings of the European Conference on Computer Vision*. 2010, pp. 29–43 (cit. on p. 9).

[BSL+11] S. Baker, D. Scharstein, J. Lewis, S. Roth, M. J. Black, R. Szeliski. “A database and evaluation methodology for optical flow.” In: *International Journal of Computer Vision* 92.1 (2011), pp. 1–31 (cit. on pp. 10, 49).

[BTS15] C. Bailer, B. Taetz, D. Stricker. “Flow fields: Dense correspondence fields for highly accurate large displacement optical flow estimation.” In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 4015–4023 (cit. on pp. 16, 31, 32, 34).

[BVZ01] Y. Boykov, O. Veksler, R. Zabih. “Fast approximate energy minimization via graph cuts.” In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 23.11 (2001), pp. 1222–1239 (cit. on pp. 62, 63).

[BWS05] A. Bruhn, J. Weickert, C. Schnörr. “Lucas/Kanade meets Horn/Schunck: Combining local and global optic flow methods.” In: *International Journal of Computer Vision* 61.3 (2005), pp. 211–231 (cit. on p. 9).

[BWSB12] D. J. Butler, J. Wulff, G. B. Stanley, M. J. Black. “A naturalistic open source movie for optical flow evaluation.” In: *Proceedings of the European Conference on Computer Vision*. 2012, pp. 611–625 (cit. on pp. 10, 49).

[BYJ14] L. Bao, Q. Yang, H. Jin. “Fast edge-preserving PatchMatch for large displacement optical flow.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 3534–3541 (cit. on p. 10).

[CGN14] H. Chao, Y. Gu, M. Napolitano. “A survey of optical flow techniques for robotics navigation applications.” In: *Journal of Intelligent & Robotic Systems* 73.1 (2014), pp. 361–372 (cit. on p. 9).

[CHY10] Y.-T. Chi, J. Ho, M.-H. Yang. “A direct method for estimating planar projective transform.” In: *Proceedings of the Asian Conference on Computer Vision*. 2010, pp. 268–281 (cit. on p. 37).

[CK16] Q. Chen, V. Koltun. “Full flow: Optical flow estimation by global optimization over regular grids.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 4706–4714 (cit. on p. 58).

[CS11] B. Cyganek, J. P. Siebert. *An introduction to 3D computer vision techniques and algorithms*. John Wiley & Sons, 2011 (cit. on pp. 9, 12, 42).

[Dem15] O. Demetz. “Feature invariance versus change estimation in variational motion estimation.” PhD thesis. Universität des Saarlandes, 2015 (cit. on pp. 16–19, 39).

[DHW13] O. Demetz, D. Hafner, J. Weickert. “The complete rank transform: A tool for accurate and morphologically invariant matching of structures.” In: *Proceedings of the British Machine Vision Conference*. 2013 (cit. on p. 16).

[DN13] M. Drulea, S. Nedevschi. “Motion estimation using the correlation transform.” In: *IEEE Transactions on Image Processing* 22.8 (2013), pp. 3260–3270 (cit. on p. 54).

[DSV+14] O. Demetz, M. Stoll, S. Volz, J. Weickert, A. Bruhn. “Learning brightness transfer functions for the joint recovery of illumination changes and optical flow.” In: *Proceedings of the European Conference on Computer Vision*. 2014, pp. 455–471 (cit. on p. 58).

[EGA+17] H. J. Escalante, I. Guyon, V. Athitsos, P. Jangyodsuk, J. Wan. “Principal motion components for one-shot gesture recognition.” In: *Pattern Analysis and Applications* 20.1 (2017), pp. 167–182 (cit. on p. 9).

[FB81] M. A. Fischler, R. C. Bolles. “Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography.” In: *Communications of the ACM* 24.6 (1981), pp. 381–395 (cit. on p. 67).

[FBK15] D. Fortun, P. Bouthemy, C. Kervrann. “Optical flow modeling and computation: A survey.” In: *Computer Vision and Image Understanding* 134 (2015), pp. 1–21 (cit. on p. 20).

[FH06] P. F. Felzenszwalb, D. P. Huttenlocher. “Efficient belief propagation for early vision.” In: *International Journal of Computer Vision* 70.1 (2006), pp. 41–54 (cit. on p. 20).

[FJ90] D. J. Fleet, A. D. Jepson. “Computation of component image velocity from local phase information.” In: *International Journal of Computer Vision* 5.1 (1990), pp. 77–104 (cit. on p. 49).

[FTV00] A. Fusiello, E. Trucco, A. Verri. “A compact algorithm for rectification of stereo pairs.” In: *Machine Vision and Applications* 12.1 (2000), pp. 16–22 (cit. on pp. 12, 41).

[GGN13] B. Girod, G. Greiner, H. Niemann. *Principles of 3D image analysis and synthesis*. Springer Science & Business Media, 2013 (cit. on p. 36).

[Gia00] A. Giachetti. “Matching techniques to compute image motion.” In: *Image and Vision Computing* 18.3 (2000), pp. 247–260 (cit. on p. 15).

[GLSG10] D. Geronimo, A. M. Lopez, A. D. Sappa, T. Graf. “Survey of pedestrian detection for advanced driver assistance systems.” In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 32.7 (2010), pp. 1239–1258 (cit. on p. 9).

[GLU12] A. Geiger, P. Lenz, R. Urtasun. “Are we ready for autonomous driving? The KITTI vision benchmark suite.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2012, pp. 3354–3361 (cit. on pp. 10, 49).

[GW17] R. C. Gonzalez, R. E. Woods. *Digital image processing*. Pearson, 2017 (cit. on pp. 20, 27).

[GZG+10] P. Gwosdek, H. Zimmer, S. Grewenig, A. Bruhn, J. Weickert. “A highly efficient GPU implementation for variational optic flow based on the Euler-Lagrange framework.” In: *Proceedings of the European Conference on Computer Vision*. 2010, pp. 372–383 (cit. on p. 9).

[Hir08] H. Hirschmuller. “Stereo processing by semiglobal matching and mutual information.” In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 30.2 (2008), pp. 328–341 (cit. on pp. 62, 63).

[HLS17] Y. Hu, Y. Li, R. Song. “Robust interpolation of correspondences for large displacement optical flow.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 4791–4799 (cit. on pp. 10, 31, 67, 68).

[HRB+13] A. Hosni, C. Rhemann, M. Bleyer, C. Rother, M. Gelautz. “Fast cost-volume filtering for visual correspondence and beyond.” In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.2 (2013), pp. 504–511 (cit. on pp. 45, 54).

[HS81] B. K. Horn, B. G. Schunck. “Determining optical flow.” In: *Artificial Intelligence* 17.1 (1981), pp. 185–203 (cit. on p. 9).

[HSL16] Y. Hu, R. Song, Y. Li. “Efficient coarse-to-fine PatchMatch for large displacement optical flow.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 5704–5712 (cit. on pp. 9, 10, 16, 31, 32, 34).

[HZ03] R. Hartley, A. Zisserman. *Multiple view geometry in computer vision*. Cambridge University Press, 2003 (cit. on pp. 9, 37, 41, 43).

[Liu09] C. Liu. “Beyond pixels: Exploring new representations and applications for motion analysis.” PhD thesis. Massachusetts Institute of Technology, 2009 (cit. on p. 20).

[LK81] B. D. Lucas, T. Kanade. “An iterative image registration technique with an application to stereo vision.” In: *Proceedings of the International Joint Conference on Artificial Intelligence*. 1981, pp. 674–679 (cit. on p. 9).

[LMB+15] Y. Li, D. Min, M. S. Brown, M. N. Do, J. Lu. “SPM-BP: Sped-up PatchMatch belief propagation for continuous MRFs.” In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 4006–4014 (cit. on p. 58).

[Low04] D. G. Lowe. “Distinctive image features from scale-invariant keypoints.” In: *International Journal of Computer Vision* 60.2 (2004), pp. 91–110 (cit. on p. 16).

[LYT] C. Liu, J. Yuen, A. Torralba. *SIFT flow: Dense correspondence across scenes and its applications*. https://people.csail.mit.edu/celiu/SIFTflow/. Accessed November 25, 2017 (cit. on pp. 16, 19).

[LYT11] C. Liu, J. Yuen, A. Torralba. "SIFT flow: Dense correspondence across scenes and its applications." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33.5 (2011), pp. 978–994 (cit. on pp. 16, 17).

[MCUP04] J. Matas, O. Chum, M. Urban, T. Pajdla. "Robust wide-baseline stereo from maximally stable extremal regions." In: *Image and Vision Computing* 22.10 (2004), pp. 761–767 (cit. on p. 16).

[MG15] M. Menze, A. Geiger. "Object scene flow for autonomous vehicles." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 3061–3070 (cit. on pp. 10, 49).

[MHG15] M. Menze, C. Heipke, A. Geiger. "Discrete optimization for optical flow." In: *Proceedings of the German Conference on Pattern Recognition*. 2015, pp. 16–28 (cit. on p. 58).

[MP97] S. Mann, R. W. Picard. "Video orbits of the projective group: A simple approach to featureless estimation of parameters." In: *IEEE Transactions on Image Processing* 6.9 (1997), pp. 1281–1295 (cit. on p. 37).

[MS05] K. Mikolajczyk, C. Schmid. "A performance evaluation of local descriptors." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 27.10 (2005), pp. 1615–1630 (cit. on p. 17).

[MSB17] D. Maurer, M. Stoll, A. Bruhn. "Order-adaptive and illumination-aware variational optical flow refinement." In: *Proceedings of the British Machine Vision Conference*. 2017 (cit. on pp. 31, 68).

[MSKS12] Y. Ma, S. Soatto, J. Kosecka, S. S. Sastry. *An invitation to 3-D vision: From images to geometric models*. Springer Science & Business Media, 2012 (cit. on pp. 41, 43).

[ON94] M. Otte, H.-H. Nagel. "Optical flow estimation: Advances and comparisons." In: *Proceedings of the European Conference on Computer Vision*. 1994, pp. 49–60 (cit. on p. 49).

[PBB+06] N. Papenberg, A. Bruhn, T. Brox, S. Didas, J. Weickert. "Highly accurate optic flow computation with theoretically justified warping." In: *International Journal of Computer Vision* 67.2 (2006), pp. 141–158 (cit. on p. 9).

[RWHS15] J. Revaud, P. Weinzaepfel, Z. Harchaoui, C. Schmid. "EpicFlow: Edge-preserving interpolation of correspondences for optical flow." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 1164–1172 (cit. on pp. 31, 34, 58).

[RWHS16] J. Revaud, P. Weinzaepfel, Z. Harchaoui, C. Schmid. "Deepmatching: Hierarchical deformable dense matching." In: *International Journal of Computer Vision* 120.3 (2016), pp. 300–323 (cit. on p. 58).

[SBM06] Z. Sun, G. Bebis, R. Miller. “On-road vehicle detection: A review.” In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 28.5 (2006), pp. 694–711 (cit. on p. 9).

[SHK+14] D. Scharstein, H. Hirschmüller, Y. Kitajima, G. Krathwohl, N. Nešić, X. Wang, P. Westling. “High-resolution stereo datasets with subpixel-accurate ground truth.” In: *Proceedings of the German Conference on Pattern Recognition*. 2014, pp. 31–42 (cit. on pp. 12, 61).

[SK99] C. Stiller, J. Konrad. “Estimating motion in image sequences.” In: *IEEE Signal Processing Magazine* 16.4 (1999), pp. 70–91 (cit. on p. 36).

[SRB10] D. Sun, S. Roth, M. J. Black. “Secrets of optical flow estimation and their principles.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2010, pp. 2432–2439 (cit. on pp. 9, 20).

[SS02] D. Scharstein, R. Szeliski. “A taxonomy and evaluation of dense two-frame stereo correspondence algorithms.” In: *International Journal of Computer Vision* 47.1 (2002), pp. 7–42 (cit. on pp. 10, 12–14, 19, 20, 22, 47, 49, 61).

[STR+13] K. Schmid, T. Tomic, F. Ruess, H. Hirschmüller, M. Suppa. “Stereo vision based indoor/outdoor navigation for flying robots.” In: *International Conference on Intelligent Robots and Systems*. 2013, pp. 3955–3962 (cit. on p. 9).

[Str16] G. Strang. *Introduction to linear algebra*. Wellesley-Cambridge Press, 2016 (cit. on p. 19).

[SVB12] M. Stoll, S. Volz, A. Bruhn. “Adaptive integration of feature matches into variational optical flow methods.” In: *Proceedings of the Asian Conference on Computer Vision*. 2012, pp. 1–14 (cit. on pp. 16, 54).

[SVMB17] M. Stoll, S. Volz, D. Maurer, A. Bruhn. “A time-efficient optimisation framework for parameters of optical flow methods.” In: *Proceedings of the Scandinavian Conference on Image Analysis*. 2017, pp. 41–53 (cit. on p. 50).

[SZ02] F. Schaffalitzky, A. Zisserman. “Multi-view matching for unordered image sets, or “How do I organize my holiday snaps?”” In: *Proceedings of the European Conference on Computer Vision*. 2002, pp. 414–431 (cit. on p. 16).

[Sze10] R. Szeliski. *Computer vision: Algorithms and applications*. Springer Science & Business Media, 2010 (cit. on p. 12).

[THZ82] R. Tsai, T. Huang, W.-L. Zhu. “Estimating three-dimensional motion parameters of a rigid planar patch, ii: Singular value decomposition.” In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 30.4 (1982), pp. 525–534 (cit. on p. 37).

[TLF10] E. Tola, V. Lepetit, P. Fua. “DAISY: An efficient dense descriptor applied to wide baseline stereo.” In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 32.5 (2010), pp. 815–830 (cit. on p. 16).

[TM98] C. Tomasi, R. Manduchi. “Bilateral filtering for gray and color images.” In: *Proceedings of the IEEE International Conference on Computer Vision*. 1998, pp. 839–846 (cit. on p. 46).

[TUI17] T. Taketomi, H. Uchiyama, S. Ikeda. “Visual SLAM algorithms: A survey from 2010 to 2016.” In: *IPSJ Transactions on Computer Vision and Applications* 9.1 (2017), p. 16 (cit. on p. 9).

[UIM08] H. Uemura, S. Ishikawa, K. Mikolajczyk. “Feature tracking and motion compensation for action recognition.” In: *Proceedings of the British Machine Vision Conference*. 2008, pp. 1–10 (cit. on p. 9).

[VSR13] C. Vogel, K. Schindler, S. Roth. “Piecewise rigid scene flow.” In: *Proceedings of the IEEE International Conference on Computer Vision*. 2013, pp. 1377–1384 (cit. on p. 9).

[WPZ+09] A. Wedel, T. Pock, C. Zach, H. Bischof, D. Cremers. “An improved algorithm for TV-L1 optical flow.” In: *Statistical and Geometrical Approaches to Visual Motion Analysis*. 2009, pp. 23–45 (cit. on p. 54).

[WRHS13] P. Weinzaepfel, J. Revaud, Z. Harchaoui, C. Schmid. “DeepFlow: Large displacement optical flow with deep matching.” In: *Proceedings of the IEEE International Conference on Computer Vision*. 2013, pp. 1385–1392 (cit. on pp. 16, 31, 58).

[WTP+09] M. Werlberger, W. Trobin, T. Pock, A. Wedel, D. Cremers, H. Bischof. “Anisotropic Huber-L1 optical flow.” In: *Proceedings of the British Machine Vision Conference*. 2009, p. 3 (cit. on p. 54).

[YL15] J. Yang, H. Li. “Dense, accurate optical flow estimation with piecewise parametric model.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 1019–1027 (cit. on p. 37).

[YWA10] Q. Yang, L. Wang, N. Ahuja. “A constant-space belief propagation algorithm for stereo matching.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2010, pp. 1458–1465 (cit. on pp. 20, 62, 63).

[Zay16] A. Zayouna. “Optical flow estimation using steered-L1 norm.” PhD thesis. Middlesex University, 2016 (cit. on p. 54).

[ZBW11] H. Zimmer, A. Bruhn, J. Weickert. “Optic flow in harmony.” In: *International Journal of Computer Vision* 93.3 (2011), pp. 368–388 (cit. on p. 34).

[ZDL17] D. Zhou, Y. Dai, H. Li. “Pixel-variant local homography for fisheye stereo rectification minimizing resampling distortion.” In: *Computing Research Repository* abs/1707.03775 (2017). arXiv: 1707.03775 (cit. on p. 42).

[Zha00] Z. Zhang. “A flexible new technique for camera calibration.” In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22.11 (2000), pp. 1330–1334 (cit. on p. 41).

[ZW94] R. Zabih, J. Woodfill. “Non-parametric local transforms for computing visual correspondence.” In: *Proceedings of the European Conference on Computer Vision*. 1994, pp. 151–158 (cit. on p. 16).

All links were last followed on November 25, 2017.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

place, date, signature